Activities for CS1 in Python

Instructor's Guide with Solutions

Tricia Shepherd, Chris Mayfield, Helen Hu

With contributions from Michael Stewart, Theresa Wilson, and Barbara Wahl

Fall 2019

Copyright © 2019. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

Source documents available in Overleaf

Contents

Introduction to Python

In this course, you will work in teams of 3–4 students to learn new concepts. This activity will introduce you to the process. We'll take a first look at variables, assignment, and input/output.

Content Learning Objectives

After completing this activity, students should be able to:

- Describe differences between program and output text.
- Identify and execute Python functions for input/output.
- Write assignment statements and use assigned variables.

Process Skill Goals

During the activity, students should make progress toward:

- Leveraging prior knowledge and experience of other students. (Teamwork)

Facilitation Notes

During the first 5–10 minutes, help teams get started by encouraging discussion and collaboration. Make sure that each team is able to find Thonny (or other development environment) on their computer. If multiple interpreters are installed (e.g., Python 2 and Python 3), double check that students run the correct version.

Keep a close watch on the first few questions, and encourage teams that make mistakes to go back and rework them (without giving the answer). Once most teams have successfully run the program in Model 1, type or open the same code on the instructor's machine. Report out questions #2 and #4, using the projected program as a visual aid.

On Model 2, make sure that teams use the Shell window (some teams might accidentally use the Editor window). Once most teams have started working on the table, you might want to demonstrate on the projector how the Shell works. Report out the last three questions, and discuss how values from user input are different from literal values in the source code.

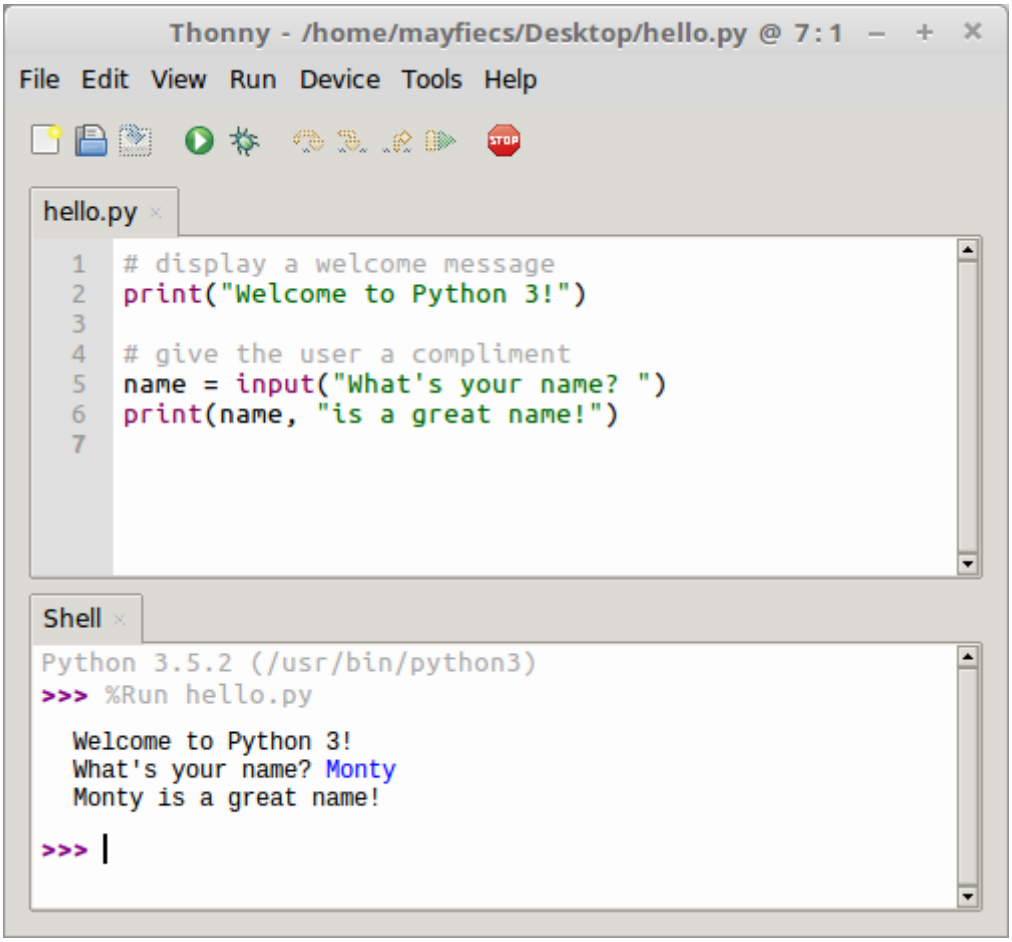
Before students begin Model 3, demonstrate how to read error messages. They do not need to write down the entire error message, but they should identify the type of error (e.g., NameError, SyntaxError). It may help to make several mistakes on the projector and show students how to find the first word at the end of the error message.

At the end of the activity, review how variables work in the Shell. Demonstrate and explain the following lines: `x = 10`, `x` (by itself), `x + 1`, `x` (by itself), `x = x + 1`, and `x` (by itself).


Copyright © 2019 T. Shepherd, C. Mayfield, and H. Hu. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

Model 1 Getting Started with Thonny

Thonny is an integrated development environment (IDE) for Python designed for learning and teaching programming. It is freely available at https://thonny.org/.



Do not run Thonny yet! Answer the questions first!

Questions (15 min) Start time: _____

1. Based on the screenshot in Model 1:

   a) where is the Shell window? the second / bottom half

   b) where is the Editor window? the first / top half

   c) what is the name of the file in the Editor? hello.py

   d) what is the directory where this file is located? Desktop

2. Identify the number of lines corresponding to:

   a) the program (in the Editor)
      The source file contains six total lines (the 7th line is blank and doesn't count)

   b) the output of the program
      The program displays three lines ("Welcome...", "What's...", and "Monty...").

3. What is the symbol at the start of a line of program text not displayed as output?

The pound sign: #

4. Consider the three program lines (in the Editor) that are not displayed as output. Describe what might be the purpose of:

   a) a ***comment*** line (starts with a pound sign: #)
      They are used to describe the purpose and/or behavior of the code.

   b) a blank line
      They make the code easier to read.

Now open Thonny on your computer, type the code shown in Model 1, save the file as hello.py, and run the program. Ask for help if you get stuck!

5. What was required before the third line of the program output was displayed?

An answer needed to be entered (typed).

6. In the Shell window, what is the color of:

   a) the program's output? black

   b) the user's input? blue

7. Based on your experience so far, what is the difference between the text in the Editor window and the text in the Shell window?

The editor contains a series of statements or commands which make up the program. The shell (or terminal) contains the program output text and an interface for program input.

8. Describe what appears to be the purpose of each line of Python code in the Editor window.

   a) line 1: a comment to describe the purpose of the code

   b) line 2: displays a message

   c) line 3: a blank line to make the code easier to read

   d) line 4: a comment to describe the purpose of the code

   e) line 5: gets input from the user

   f) line 6: displays the final output

Model 2 Python Built-In Functions

You can use built-in Python ***functions*** to perform specific operations. Sometimes a function will require information (referred to as ***arguments***) to perform its operation. A function will also ***return*** a result after the operation.

To ***call*** (or use) a Python function:

- You must include parentheses after the function's name (e.g., `print()` prints a blank line).
- If the function takes one or more arguments to perform its operation, you must put that information in the parentheses (e.g., `print("Hello, world!")` prints a message).

Do not type anything yet! Read the questions first!

Python code	Shell output
`input("enter the mass in grams:  ")`	enter the mass in grams: 100
`mass = input("enter another mass in grams:  ")`	enter another mass in grams: 10
`mass`	'10'
`unit = input("enter the units for mass:  ")`	enter the units for mass: g
`print(mass, unit)`	10 g
`print(mass / 2)`	TypeError
`ten = 10`	
`print(ten / 2)`	5.0
`abs(-1)`	1
`abs(-1 * ten)`	10

Questions (15 min) Start time: _____

9. List the names of the three functions used in Model 2.

input, print, abs

10. What are the arguments of the first use of the `print` function? _____ mass, unit

11. Type each line of code in a Python Shell, one line at a time, and write the corresponding output (if observed) in the right column of the table. If an error occurs, write what type of error it was (i.e., the first word of the last line of the error message).

Place an asterisk (*) next to any output for which you were surprised, and note what was unexpected about the output. Don't worry yet about *understanding* any strange output you may see; we will discuss what it all means by the end of class.

12. Which function delayed execution until additional input was entered?

The input function.

13. Which term, ***user*** or ***programmer***, best defines the role of the person who entered the additional input? Explain.

User is a better term; programs are ultimately written to be used by non-programmers.

14. Based on the Shell output, what does the word `mass` represent, and how did it get its value?

(Answers may vary) Its value is 10, which the user entered as input.

15. What does the word `ten` represent, and how did it get its value?

Its value is 10 based on the statement "ten = 10".

16. Do the values of `mass` and `ten` both represent a number? Explain why or why not.

Although they both appear to represent a numerical value, `mass` divided by 2 gives an error, while `five` divided by 2 gives the expected numerical outcome.

Model 3 Variables and Assignment

In programming, an ***assignment statement*** saves a value to a ***variable***. The variable "is set to" the value after the = ***operator***. Selecting concise yet descriptive variable names is considered good programming style and will make your programs easier to read.

Do not type anything yet! Read the questions first!

Python code	Shell output
`data = 12`	
`data`	12
`Data`	NameError
`Data = 34`	
`data`	12
`Data`	34
`my data = 56`	SyntaxError
`my_data = 78`	
`3data = "hello"`	SyntaxError
`data3 = "world"`	
`data3 = hello`	NameError
`hot = 273 + 100`	
`273 + 100 = hot`	SyntaxError
`hot`	373
`Hot + 100`	NameError
`hot - 100`	273

Questions (15 min) Start time: _____

17. Based on the information and Python code in Model 3, give an example representing each of the following:

a) an assignment statement data = 12

b) the variable being assigned data

c) the assignment operator =

d) the value of the variable immediately after the assignment 12

18. Similar to Model 2, type each line of code in a Python Shell and write the corresponding output in the space above. If an error occurs, write what type of error. Place an asterisk (*) next to any output for which you were surprised.

19. Circle each *successful* assignment statement in Model 3. How many are there? _____ 5

20. What is the observed output of a successful assignment statement?

There is no error message (there is no output at all).

21. After the successful execution of an assignment statement, how can you confirm the value of this variable?

You can type back the variable name to see its current value.

22. For each assignment statement that executed without an error, write the corresponding variable name.

data, Data, my_data, data3, hot
(see the lines with no output)

23. Based on the Model 3 output, indicate whether each statement below is true or false.

a) Variable names in Python can start with a number. _____ false

b) Variable names in Python must start with a lower-case letter. _____ false

c) Variable names in Python may not include spaces. _____ true

d) Variable names in Python are case-sensitive. _____ true

24. Each of the following assignment statements has an error. Write a valid line of Python code that corrects the assignment statement. Double-check your code using a computer.

a) `3 + 4 = answer` answer = 3 + 4

b) `oh well = 3 + 4` oh_well = 3 + 4

c) `2x = 7` x2 = 7

25. Predict the value of the variable `hot` after executing all lines of code in Model 3. Then test your prediction on a computer, and explain the result.

The value is 373, because `hot` was only assigned one time. The other lines did not change the value of `hot`.

26. Write a line of Python code to assign the current value of `hot` to the variable `cold`. Show output that confirms that you have done this correctly, and explain the code.

The correct line is `cold = hot`. To verify the result, simply type `cold`. The assignment operation reads from right to left.

Arithmetic Expressions

Now that you've written a few programs, let's take a step back and discuss how to do arithmetic. The behavior of Python operators (+, -, *, /) depends on what type of data you have.

Content Learning Objectives

After completing this activity, students should be able to:

- Execute mathematical expressions similar to a calculator.
- Describe the function of the three Python division operators.
- Explain differences between integer and floating-point data.

Process Skill Goals

During the activity, students should make progress toward:

- Recognizing mathematical operations based on tables. (Information Processing)

Facilitation Notes

Help students get started quickly on the table in Model 1, so that they will have enough time to answer the questions. When reporting out, ask teams what surprises they had and what they learned from interacting with the code using a Shell.

If you are using Thonny (see Activity 1), now is a good opportunity to explain the "Assistant" that opens when errors occur. It may also be helpful to visualize assignment statements using the debugger. Thonny will even step into expressions, showing the order of operations. Have students check their work as you demonstrate the order and result of each expression.

On Model 2, report out after #15. Introduce the terms ***modulo operation*** (and/or ***modulus***) and ***remainder operator***. Explain that "x % y" can be pronounced "x mod y". #16 refers to the instructor hypothetically giving mints to teams to divide among themselves. Consider bringing real mints to build rapport with the class. Or you can use other small objects, like paperclips.

During Model 3, be aware of the difference between `**`, `pow`, and `math.pow`. The ** operator and built-in `pow` function preserve the data type of the operands: `5 ** 2` is 25. In contrast, `math.pow` always uses floating-point arithmetic: `math.pow(5, 2)` is 25.0. If students use `math.pow` on #24, they'll get the incorrect answer if it overflows. You can either warn students about `math.pow` in advance, or have students discover this issue when reporting out.


Copyright © 2019 T. Shepherd, C. Mayfield, and H. Hu. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

Model 1 Python Calculator

In a Python Shell window, ">>>" is a ***prompt*** indicating that the interpreter is waiting for input. All text entered after the prompt will be executed immediately as Python code.

If you type a Python ***expression*** (code that results in a value) after the prompt, Python will show the value of that expression, similar to a calculator. You can use Python's `math` module to perform more complex mathematical operations like logarithms and trigonometric operations.

Do not type anything yet! Read the questions first!

Python code	Predicted output	Actual output
`2 + 3`		5
`3 * 4 + 2`		14
`3 * 4 + 2.0`		14.0
`3(4 + 2)`		TypeError
`3 * (4 + 2)`		18
`5 / 10`		0.5
`5 / 10.0`		0.5
`5 / 9`		0.5555556
`2 ** 4`		16
`abs(-2) ** 4`		16
`math.pow(2, 4)`		NameError
`import math`		
`math.pow(2, 4)`		1.0
`sqrt(4)`		NameError
`math.sqrt(4)`		2.0
`math.cos(0)`		1.0
`math.pi`		3.141592653589793
`math.sin(math.pi / 2)`		1.0

Questions (15 min) Start time: _____

1. In the middle "Predicted output" column, write what value you expect will be displayed, based on your team's experience using a calculator. If there are any lines you are not confident about, place an asterisk next to your predicted output.

2. Open a Python Shell on your computer. Type each Python expression at the prompt, one line at a time, and write the corresponding Python output in the third column above. If an error occurs, write what type of error it was (i.e., the first word of the last line of the error message).

3. What does the ** operator do?

It raises a number to a power.

4. Based on the Python code in Model 1, identify four examples of:

a) mathematical operator +, *, /, **

b) mathematical function abs, math.pow, math.sqrt, math.cos, math.sin

5. For addition and multiplication to produce an output with a decimal value, what type of number must be part of the input? Provide justification for your team's answer.

At least one of the numbers must have a decimal value. For example, `3 * 4 + 2` is the integer value 14, but `3 * 4 + 2.0` is the decimal value 14.0.

6. Does division follow the same rule as in #5? Provide justification for your team's answer.

No; when dividing integers, the result is always a decimal number. The same is true even when there is no remainder, i.e., 8 / 4 is 2.0.

7. The output of Model 1 displayed three different errors. Explain the reason for each:

a) TypeError need to use * to multiply [1]

b) 1st NameError need to import math

c) 2nd NameError need "math." before function

8. Identify two differences between using a Python built-in function (e.g., `abs`) and a function from the `math` module.

Need to "`import math`" first, and all function names start with "`math.`" before the function.

[1] When students enter `3(4 + 2)` the interpreter says, "TypeError: 'int' object is not callable". In other words, the integer 3 is not a function and can't be called.

Model 2 Dividing Numbers

Table A

`9 / 4`	*evaluates to*	2.25
`10 / 4`	*evaluates to*	2.5
`11 / 4`	*evaluates to*	2.75
`12 / 4`	*evaluates to*	3.0
`13 / 4`	*evaluates to*	3.25
`14 / 4`	*evaluates to*	3.5
`15 / 4`	*evaluates to*	3.75
`16 / 4`	*evaluates to*	4.0

Table B

`9 // 4`	*evaluates to*	2
`10 // 4`	*evaluates to*	2
`11 // 4`	*evaluates to*	2
`12 // 4`	*evaluates to*	3
`13 // 4`	*evaluates to*	3
`14 // 4`	*evaluates to*	3
`15 // 4`	*evaluates to*	3
`16 // 4`	*evaluates to*	4

Table C

`9 % 4`	*evaluates to*	1
`10 % 4`	*evaluates to*	2
`11 % 4`	*evaluates to*	3
`12 % 4`	*evaluates to*	0
`13 % 4`	*evaluates to*	1
`14 % 4`	*evaluates to*	2
`15 % 4`	*evaluates to*	3
`16 % 4`	*evaluates to*	0

Questions (15 min)

Start time: _____

9. For each operator in Model 2, identify the symbol and describe the type of numerical result.

/ decimal // integer % integer

Note that Python refers to decimal numbers as "floating-point" numbers.

10. If the result of the / operator were rounded to the nearest integer, would this be the same as the result of the // operator? Explain how the results in Table A compare to Table B.

No, the pattern is off by two rows. The 0.5 and 0.75 values would round up, but in the second table they round down.

11. If the table included more rows, list all numbers // 4 would evaluate to 2 and all the numbers // 4 would evaluate to 4.

8, 9, 10, and 11 evaluate to 2.
16, 17, 18, and 19 evaluate to 4.

12. Based on the results of Table C, propose another number % 4 evaluates to 0, and explain what all these numbers have in common.

Other numbers include 0, 4, 8, 20, 24. All of these numbers are multiples of four.

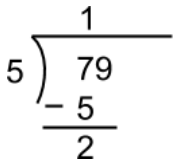
13. Consider the expressions in Table C that evaluate to 1. How do the left ***operands*** in these expressions (i.e., 9, 13) differ from those that evaluate to 0?

They each differ by one; they are one higher than a multiple of four.

14. Describe the reason for the repeated sequence of numbers (0, 1, 2, 3) for the result of % 4.

The difference (remainder) increases by one until the number is exactly divisible by 4.

15. Recall how you learned to do long division in elementary school. Finish solving for $79 \div 5$ below. Which part of the answer is `79 // 5`, and which part is `79 % 5`?



We first bring the 9 down, then 5 goes into 29 five times, and so we subtract 25. The final answer is 15 remainder 4. So `79 // 5` is 15, and `79 % 5` is 4.

16. Imagine that you are given candy mints to divide evenly among your team members.

a) If your team receives 11 mints, how many mints would each student get, and how many are left over? Write a Python expression to compute each result.

`11 / 3` is 3 and `11 % 3` is 2 or `11/ 4` is 2 and `11 % 4` is 3

b) If your team receives 2 mints, how many mints would each student get, and how many are left over? Write a Python expression to computes this result.

`2 / 3` is 0 and `2 % 3` is 2 or `2 / 4` is 0 and `2 % 4` is 2

17. Python has three division operators: "floor division", "remainder", and "true division". Which operator (symbol) corresponds to each name?

`//` is floor division, because it throws away the decimal place (i.e., it "floors" the result). `%` is the remainder operator, which is sometimes called the modulo operator. `/` is true division, because it gives you the mathematically correct answer.

Model 3 Integers and Floats

Every value in Python has a ***data type*** which determines what can be done with the data. Enter the following code, one line at a time, into a Python Shell. Record the output for each line (if any) in the second column.

Python code	Shell output
`integer = 3`	
`type(integer)`	<class 'int'>
`type("integer")`	<class 'str'>
`pi = 3.1415`	
`type(pi)`	<class 'float'>
`word = str(pi)`	
`word`	'3.1415'
`number = float(word)`	
`print(word * 2)`	3.14153.1415
`print(number * 2)`	6.283
`print(word + 2)`	TypeError
`print(number + 2)`	5.14159
`euler = 2.7182`	
`int(euler)`	2
`round(euler)`	3

Questions (15 min) Start time: _____

18. What is the data type (`int`, `float`, or `str`) of the following values? (Note: if you're unsure, use the `type` function in a Python Shell.)

a) pi float

b) integer int

c) word str

d) number float

19. List the function calls that convert a value to a new data type.

The calls are: `str(pi)`, `float(word)`, and `int(euler)`. Note there is a function named after each data type.

20. How does the behavior of the operators (+ and *) depend on the data type?

The + operator appends text, and the * operator copies text.

21. What is the difference between the `int` function and the `round` function?

The `int` function truncates the value, throwing away the decimal places. The `round` function rounds the value up or down to the nearest integer.

22. What is the value of `3 + 3 + 3`? What is the value of `.3 + .3 + .3`? If you enter these expressions into a Python Shell, what do you notice about the results?

Adding three 3's is 9, but adding three .3's is 0.8999999999999999. The answer is slightly off when using floating-point numbers.

23. In order to store a number with 100% accuracy, what data type is required? How might you precisely represent a bank account balance of $123.45?

Integers should be used to avoid floating-point errors. Simply multiply the balance by 100 when storing, and divide by 100 when displaying. $123.45 would be represented as 12345 cents.

24. Try calculating a very large integer in a Python Shell, for example, 123^{456}. Is there a limit to the integers that Python can handle?

There is no limit, other than the computer's memory size. But the larger the integer, the longer it takes to compute it.

25. Try calculating a very large floating-point number in a Python Shell, for example, 123.0^{465}. Is there a limit to the floating-point numbers that Python can handle?

Yes; at some point the numbers get too big. For example, `123.0 ** 456` results with an OverflowError: 'Numerical result out of range'.

26. Summarize the difference between the numeric data types (`int` and `float`). What are their pros and cons?

Integers have unlimited range and precision, but floating-point numbers are an approximation. (Note: `float` in Python is usually implemented using `double` in C.)

Basic Data Structures

Python has a wide variety of built-in types for storing anything from numbers and text (e.g., `int`, `float`, `str`) to common data structures (e.g., `list`, `tuple`, `dict`).

Content Learning Objectives

After completing this activity, students should be able to:

- Reference a specific element of a sequence by an index.
- Compare and contrast numeric and sequence data types.
- Create a dictionary of strings and look up values by key.

Process Skill Goals

During the activity, students should make progress toward:

- Providing feedback on how well other team members are working. (Teamwork)

Facilitation Notes

All three models involve completing tables interactively using a Python Shell. Encourage teams to complete them as quickly as possible, without thinking about individual results. The questions are designed for students to go back and interpret the results as a whole. If they spend too long discussing the table, they won't finish the questions.

The questions on Model 1 are short and may not lead to much whole-class discussion. Have neighboring teams compare answers and discuss any differences that arise. Some students may be aware of negative indexes, but you may want to avoid them at this point. A future activity will explore more advanced topics with indexing and slicing.

On Model 2, students might ask when to use a tuple instead of a list. Avoid long discussions about mutability, and instead focus on everyday examples. Some data are inherently lists: class rosters, grocery lists, etc. Other data are inherently tuples: coordinates, dates, etc. Have at least 2–3 teams report out #16 and encourage different ways of summarizing the main ideas.

When teams get to Model 3, remind the managers to keep track of time. Reserve some time at the end to report out #20 and #23. Discuss the similarities and differences of lists, tuples, and dictionaries. If time permits, have teams brainstorm everyday examples of each data type.


Copyright © 2019 T. Shepherd, C. Mayfield, and H. Hu. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

Model 1 Lists

A variable can hold multiple values in the form of a ***list***. The values are separated by commas and wrapped in square brackets. For example:

```
primes = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29]
```

Each ***element*** of the list can be referenced by an ***index***, which is the sequential position starting at 0. For example, `primes[4]` is 11.

index	0	1	2	3	4	5	6	7	8	9
value	2	3	5	7	11	13	17	19	23	29

Do not type anything yet! Read the questions first!

Python code	Shell output
`odd = [1, 3, 5, 7]`	
`odd`	[1, 3, 5, 7]
`odd[2]`	5
`odd[4]`	IndexError
`len(odd)`	4
`number = odd[1]`	
`number`	3
`odd[1] = 2`	
`odd`	[1, 2, 5, 7]
`number`	3

Questions (10 min) Start time: _____

1. What is the index of the second element of `primes`? What is the value at that index?

The index is 1. The value is 3.

2. How does the index number compare to the position of the element?

One less — index zero is the first element.

3. Type each line of code in a Python Shell and write the corresponding output in the space above. If an error occurs, write what type of error. Place an asterisk (*) next to any output for which you were surprised.

4. How did you reference the value of the 3rd element of `odd`?

odd[2]

5. What did the output of the len() function tell you about the list?

The length of the list

6. The output of Model 1 displayed an error. Explain the reason for the error.

It says "IndexError: list index out of range". The maximum index is 3 (i.e., length - 1), so 4 is too big.

7. Write a statement that assigns a list of three integers to the variable `run`.

run = [1, 2, 3]

8. Write a statement that assigns the value 100 to the last element of `run`.

run[2] = 100

9. Write a statement that assigns the first value of `run` to a variable named `first`.

first = run[0]

Model 2 Sequences

Lists and strings are examples of ***sequence*** types. Complete the table below to explore how sequences work.

Python code	Shell output
`seq1 = "one two"`	
`type(seq1)`	<class 'str'>
`len(seq1)`	7
`seq1[1]`	'n'
`seq1[1] = '1'`	TypeError: 'str' object does not support item assignment
`seq2 = "one", "two"`	
`type(seq2)`	<class 'tuple'>
`len(seq2)`	2
`seq2[1]`	'two'
`seq2[1] = '1'`	TypeError: 'tuple' object does not support item assignment
`seq3 = ["one", "two"]`	
`type(seq3)`	<class 'list'>
`seq3[1]`	'two'
`seq3[1] = 1`	
`seq4 = ("one", 1)`	
`type(seq4)`	<class 'tuple'>
`number = 12345`	
`number[3]`	TypeError: 'int' object is not subscriptable

Questions (15 min) Start time: _____

10. How does a sequence type differ from a number? (See the last row of the table.)

The value of each element of a sequence is accessible by an index

11. What are the names of the three sequence types introduced in Model 2?

list, string, and tuple

12. How does the syntax of creating a tuple differ from creating a list?

Tuples use parentheses, and lists use square brackets.

13. Is there more than one way (syntax) to create a tuple? Justify your answer.

Yes, the parentheses are optional as long as you use the comma operator. For example, the variable `seq2` is a tuple.

14. Which sequence types allow their elements to be changed? Which do not?

Only lists allow elements to be changed; strings and tuples do not support assignment.

15. Is it possible to store values of different types in a sequence? If yes, give an example from the table; if no, explain why not.

Yes; the `seq4` tuple ("one", 1) stores both a string and an integer.

16. Summarize the difference between lists and tuples. How do they look differently, and how do they work differently?

Tuples use parentheses, but lists use square brackets. Tuples cannot be modified, but lists can be modified.

Model 3 Dictionaries

In Python, a ***dictionary*** stores "`key: value`" pairs. The pairs are separated by commas and wrapped in curly braces. For example:

```
elements = {'C': 'carbon', 'H': 'hydrogen', 'O': 'oxygen', 'N': 'nitrogen'}
```

Key	Value
'C'	'carbon'
'H'	'hydrogen'
'O'	'oxygen'
'N'	'nitrogen'

In contrast to sequence types, a dictionary is a ***mapping*** type. Values are referenced by ***keys***, rather than by indexes.

Type the `elements` dictionary above into a Python Shell, and then complete the following table to explore how it works.

Python code	Shell output
`type(elements)`	<class 'dict'>
`elements.keys()`	dict_keys(['C', 'H', 'O', 'N'])
`elements.values()`	dict_values(['carbon', 'hydrogen', 'oxygen', 'nitrogen'])
`elements['C']`	'carbon'
`atom = 'N'`	
`elements[atom]`	'nitrogen'
`elements[N]`	NameError: name 'N' is not defined
`elements['nitrogen']`	KeyError: 'nitrogen'
`elements[1]`	KeyError: 1
`len(elements)`	4
`elements['B'] = 'Boron'`	
`elements.items()`	dict_items([('C', 'carbon'), ('H', 'hydrogen'), ...])

Questions (20 min) Start time: _____

17. List all the keys stored in the `elements` dictionary after completing the table.

The keys are `'C'`, `'H'`, `'N'`, `'B'`, and `'O'`.

18. What is the data type of the keys in the `elements` dictionary?

The keys are all strings. (Note: there is no "char" type in Python.)

19. Explain the reason for the error after entering each of the following lines:

a) `elements[N]` The letter N is treated as a variable name, and it's undefined.

b) `elements['nitrogen']` The string `"nitrogen"` is not one of the keys.

c) `elements[1]` The integer 1 is also not a key, and there are no indexes.

20. Ignoring the "dict_items()" part, describe the contents and type of data returned by the `items()` method.

It appears to return a list of tuples (of key-value pairs). In reality, the function returns a "view" of the dictionary. See https://docs.python.org/3/library/stdtypes.html#dict-views.

21. Write a Python expression that creates a dictionary for the seven days of the week, i.e., Sun=1, Mon=2, Tue=3, etc. Assign the dictionary to the variable `dow`.

```
dow = {'Sun': 1, 'Mon': 2, 'Tue': 3, 'Wed': 4, 'Thu': 5, 'Fri': 6, 'Sat': 7}
```

22. If you assign two different values to the same key (i.e., two assignment statements with one value each), which value is stored in the dictionary? Justify your answer with an example.

If you were to assign `dow['Sun'] = 8` followed by `dow['Sun'] = 0`, then 0 would replace the previous value.

23. Another way to store the data in Model 3 is to use two lists:

```
keys = ['C', 'H', 'O', 'N']
vals = ['carbon', 'hydrogen', 'oxygen', 'nitrogen']
```

What is a disadvantage of this approach? Explain your reasoning.

It's more difficult to insert new items: you have to write two assignment statements instead of one. It's even more difficult to update items: you have to determine the index of the key and replace the corresponding value in the other list.

Conditions and Logic

Computer programs make decisions based on logic: if some condition applies, do something, otherwise, do something else.

Content Learning Objectives

After completing this activity, students should be able to:

- Evaluate boolean expressions with comparison operators (<, >, <=, >=, ==, !=).
- Explain the syntax and meaning of `if`/`else` statements and indented blocks.
- Evaluate boolean expressions that involve comparisons with `and`, `or`, and `not`.

Process Skill Goals

During the activity, students should make progress toward:

- Evaluating complex logic expressions based on operator precedence. (Critical Thinking)

Facilitation Notes

Model 1 is quick and straightforward; you may not need to report out at all. Just keep an eye on each team's responses and give individual help as needed.

On Model 2, have teams share their answer for #9. Ask other teams if they tried additional experiments in the Python Shell to figure out the indenting rules of Python. You may want to demonstrate (on the projector) what happens if you forget the colon. It might also be helpful to show an if-statement with more than one line in the body.

During Model 3, explain that the variables p and q are often used to represent logic values in discrete math. Explain that "`not`" is a unary operator, and that "`and`" and "`or`" are binary operators. Double check each team's truth table before they complete the remaining questions.

Have each team write their answers to #24 and #25 on the board. Ideally you will have a variety of solutions, some of which are logically equivalent. Discuss the correct solutions to these questions, and reinforce intuition about logic.

As a wrap-up discussion, step through the last print statement of Model 3 using a debugger. Have the students check their work as you demonstrate the order of operations and the result of each expression.


Copyright © 2019 T. Shepherd, C. Mayfield, and H. Hu. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

Model 1 Comparison Operators

In Python, a comparison (e.g., `100 < 200`) will yield a ***Boolean*** value of either `True` or `False`. Most data types (including `int`, `float`, `str`, `list`, and `tuple`) can be compared using the following operators:

Operator	Meaning
<	less than
<=	less than or equal
>	greater than
>=	greater than or equal
==	equal
!=	not equal

Type the following code, one line at a time, into a Python Shell. Record the output for each line (if any) in the second column.

Python code	Shell output
`type(True)`	<class 'bool'>
`type(true)`	NameError
`type(3 < 4)`	<class 'bool'>
`print(3 < 4)`	True
`three = 3`	
`four = 4`	
`print(three == four)`	False
`check = three > four`	
`print(check)`	False
`type(check)`	<class 'bool'>
`print(three = four)`	TypeError
`three = four`	
`print(three == four)`	True

Questions (10 min) Start time: _____

1. What is the name of the data type for Boolean values? bool

2. Do the words `True` and `False` need to be capitalized? Explain how you know.

Yes, because `type(true)` resulted in NameError: name 'true' is not defined.

3. For each of the following terms, identify examples from the table in Model 1:

   a) Boolean variables: check

   b) Boolean operators: <, ==, >

   c) Boolean expressions: 3 < 4, three == four, three > four

4. Explain why the same expression `three == four` had two different results.

The two variables were initially different values, so the first comparison was `False`. But later on, the value of `four` was assigned to `three`, so the second comparison was `True`.

5. What is the difference between the = operator and the == operator?

The = operator assigns a value to a variable, and the == operator compares two values.

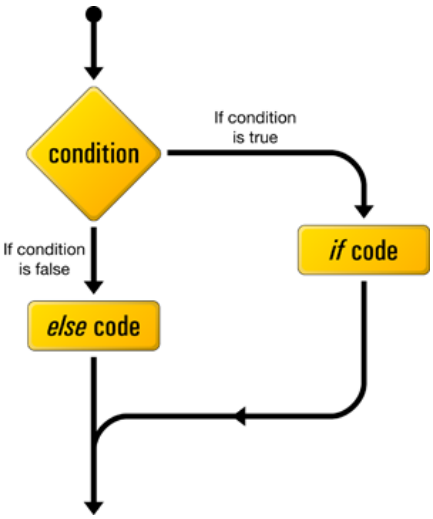
6. Write a Boolean expression that uses the `!=` operator and evaluates to `False`.

5 != 5

Model 2 `if`/`else` Statements

An `if` statement makes it possible to control what code will be executed in a program, based on a condition. For example:

```
number = int(input("Enter an integer: "))
if number < 0:
    print(number, "is negative")
else:
    print(number, "is a fine number")
print("Until next time...")
```



Python uses ***indentation*** to define the structure of programs. The line indented under the `if` statement is executed only when `number < 0` is `True`. Likewise, the line indented under the `else` statement is executed only when `number < 0` is `False`. The flowchart on the right illustrates this behavior.

Questions (15 min) Start time: _____

7. What is the Boolean expression in Model 2?

number < 0

8. Enter this short program into a Python Editor. What is the output when the user enters the number 5? What is the output when the user enters the number -5?

```
5 is a fine number
Until next time...
```

```
-5 is negative
Until next time...
```

9. After an if-condition, what syntax differentiates between (1) statements that are executed based on the condition and (2) statements that are always executed?

The indentation; statements that are indented under the if are based on the condition, and statements indented at the same level (later in the program) are always executed.

10. Enter the line ␣␣␣␣`print("Hello")` into a Python Editor (where ␣ is a space), save the file as `hello.py`, and run the program. What happens if you indent code inconsistently?

SyntaxError: unexpected indent

11. Based on the program in Model 2, what must each line preceding an indented block of code end with?

A colon.

12. Write an `if` statement that first determines whether `number` is even or odd, and then prints the message `"(number) is even"` or `"(number) is odd"`. (Hint: use the `%` operator.)

```
if number % 2 == 0:
    print(number, "is even")
else:
    print(number, "is odd")
```

13. Does an `if` statement always need to be followed by an `else` statement? Why or why not? Give an example.

No; you can have an if statement without an else. For example, you could determine that a number is even and print a message, without printing a different message if it's odd.

Model 3 Boolean Operations

Expressions may include Boolean operators to implement basic logic. If all three operators appear in the same expression, Python will evaluate `not` first, then `and`, and finally `or`. If there are multiple of the same operator, they are evaluated from left to right.

Do not type anything yet! Read the questions first!

Python code	Predicted output	Actual output
`print(a < b and b < c)`		True
`print(a < b or b < c)`		True
`print(a < b and b > c)`		False
`print(a < b or b > c)`		True
`print(not a < b)`		False
`print(a > b or not a > c and b > c)`		False

Questions (20 min) Start time: _____

14. What data type is the result of `a < b`? What data type is the result of `a < b and b < c`?

The type of each is `bool`; both are Boolean expressions.

15. Predict the output of each print statement, based on the variables `a = 3`, `b = 4`, and `c = 5`. Then execute each line in a Python Shell to check your work.

16. Based on the variables in #15, what is the value of a < b? What is the value of b < c?

They are both true.

17. If two `True` Boolean expressions are compared using the `and` operator, what is the resulting Boolean value?

True and True is True.

18. Using the variables defined in #15, write an expression that will compare two `False` Boolean expressions using the `or` operator. Check your work using a Python Shell.

a > b or a > c

19. Assuming `P` and `Q` each represent a Boolean expression that evaluates to the Boolean value indicated, complete the following table. Compare your team's answers with another team's, and resolve any inconsistencies.

P	Q	P and Q	P or Q
False	False	False	False
False	True	False	True
True	False	False	True
True	True	True	True

20. Assume that two Boolean expressions are compared using the `and` operator. If the value of the first expression is `False`, is it necessary to determine the value of the second expression? Explain why or why not.

It is unnecessary, because "false and anything" is false.

21. Assume that two Boolean expressions are compared using the `or` operator. If the value of the first expression is `True`, is it necessary to determine the value of the second expression? Explain why or why not.

It is unnecessary, because "true or anything" is true.

22. Examine the last row of the table in #15. Evaluate the Boolean expression following the order of precedence rules explained in Model 3. Show your work by rewriting the line at each step and replacing portions with either `True` or `False`.

```
a > b or not a > c and b > c
False or not a > c and b > c
False or not False and b > c
   False or True and b > c
   False or True and False
       False or False
           False
```

23. Suppose you wanted to execute the statement `sum = x + y` only when both `x` and `y` are positive. Determine the appropriate operators, and write a single Boolean expression for the if-condition.

x > 0 and y > 0

24. Rewrite the expression from #23 using the `not` operator. Your answer should yield the same result as in #23, not the opposite. Describe in words what the new expression means.

not (x <= 0 or y <= 0)
In other words, "both x and y are positive" is equivalent to "neither x nor y is negative/zero".

25. Suppose that your team needs to execute the statement `sum = x + y` except when both `x` and `y` are positive. Write a Boolean expression for this condition. How is it different from the previous question?

not (x > 0 and y > 0)
To represent "except when" logic, we simply negate the original condition. The previous question negated each of the operators as well, which is known as De Morgan's law.

Loops and Iteration

A loop allows you to execute the same statements multiple times. Python has two kinds of loop structures: `for` loops, which iterate over the items of a sequence, and `while` loops, which continue to execute as long as a condition is true.

Content Learning Objectives

After completing this activity, students should be able to:

- Explain the syntax and the purpose of a `for` statement.
- Predict how `range()` works given 1, 2, or 3 arguments.
- Identify the three main components of a `while` loop.

Process Skill Goals

During the activity, students should make progress toward:

- Tracing the execution of while/for loops and predict their final output. (Critical Thinking)

Facilitation Notes

Encourage teams to complete Model 1 quickly. Monitor the first page to ensure that each team answers the questions correctly. Report out by having teams swap presenters and compare answers on the second page.

On Model 2, there might be some confusion about the use of the `list()` function. In order to view the actual numbers in the range, we need to convert it to a list. Technically speaking, `range()` is not a function in Python 3, but an immutable sequence type. Be prepared to explain this detail to students if they ask. Make sure teams don't use `list()` in their answers for #14.

If you're running short on time, skip Question #16 or assign as homework. It assumes students have already learned how to convert integers to characters.

For question #25 on Model 3, it may be easier for students to write the loop itself than to explain its components. If they get stuck, have them write the code on a separate sheet of paper and then use the questions to analyze their solution.

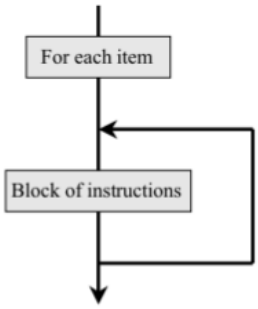
As a wrap-up question, ask the class whether it's possible for a `for` loop to run forever. Discuss the difference between repeating a definite number of times (`for`) vs indefinitely (`while`).


Copyright © 2019 T. Shepherd, C. Mayfield, and H. Hu. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

Model 1 `for` Statements

A `for` loop executes the same block of code "for each item in a sequence". Create a new file named `loops.py`, and enter the following code:

```
print("hello")
for x in [2, 7, 1]:
    print("the number is", x)
print("goodbye")
```



Questions (15 min) Start time: _____

1. Run the `loops.py` program. How many times does the indented line of code execute under the `for` loop?

3 times

2. How many times does the line of code NOT indented execute after the `for` loop?

1 time

3. Identify the value of `x` each time the indented line of code is executed.

   a) 1st time: x = 2

   b) 2nd time: x = 7

   c) 3rd time: x = 1

4. Modify the list `[2, 7, 1]` in the following ways, and rerun the program each time. Indicate how many times the `for` loop executes.

   a) non-consecutive numbers: `[5, -7, 0]` 3 times

   b) numbers decreasing in value: `[3, 2, 1, 0]` 4 times

   c) all have the same value: `[4, 4]` 2 times

   d) single value in a list: `[8]` 1 time

5. In general, what determines the number of times that the loop repeats?

The length of the list.

6. What determines the value of the variable `x`? Explain your answer in terms of what is assigned (`x = ...`) each time the loop runs.

The value `x` is selected from the list. Each time the loop runs, the next value from the list is assigned to `x`.

7. Modify the program as follows:

   a) Write a statement that assigns `[0, 1, 2, 3, 4]` to the variable `numbers`.

   ```
   numbers = [0, 1, 2, 3, 4]
   ```

   b) Rewrite the `for x ...` statement to use the variable `numbers` instead.

   ```
   for x in numbers:
   ```

   c) Does the assignment need to come before or after the `for` statement?

   Before

8. Add the following code at the end of your program:

```
for c in "Hi!":
    print(c)
```

   a) What is the output of this `for` statement?

   ```
   H
   i
   !
   ```

   b) What determined how many times `print(c)` was called?

   The length of the string

   c) Explain what a `for` statement does with strings.

   It iterates over each character

9. What other data types (besides lists and strings) can a `for` loop handle? Experiment by adding examples to your `loops.py` program. Summarize here what works and what doesn't.

Tuples work similar to lists. Dictionaries give you the keys.
Numbers don't work; you can loop over integers and floats.

Model 2 The `range` Function

The Python `range` function will generate a list of numbers. The `range` function can take up to three numbers as arguments. Fill in the table below by typing the code into a Python Shell:

Python code	Shell output
`range(5)`	range(0, 5)
`list(range(5))`	[0, 1, 2, 3, 4]
`x = range(3)`	
`print(x)`	range(0, 3)
`print(list(x))`	[0, 1, 2]
`list(range(5, 10))`	[5, 6, 7, 8, 9]
`list(range(-3, 4))`	[-3, -2, -1, 0, 1, 2, 3]
`list(range(4, 10, 2))`	[4, 6, 8]
`for i in range(5):` `  print(i)`	prints 0, 1, 2, 3, 4 (on separate lines)

Questions (15 min) Start time: _____

10. Explain the difference in output between the first two lines of code (with and without the `list` function).

The first line of output describes the range as a function. The second line shows the actual range of values as a list.

11. If the argument of the `range` function specifies a single number (x):

a) What will be the first number listed? 0
b) What will be the last number listed? x - 1
c) How many numbers will be in the list? x
d) Use the range function to generate the sequence 0, 1, 2, 3. range(4)

12. If the argument of the `range` function specifies two numbers (x, y):

a) What will be the first number listed? x
b) What will be the last number listed? y - 1
c) How many numbers will be in the list? y - x
d) Use the range function to generate the sequence 1, 2, 3, 4. range(1, 5)

13. If the argument of the `range` function specifies three numbers (x, y, z):

a) What will be the first number listed? x

b) What does the third argument represent? how much to add each time

c) How many numbers will be in the list? $\lceil (y - x)/z \rceil$

d) Use the range function to generate the sequence 1, 3, 5, 7. range(1, 8, 2) or range(1, 9, 2)

14. In your Editor, make a copy of the Model 1 code. Then modify the `for` statement so that the number of times the loop executes is determined by a variable named `times`.

a) How did you change the `for` statement?

```
for i in range(times):  # no need for list() conversion
```

b) How would you cause the loop to print the values 0 to 5?

Add this line before the loop: `times = 6`

15. Consider the two different types of `for` statements used in Model 1 and Model 2.

a) If you wanted to execute a loop 100 times, which type of `for` statement would you choose and why?

for i in range(number), so that you don't have to specify the list.

b) If you wanted to use each item of an existing list inside the loop, which type of `for` statement would you choose and why?

for i in list, since the list exists already and might not be a range.

16. Does the `range` function work with strings? If so, show an example. If not, show how to print the letters A to Z in a loop.

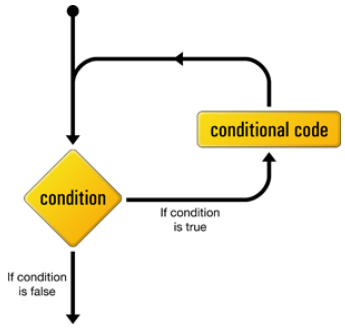
The arguments to range must be integers. You can use the built-in function `chr` to convert integers to their corresponding Unicode characters:

```
for i in range(65, 91):
    print(chr(i))
```

Model 3 `while` Statements

A more general looping structure is the `while` statement. Add the code below to your current `loops.py` program:

```
i = 0
while i < 3:
    print("the number is", i)
    i = i + 1
print("goodbye")
```



Questions (15 min) Start time: _____

17. What must the value of the Boolean expression (after the `while`) be in order for the first print statement to execute? True

18. Circle the statement that changes the variable `i` in the above code. i = i + 1

19. What happens to the value of the variable `i` during the execution of the loop?

It increments by one each time the loop body is executed.

20. Explain why the loop body does not execute again after it outputs "the number is 2".

The variable `i` then becomes 3, which causes the condition `i < 3` to be false.

21. Reverse the order of the statements in the loop body:

```
while i < 3:
    i = i + 1
    print("the number is", i)
```

a) How does the order impact the output displayed by the `print` function?

It prints the numbers 1,2,3 instead of 0,1,2.

b) Does the order impact the total number of lines that are output?

No it does not—either way, there are 3 lines.

22. Identify three different ways to modify the code so that the loop only executes twice.

You can change the first line to `i = 1`. Or you can change the condition to `i < 2`. Or you can change the last line to `i = i + 2`.

23. Describe the three parts of a `while` loop that control the number of times the loop executes.

The first part initializes the variable or condition. The second part tests whether the end has been reached. The third part updates the variable or condition.

24. Comment out the statement `i = i + 1`, and run the module. Then press Ctrl-C (hold down the Ctrl key and press C). Describe the behavior you see, and explain why it happened.

It prints "the number is 0" forever, until you press Ctrl-C. Then it displays "KeyboardInterrupt" as an error message. This all happened because the value of `i` never changed.

When writing a `while` loop, it's helpful to answer a few questions before you start:

- *What needs to be initialized before the loop?*
- *What condition must be true for the loop to repeat?*
- *What will change so that the loop eventually ends?*

25. Consider the function `add(n)` that prompts the user for n numbers and returns the sum of these values. For example, when `add(5)` is called, the user is asked to input five numbers. If the user inputs 3, 1, 5, 2, and 4, the function would return the value 15.

a) Describe the variable that needs to be initialized before the loop begins. i = 0

b) Describe the Boolean expression that must be true for the loop to continue. `i < n`

c) Describe what will need to change so that the loop will eventually end. i = i + 1

d) Now list what else needs to happen inside the body of the loop for you to calculate the sum of the user input.

1) Prompt the user to input a number, and 2) add that number of a running total.

e) Given your previous answer, are there any other values that need to be initialized before the start of the loop?

Yes, the running total should be initialized to zero.

Defining Functions

Python programs typically have one or more functions, each of which has one or more statements. Defining new functions allows you to break down a complex program into smaller blocks of reusable code.

Content Learning Objectives

After completing this activity, students should be able to:

- Explain the flow of execution from one function to another.
- Describe the syntax of function definitions and function calls.
- Write short functions that have parameters and return results.

Process Skill Goals

During the activity, students should make progress toward:

- Tracing the execution of functions with Python Tutor. (Information Processing)

Facilitation Notes

This activity uses Python Tutor (by Philip Guo) to visualize the flow of execution. Each model is interactive—to encourage discussion, have only one student per team run the visualization. It's particularly important in this activity that managers keep track of time. You can save time by providing students with the source code in advance.

Check in with each team during Model 1, and make sure they can run the visualization. Double check their answers for the first several questions. Have the first team that finishes draw their answer to #12 on the board. Invite another team to draw their answer to #8 on the board.

At the end of Model 2, invite teams to explain the difference between parameters and arguments. Discuss how arguments are passed to their corresponding parameters by assignment. If time permits, demonstrate the solution for the last two questions in Python Tutor.

On Model 3, remind students to work quickly. During report out, invite teams to explain what happened in #25. Emphasize that a value is always returned when a function call completes. Explain that `None` is a built-in constant like `False` and `True`.


Copyright © 2019 T. Wilson, B. Wahl, and C. Mayfield. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

Model 1 Flow of Execution

In addition to using Python's built-in functions (e.g., `print`, `abs`) and functions defined in other modules (e.g., `math.sqrt`), you can write your own functions.

```
1  def model_one():
2      word = input("Enter a word: ")
3      L = len(word)
4      ans = word * L
5      print(ans)
6
7  def main():
8      print("Starting main...")
9      model_one()
10     print("All done!")
11
12 main()
```

Questions (20 min) Start time: _____

1. Based on the program in Model 1:

   a) What is the Python keyword for defining a function? def

   b) On what line is the `model_one` function defined? _____ 1 called? _____ 9

   c) On what line is the `main` function defined? _____ 7 called? _____ 12

2. Open a web browser and go to PythonTutor.com. Click on "Visualize your code", and type (or paste) the program above. Make sure the line numbers match.

3. Click the "Visualize Execution" button. As you step through the program, pay attention to what is happening on the **left side** of the visualization.

   a) What does the **red** arrow indicate? The next line to execute.

   b) What does the **green** arrow indicate? The line that just executed.

4. Notice the order in which the program runs:

   a) After line 12 of the program executes (Step 3), what is the next line that executes? Line 7

   b) After line 9 of the program executes (Step 6), what is the next line that executes? Line 1

5. Go back to the beginning of the program execution. This time as you step through the program, pay attention to what changes on the **right side** of the visualization.

a) Describe what changes in the visualization after Step 1.

Under Frames, a box labeled "Global Frame" appears. Inside the box, it lists `model_one` with an arrow pointing to `function model_one()` under Objects.

b) Describe what changes in the visualization after Step 2.

The name `main` is added to the "Global Frame" box, with an arrow pointing to `function main()` under Objects.

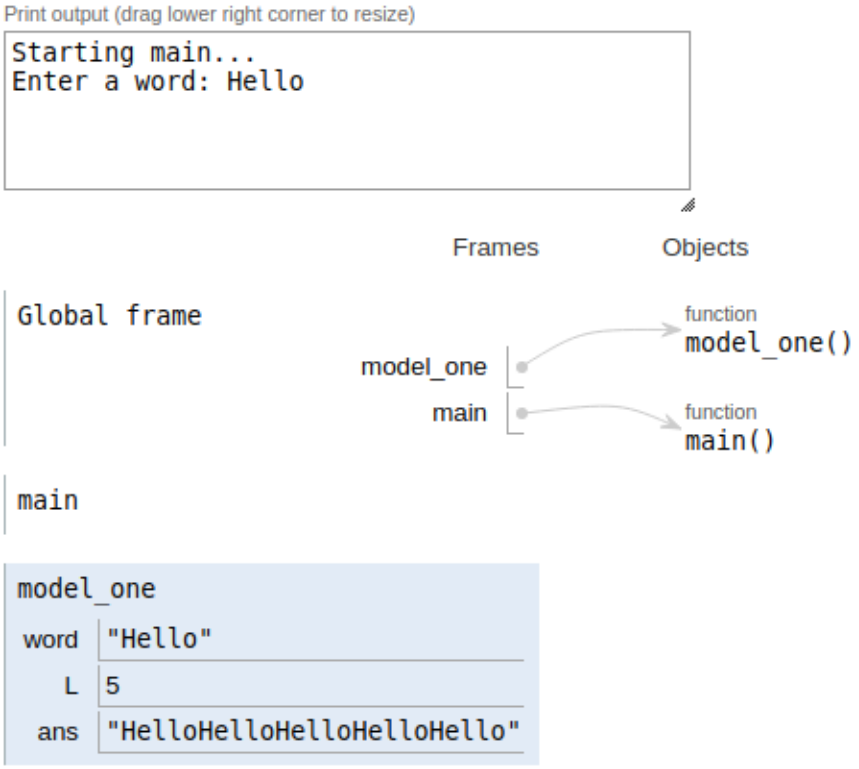
6. In general, what happens on the right side of the visualization when a function is called?

A box with the function name appears under Frames.

7. In terms of execution order, what is the effect of calling a function?

The code for the function will begin executing.

8. Draw the right side of the visualization for Step 11 in the space below.



9. Notice that the variable `ans` is printed from within the `model_one` function. What happens if you try to `print(ans)` inside the main function?

NameError: name 'ans' is not defined

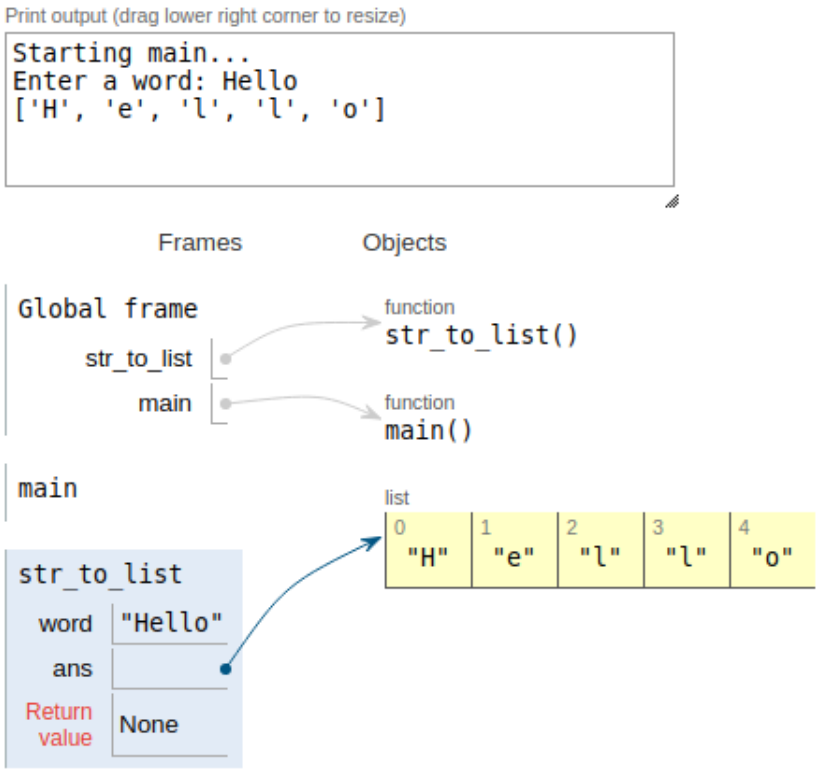
10. Explain what happened in the previous question in terms of frames in the visualization.

The variable `ans` never shows up in the frame for `main`, because it is local to `model_one`. When `model_one` finishes executing, everything associated with its frame disappears.

11. In the space below, write a definition for a function called `str_to_list` that prompts the user to enter a word. The function should convert the string to a list and print the list.

```
def str_to_list():
    word = input("Enter a word: ")
    ans = list(word)
    print(ans)
```

12. Edit the program in Python Tutor so that, instead of defining and calling the function `model_one`, it defines and calls the function `str_to_list`. Verify your changes by visualizing the execution, and draw a picture of the right side immediately after the list is printed.



Model 2 Passing Arguments

Instead of using `input` inside a function to get data, we can define a function to take a ***parameter*** (variable). When we call the function, we need to provide an ***argument*** (value). Change the program in Python Tutor as follows:

```
1  def model_two(word):
2      ans = word * len(word)
3      print(ans)
4
5  def main():
6      print("Starting main...")
7      w = input("Enter a word: ")
8      model_two(w)
9      print("All done!")
10
11 main()
```

Questions (15 min) Start time: _____

13. Underline the parameter in the `model_two` function definition, then circle each use of the parameter inside the function. two uses of the parameter `word` should be circled in line 2

14. Find the `model_two` function call in `main`, and underline the argument being passed by the function call. word should be underlined on line 1, and w should be underlined on line 8

15. Visualize the execution of `model_two` until Step 8.

    a) How does the frame for `model_two` at this point in the execution differ from the frame for `model_one` previously?

    The parameter `word` is already showing with a value in the frame for `model_two`.

    b) Write the implied assignment statement to show how the parameter `word` gets its value.

    `word = w`

    c) When a variable is used as an argument, does the name of the variable need to be the same as the parameter variable name?

    No

16. Assume that `s1 = "Hi"` and `s2 = "ya"`. In the function call `model_two(s1 + s2)`:

a) What is the argument for the function call? s1 + s2 (the expression)

b) Write the implied assignment statement that happens during the call. word = s1 + s2

c) What will be the value of parameter `word` when `model_two` begins executing? "Hiya"

d) Predict the output that will be produced by the function call. HiyaHiyaHiyaHiya

17. Review the two implied assignment statements that you have written. What exactly gets "passed" when you call a function?

The value of the argument is passed; it gets assigned to the parameter. Note that the argument itself is not passed.

18. Change `model_two` so that, instead of multiplying `word` by the length of `word`, it will multiply by an integer passed as the second argument to the function. Write the new version of `model_two` in the space below. Use `times` for the name of the new integer parameter.

```
def model_two(word, times):
    ans = word * times
    print(ans)
```

19. How does the call to `model_two` in `main` need to change so that it matches the new function definition? Give an example.

A second argument (of type `int`) must be added to the function call: `model_two("Hi", 2)`

Model 3 Returning Values

Functions may optionally send a value back to the calling function using a `return` statement. Change the program in Python Tutor as follows:

```
1  def model_three(word):
2      ans = word * len(word)
3      return ans
4
5  def main():
6      print("Starting main...")
7      w = input("Enter a word: ")
8      result = model_three(w)
9      print(result)
10     print("All done!")
11
12 main()
```

Questions (10 min) Start time: _____

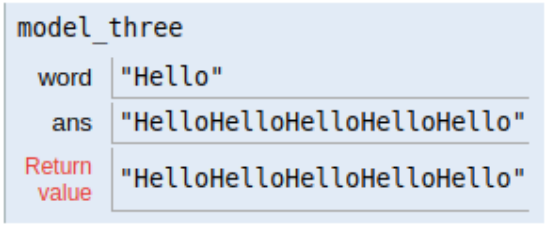
20. Aside from the function name, how does line 8 in Model 3 differ from line 8 in Model 2?

It is now an assignment statement.

21. At what step number (in the simulation) has `model_three` completed its execution, but control has not yet returned to the `main` function?

Step 11.

In the space below, draw the frame for `model_three` after this step.



22. In general, what value will be returned by `model_three`?

Whatever is the value of the variable `ans`.

23. What changes in the frame for `main` at Step 12 of the execution?

The variable `result` has been added to the frame. It has the value returned by `model_three`.

24. Edit `model_three` and delete the return statement at the end of the function. Visualize the execution. What value is returned by a function when there is no return statement?

The value `None`.

25. Edit `model_three` again, and add the return statement back to the end of the function. Then change line 8 so that `model_three` is still called but there is no assignment to `result`. What do you predict will happen in `main` after the `model_three` function call completes?

Because `result` is not being assigned the value returned by the `model_three` function call, trying to print the value of `result` will cause a `NameError`.

26. Why is a function that returns the value of a variable more useful than a function that simply prints the value of that variable?

Printing a value to the screen only benefits the program in that moment. If you instead return the value and assign it to a variable, you can continue using it in the calling function.

Lists and Strings

Many interesting problems involve manipulating sequences of data. You've learned about lists and strings before, but this activity provides a more in-depth look at what they can do.

Content Learning Objectives

After completing this activity, students should be able to:

- Name four methods that lists provide, and describe what each method does.
- Explain the syntax and meaning of slice operations, with and without indexes.
- Name four methods that strings provide, and describe what each method does.

Process Skill Goals

During the activity, students should make progress toward:

- Gaining insight about data structures from many examples. (Information Processing)

Facilitation Notes

This activity is slightly longer than usual, so it's important that the managers keep track of time and make sure teams don't fall behind. Remind students not to give too much thought to the shell output when completing the tables in each model.

Questions #5 and #6 in Model 1 are good for reporting out. Call on at least 2–3 teams to share different perspectives on these answers. Ask teams to explain what else they learned about these functions, if time permits.

When reporting out Model 2, make a connection between the slice `[m:n]` and the `range(m, n)`. In both cases, the range includes `m` but not `n`. Briefly explain that slices may have a third argument, just like the `range` function. Just for fun, show an example of reversing a list or string using the slice `[::-1]`.

On Model 3, make sure students notice the type of `dna` vs the type of `dna[0]`. When discussing question #20, challenge students to describe (hypothetically) what a `replace` method might look like for lists, and how it would be different than `replace` for strings. Report out for other questions as time permits.


Copyright © 2019 T. Shepherd, C. Mayfield, and H. Hu. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

Model 1 Working with Lists

Recall that a variable can hold multiple values in the form of a list. The values are separated by commas and wrapped in square brackets.

Lists have ***methods*** (built-in functions) that can be called using dot notation. For example, to add a new element to the end of a list, we can use the `append` method.

Python code	Shell output
`rolls = [4, 6, 6, 2, 6]`	
`len(rolls)`	5
`print(rolls[5])`	IndexError: list index out of range
`rolls.append(1)`	
`print(rolls)`	[4, 6, 6, 2, 6, 1]
`print(rolls[5])`	1
`lucky.append(1)`	NameError: name 'lucky' is not defined
`lucky = []`	
`print(lucky[0])`	IndexError: list index out of range
`lucky.append(5)`	
`print(lucky)`	[5]
`print(lucky[0])`	5
`rolls.count(6)`	3
`rolls.remove(6)`	
`print(rolls)`	[4, 6, 2, 6, 1]
`help(rolls.remove)`	remove first occurrence of value
`help(rolls)`	Help on list object (multiple pages)

Questions (15 min) Start time: _____

1. What is the result of calling the `append` method on a list?

The value gets added to the end of the list. Nothing is returned.

2. What must be defined prior to using a method like `append`?

The list itself; `lucky.append(5)` is an error if `lucky` is not defined.

3. Explain why two lines in Model 1 caused an `IndexError`.

In both cases, we asked for an index that was out of range. If the length of an index is n, the highest index is $n - 1$.

4. What is the result of calling the `remove` method on a list?

It removes the first occurrence of a value. The list changes as a result of this method.

5. Based on the `help` output, name several list methods not shown in Model 1. Do not include methods that begin and end with two underscores (e.g., __add__).

Answers may include: clear, copy, extend, index, insert, pop, reverse, sort.

6. Give one example of a list method that requires an argument and one that does not.

Methods that require arguments: append, count, extend, index, insert, remove. Methods that do not: clear, copy, pop, reverse, sort.

7. Describe the similarities and differences between using a list method like `append` and Python built-in functions like `print`.

Both use parentheses and take arguments. The list methods come after the dot operator, and the built-it functions surround the list itself.

8. Complete the function below (two lines are missing). It should prompt the user for numbers and build a list by adding one number at a time to the end of the list. The loop terminates when the user inputs the number 0.

```
def input_numbers():
    x = 1
    numbers = []
    _____________________
    while x != 0:
        x = int(input("Enter the next number: "))
        numbers.append(x)
        ___________________________________
    return numbers
```

Model 2 Indexing and Slicing

A string is a sequence of characters in single quotes (') or double quotes ("). Depending on the application, we can treat a string as a single value (e.g., `dna`), or we can access individual characters using square brackets (e.g., `dna[0]`). We can also use ***slice notation*** (e.g., `dna[4:8]`) to refer to a range of characters. In fact, all types of sequences (including `list` and `tuple`) support indexing and slicing.

Python code	Shell output
`dna = 'CTGACGACTT'`	
`dna[5]`	'G'
`dna[10]`	IndexError: string index out of range
`len(dna)`	10
`dna[:5]`	'CTGAC'
`dna[5:]`	'GACTT'
`dna[5:10]`	'GACTT'
`triplet = dna[2:5]`	
`print(triplet)`	GAC
`dna[-5]`	'G'
`dna[-10]`	'C'
`dna[:-5]`	'CTGAC'
`dna[-5:]`	'GACTT'
`triplet = dna[-4:-1]`	
`print(triplet)`	'ACT'

Questions (15 min) Start time: _____

9. What is the *positive* index of each character in the `dna` string? Check your answers above.

Character:	C	T	G	A	C	G	A	C	T	T
Index:	0	1	2	3	4	5	6	7	8	9

10. What is the *negative* index of each character in the `dna` string? Check your answers above.

Character:	C	T	G	A	C	G	A	C	T	T
Index:	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

11. Based on the previous questions, what are `dna[2]` and `dna[-2]`? Explain your answers.

They are G and T, respectively. Index 2 means to the third from the left, and index -2 means the second from the right.

12. Explain the IndexError you observed. What is the range of indexes for the `dna` string?

Because the length of the string is 10, the indexes range from 0 to 9. Therefore, `dna[10]` is out of range.

13. Consider the notation of the operator `[m:n]` for slicing the string.

a) Is the value at `m` the same as the corresponding index value (i.e., `dna[m]`)? If not, describe what it means. Yes; `m` is the first character in the slice.

b) Is the value at `n` the same as the corresponding index value (i.e., `dna[n]`)? If not, describe what it means. No; `n` is the index after the last character.

c) Explain what it means when only a single number is referenced when creating a slice, such as `[m:]` or `[:n]`. The slice `[m:]` means "from the index `m` to the end". The slice `[:n]` means "from the beginning to the index just before `n`" (i.e., the first `n` characters).

14. What is the simplest way to get the first three characters of `dna`? What is the simplest way to get the last three characters?

Based on the previous question, we know that `dna[:3]` gets the first three characters. To get the last three, we use `dna[-3:]`.

15. Write a Python expression that slices `'GACT'` from `dna` using positive indexes. Then write another expression that slices the same string using negative indexes.

```
dna[5:9]      dna[-5:-1]
```

16. Write a Python assignment statement that uses the `len` function to assign the last letter of `dna` to the variable `last`.

```
last = dna[len(dna) - 1]
```

17. Write a Python assignment statement that uses a negative index to assign the last letter of `dna` to the variable `last`.

```
last = dna[-1]
```

Model 3 Common String Methods

Like lists, strings have ***methods*** (built-in functions) that can be called using dot notation. See https://docs.python.org/3/library/stdtypes.html#string-methods for more details.

Python code	Shell output
`dna = 'CTGACGACTT'`	
`dna.lower()`	'ctgacgactt'
`print(dna)`	CTGACGACTT
`lowercase = dna.lower()`	
`print(lowercase)`	ctgacgactt
`dnalist = list(dna)`	
`print(dnalist)`	['C', 'T', 'G', 'A', 'C', 'G', 'A', 'C', 'T', 'T']
`dnalist.reverse()`	
`print(dnalist)`	['T', 'T', 'C', 'A', 'G', 'C', 'A', 'G', 'T', 'C']
`type(dna)`	<class 'str'>
`dna = dna.split('A')`	
`print(dna)`	['CTG', 'CG', 'CTT']
`type(dna)`	<class 'list'>
`dna.replace('C', 'g')`	AttributeError: 'list' object has no attribute 'replace'
`print(dna[0])`	CTG
`type(dna[0])`	<class 'str'>
`dna[0].replace('C', 'g')`	'gTG'
`print(dna)`	['CTG', 'CG', 'CTT']

Questions (15 min) Start time: _____

18. Does the `lower` method change the contents of the `dna` string? Justify your answer.

No, it does not. The next line of code prints `dna`, which is unchanged.

19. Describe the `list` function—what does `list(dna)` return in Model 3?

It returns a list of the individual characters. Each element of the list is a string of length 1. (Note that Python does not have a character data type.)

20. Why is it possible to call the `replace` method on `dna[0]` but not `dna`?

The `list` data type does not include a replace method. However, strings allow you to "find and replace" any text.

21. Name several other string methods not shown in Model 3. (Read the documentation.)

There are dozens of string methods; the model only uses lower, split, and replace.

22. Consider the application of a method on a variable:

a) Does a string variable change after applying a method? Provide justification.

No it doesn't; neither `lower` nor `replace` modify the string.

b) Does a list variable change after applying a method? Provide justification.

It might; for example, the `reverse` method changes the list.

c) Identify the data type that is ***immutable*** (i.e., the value never changes).

String

23. Write a single statement to change the final contents of `dna` to `['CTG', 'cc', 'CTT']`. Confirm that your code works in a Python Shell.

```
dna[1] = 'cc'
```

24. Why do you think Python has a `replace` method for strings but not for lists?

Answers may vary. One reason might be that lists are more complex than strings: they can store any type of data, not just characters. Another reason might be that there are fewer applications of replacing data in lists than patterns in text.

Importing Modules

Python comes with an extensive library of built-in modules that make it easy to accomplish everyday tasks. With just a few lines of code, you can do anything from generating random numbers and drawing graphics to sending emails and accessing websites.

Content Learning Objectives

After completing this activity, students should be able to:

- Use the `random` module to generate random float and integer sequences.
- Explain the purpose of the common line `if __name__ == "__main__"`.
- Summarize several built-in modules, including `random` and `turtle`.

Process Skill Goals

During the activity, students should make progress toward:

- Navigating the Python standard library documentation. (Information Processing)

Facilitation Notes

This activity addresses misconceptions about the `import` statement and shows how to write "longer" programs using multiple source files. Model 1 uses the `random` module as an example: someone else has already written code to generate random numbers, and you can import this code into your own program.

During Model 2, students learn how to write their own modules that can be imported. To save time, be sure to provide students with the source files at the beginning of the activity. When reporting out, spend time discussing the Python idiom `if __name__ == "__main__"`.

Model 3 introduces the `turtle` module, but it's more about navigating the Python library documentation. Before students answer the questions, you might show them how to bring up the documentation page (https://docs.python.org/3/library/turtle.html). Have each team report out on the last question and share with the class what they found in the standard library.


Copyright © 2019 T. Shepherd, C. Mayfield, and H. Hu. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

Model 1 Random Numbers

You can generate a sequence of numbers using the Python `random` module. A mathematical function is used to produce the sequence based on a ***seed*** value. (If no seed is given, the current system time is used.) The sequence is more accurately described as ***pseudorandom***, since its output is inherently predictable.

Python code	Shell output
`import randint`	ImportError
`import random`	
`randint(1, 10)`	NameError
`random.randint(1, 10)`	integer in range [1..10]
`from random import randint`	
`randint(1, 10)`	integer in range [1..10]
`seed(100)`	NameError
`random.seed(100)`	
`random.random()`	0.1456692551041303 – decimal in range [0..1)
`random.random()`	0.45492700451402135 – decimal in range [0..1)
`random.seed(100)`	
`random.random()`	0.1456692551041303 – decimal in range [0..1)
`random.random()`	0.45492700451402135 decimal in range [0..1)

Questions (20 min) Start time: _____

1. What is the name of the module that must be imported before generating a random number?

random

2. Based on Model 1, what are the names of three functions defined in the `random` module?

randint(), random(), seed()

3. Identify the syntax of the statement to import:

   a) a module import module

   b) a function from module import function

4. Identify the syntax of a function call assuming:

   a) the module was imported module name "." function name

   b) the function was imported function name

5. How could you eliminate the need for typing the word "random" twice (in a function call) to generate a random number?

from random import random

6. Compare the shell output of your team with at least one other team. Describe the similarities and differences observed.

Different random numbers initially and then same sequence of numbers after a seed is set – but unique for each different seed argument.

7. What is the effect on the random numbers generated after calling the `seed` method?

same sequence of random numbers generated

8. Describe one reason to set the same seed each time a program is run, and one reason to not use the `seed` method.

Use: debugging and testing. Don't use: when it's supposed to be random.

9. Run `random.random()` multiple times. Based on the results, describe:

   a) the range of numbers returned by the `random` function between [0..1) exclusive

   b) the nature of the distribution of numbers generated. (Do they appear clustered around a particular value, or are they spread out uniformly over the range?) uniform

10. Run `random.randint(1, 10)` multiple times. Based on the results, describe:

   a) the range of numbers returned by the `randint` function between [0..10] inclusive

   b) the nature of the distribution of numbers generated. (Do they appear clustered around a particular value, or are they spread out uniformly over the range?) uniform

Model 2 Multiple Modules

Create a new file `move.py`, and enter the code:

```
1  import random
2
3  def angle():
4      number = random.randint(-90, 90)
5      return number
6
7  print("in move: __name__ ==", __name__)
8  print("will always execute: angle ==", angle())
9
10 if __name__ == "__main__":
11     print("only if True: angle ==", angle())
```

Run `move.py`, and record the output below. (numbers will vary)

Output Line 1	in move: __name__ == __main__
Output Line 2	will always execute: angle == 68
Output Line 3	only if True: angle == -39

Create a new file `stop.py` (in the same directory), and enter the code:

```
1  import move
2
3  print("in stop: __name__ ==", __name__)
4  print("from module: angle ==", move.angle())
```

Run `stop.py`, and record the output below. Draw an arrow from each line of output to its corresponding print statement in the code.

Output Line 1	in move: __name__ == move	arrow to Line 7 of `move.py`
Output Line 2	will always execute: angle == 74	arrow to Line 8 of `move.py`
Output Line 3	in stop: __name__ == __main__	arrow to Line 3 of `stop.py`
Output Line 4	from module: angle == -11	arrow to Line 4 of `stop.py`

Questions (15 min) Start time: _____

11. Upon execution of `move.py`:
    a) what is the value of the variable `__name__`? `__main__`

    b) does the output correspond solely to the print statements contained in this file? yes

12. Upon execution of `stop.py`:
    a) what is the value of the variable `__name__` from the print statement in `move` `move`

    b) what is the value of the variable `__name__` from the print statement in `stop` `__main__`

    c) does the output correspond solely to the print statements contained in this file? no

13. What was the reason to include the `import move` statement in `stop.py`?

To use the angle function defined in `move.py`.

14. Based on the output of `stop.py`, describe what happens (as a side effect) when another module is imported.

All the code in the imported file, including top-level print statements, is executed.

15. What line in `move.py` did not print when `stop.py` was executed? Why?

The print "only if True" statement inside `if __name__ == "__main__"` was False because `__name__ == "move"` that time.

16. In order for the output of `stop.py` to correspond solely to the print statements contained in `stop.py`, what modifications need to be made to `move.py`?

Move all the print statements inside `if __name__ == "__main__"`.

17. Describe what code in general to include inside `if __name__ == "__main__"`, and why.

Code that you don't want to be executed when the module is imported.

Model 3 Turtle Graphics

The `turtle` module can be used to create graphics. Create a new file `draw.py` (in the same directory), and enter the following code. Run the program and see what happens.

```
1  import move
2  import turtle
3
4  def randomwalk(steps):
5      turtle.shape("turtle")
6      turtle.color("green")
7      for i in range(steps):
8          turtle.left(move.angle())
9          turtle.forward(10)
10     turtle.bye()
11
12 if __name__ == "__main__":
13     randomwalk(100)
```

Questions (10 min) Start time: _____

18. For each outcome, describe the type of edit necessary to `draw.py` and `move.py`:

a) a blue turtle change the argument of `turtle.color()` to "blue"

b) a longer simulation change the argument of `randomwalk()` to a number greater than 100

c) a smaller range of angles (e.g., -45 to 45) that define the direction of the turtle
change the argument(s) of `random.ranint()` in the `angle` function defined in `move.py`

d) a random range of integers (e.g., 10 to 20) that define the length of a turtle move
import random and change argument of `turtle.forward()` to `random.ranint(10,20)`

19. Describe the type of edit necessary to produce the same outcome in Question #18d if the argument of `forward` is `move.length()` instead of 10:

Add a function named `length` in the file `move.py`.

20. Go to https://docs.python.org and click the modules link in the upper right corner. Find at least two built-in modules that interest you, and summarize what functions they provide.

See also https://github.com/vinta/awesome-python.

Nested Structures

Containers are objects that store other objects. For example, `list` stores a sequence of objects, and `dict` stores a mapping of objects to objects. Containers can also hold other containers, which makes it possible to represent any type (or shape) of data.

Content Learning Objectives

After completing this activity, students should be able to:

- Explain how rows and columns of data can be stored in lists.
- Write nested for loops to iterate data and compute functions.
- Navigate a complex data structure with dictionaries and lists.

Process Skill Goals

During the activity, students should make progress toward:

- Developing algorithms that loop through lists to compute a result. (Problem Solving)

Facilitation Notes

Given the complexity of the of examples, it's important to provide students with the source code in advance. See the accompanying `nested.py` file in the repository. The students should only have to type the expressions in the tables.

On Model 1, you might want to explain that `'R'` and `'Y'` represent Red and Yellow discs. It's less obvious what they mean when printed in black and white. You might also want to point out the optional comma after the last row. Python allows you to put a comma after every item in a list, including the last one, for convenience.

When reporting out Model 1 and Model 3, it's useful to show how these data structures look on Python Tutor (pythontutor.com). You might want to use the "render all objects on the heap (Python/Java)" option to discuss references.

Model 2 explores nested for loops in two ways: first with containers (`groceries` and `grid`) and then with the `range` function. Consider having teams write their solutions on the board, one team per question. If time permits, step through the solutions with a debugger or Python Tutor.


Copyright © 2019 T. Shepherd, C. Mayfield, and H. Hu. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

Model 1 Lists of Lists

Connect Four (® Hasbro, Inc.) is a two-player game in which the players take turns dropping colored discs into a six-row by seven-column grid. The objective of the game is to be the first player to form a horizontal, vertical, or diagonal line of four of one's own discs. (paraphrased from https://en.wikipedia.org/wiki/Connect_Four)


```
# current state of the game
grid = [
    [' ', ' ', ' ', ' ', ' ', ' ', ' '],
    [' ', ' ', ' ', ' ', ' ', ' ', ' '],
    ['Y', ' ', ' ', ' ', 'Y', 'Y', ' '],
    ['R', ' ', ' ', 'Y', 'R', 'R', ' '],
    ['R', 'R', 'Y', 'R', 'Y', 'R', ' '],
    ['R', 'Y', 'R', 'Y', 'Y', 'Y', 'R'],
]
```

Enter the `grid` code above into a Python Shell, and run each line of the table below. If the output is longer than one line, summarize it with a few words.

Python code	Shell output
`print(grid)`	prints the grid without line breaks
`print(grid[5])`	['R', 'Y', 'R', 'Y', 'Y', 'Y', 'R']
`print(grid[5][0])`	R
`type(grid)`	<class 'list'>
`type(grid[5])`	<class 'list'>
`type(grid[5][0])`	<class 'str'>
`len(grid)`	6
`len(grid[5])`	7
`len(grid[5][0])`	1
`import pprint`	
`help(pprint)`	Pretty-print a Python object to a stream
`pprint.pprint(grid)`	prints the grid on multiple lines
`for item in grid:` `  print(item)`	prints each row on a separate line
`for i in range(len(grid)):` `  print(grid[i])`	prints each row on a separate line

Questions (15 min) Start time: _____

1. What does `grid` look like when you first `print` it? (How is the output different from the original format shown in Model 1?)

It is not formatted neatly; the entire list prints with no line breaks.

2. What does `grid` look like when you use `pprint` instead? Explain what `pprint` means.

It looks rectangular and is much easier to read. The word `pprint` stands for "pretty-print".

3. When viewed as a rectangle, how many "rows" and "columns" does `grid` have?

There are six rows and seven columns.

4. What type of object is `grid`? What type of objects does it contain?

`grid` is a list, and its elements are also lists.

5. What type of object is `grid[5]`? What type of objects does it contain?

`grid[5]` is a list, and its elements are strings.

6. In the expression `grid[5][0]`, which index corresponds to the row, and which index corresponds to the column?

The first index [5] is the row, and the second index [0] is the column.

7. Is `grid` a list of rows or a list of columns? Justify your answer.

It is a list of rows; `len(grid)` is 6, and the `for` loop prints rows.

8. Describe how to append one more row to `grid`.

Simply use `grid.append([...])` to add the entire row in one step.

9. What is necessary to append a "column" to `grid`?

We need a `for` loop to append one string at a time to the end of each row.

Model 2 Nested `for` Loops

Example A
We typically use a `for` loop to examine the contents of a list:

```
1 groceries = ["Apples", "Milk", "Flour", "Chips"]
2 for item in groceries:
3     print("Don't forget the", item)
```

Example B
If a list contains another list, we need a `for` loop that contains another `for` loop. For example, to count the "spaces" in the `grid` from Model 1:

```
4 count = 0
5 for row in grid:  # outer loop
6     print("row =", row)
7     for cell in row:  # inner loop
8         print("cell =", cell)
9         if cell == ' ':
10            count += 1
11 print(count, "spaces remaining")
```

Questions (15 min) Start time: _____

10. As a team, discuss the two examples from Model 2. Predict how many times each of the following lines will execute. Then run the code and check your answers based on the output.

a) How many times does Line 3 execute? Predicted: _____ Actual: _____ 4

b) How many times does Line 6 execute? Predicted: _____ Actual: _____ 6

c) How many times does Line 8 execute? Predicted: _____ Actual: _____ 42

d) How many times does Line 10 execute? Predicted: _____ Actual: _____ 22

11. What determined how many times the "`for item`" loop would run? Number of groceries

12. Answer the following questions in terms of `grid`.

    a) What determined how many times the "`for row`" loop would run?

    The number of rows in the grid

    b) What determined how many times the "`for cell`" loop would run?

    The total number of cells in the grid (i.e., number of rows * number of cols)

13. In the example below, predict how many times the `print` statement will execute. Then run the code to verify your answer. 6 * 7 = 42 times

```
for i in range(6):
    for j in range(7):
        print(i, '+', j, '=', i + j)
```

14. Rewrite the nested `for` loops in Model 2 Lines 4–10 using the `range` function. Replace the variables `row` and `cell` with `i` and `j`, respectively. For simplicity, you may omit the `print` statements in your answer.

```
count = 0
for i in range(len(grid)):  # outer loop
    for j in range(len(grid[i])):  # inner loop
        if grid[i][j] == ' ':
            count += 1
```

15. Write a `for` loop (using `range`) that computes the factorial of a given integer n. Recall that $n! = n * (n-1) * (n-2) * \ldots * 1$. Store your result in a variable named `fact`.

```
fact = 1
for i in range(n, 0, -1):
    fact *= i
```

16. Write nested loops that compute and display the factorial of each integer from 1 to 20. Use your code from the previous question as the inner loop. Your output should be in this format:

```
The factorial of 1 is 1
The factorial of 2 is 2
The factorial of 3 is 6
The factorial of 4 is 24
The factorial of 5 is 120
```

```
for n in range(1, 21):
    fact = 1
    for i in range(n, 0, -1):
        fact *= i
    print("The factorial of", n, "is", fact)
```

Model 3 Nested Dictionaries

Containers can be nested in arbitrary ways. For example, the following data could be described as a "dictionary of dictionaries of integers and lists of strings".

Enter the following code into a Python Shell, and complete the table. If the output is longer than one line, summarize it with a few words.

```
movies = {
    "Casablanca": {
        "year": 1942,
        "genres": ["Drama", "Romance", "War"],
    },
    "Star Wars": {
        "year": 1977,
        "genres": ["Action", "Adventure", "Fantasy"],
    },
    "Groundhog Day": {
        "year": 1993,
        "genres": ["Comedy", "Fantasy", "Romance"],
    },
}
```

Python code	Shell output
`movies`	prints all of `movies` without any formatting
`movies["Casablanca"]`	{'genres': ['Drama', 'Romance', 'War'], 'year': 1942}
`movies["Casablanca"]["year"]`	1942
`movies["Casablanca"]["genres"]`	['Drama', 'Romance', 'War']
`type(movies)`	<class 'dict'>
`type(movies["Casablanca"])`	<class 'dict'>
`type(movies["Casablanca"]["year"])`	<class 'int'>
`type(movies["Casablanca"]["genres"])`	<class 'list'>
`len(movies)`	3
`len(movies["Casablanca"])`	2
`len(movies["Casablanca"]["year"])`	TypeError: object of type 'int' has no len()
`len(movies["Casablanca"]["genres"])`	3
`for key in movies:` `  print(key)`	prints the keys: Casablanca, Groundhog Day, Star Wars
`for key, val in movies.items():` `  print(key, val)`	prints each individual movie (the inner dictionaries)

Questions (15 min) Start time: _____

17. Explain the `TypeError` you encountered.

The expression `movies["Casablanca"]["year"]` is an integer, so you can't get the length of it.

18. In the expression `movies["Casablanca"]["genres"]`, describe the purpose of the strings `"Casablanca"` and `"genres"`.

They are keys to their corresponding dictionaries. The first string selects a particular movie, and the second string selects the corresponding movie data.

19. When iterating a dictionary using a `for` loop (i.e., `for x in movies`), what gets assigned to the variable?

The keys of the dictionary.

20. What is wrong with the following code that attempts to `print` each movie?

```
for i in range(len(movies)):
    print(movies[i])
```

You cannot iterate a dictionary by index number; it is not a sequence. Running this code results in `KeyError: 0`.

21. Write nested loops that output every *genre* found under the `movies` dictionary. You should have nine total lines of output.

```
for key in movies:
    movie = movies[key]
    for genre in movie["genres"]:
        print(genre)
```

22. Each movie in Model 3 has a title, a year, and three genres.

a) Is it necessary that all movies have the same format? No

b) Name one advantage of storing data in the same format: It simplifies the code

c) Show how you would represent The LEGO Movie (2014) with a runtime of 100 min and the plot keywords "construction worker" and "good cop bad cop".

```
"The LEGO Movie": {
    "year": 2014,
    "runtime": "100 min",
    "keywords": ["construction worker", "good cop bad cop"],
},
```

File Input/Output

Most data is stored in files, not input by the user every time. In this activity, you'll learn the basics of reading and writing plain text files.

Content Learning Objectives

After completing this activity, students should be able to:

- Create a new text file, and output several lines to it.
- Open an existing file, and append several lines to it.
- Read a text file line by line, and extract data from it.

Process Skill Goals

During the activity, students should make progress toward:

- Justifying answers based on the results of an experiment. (Critical Thinking)

Facilitation Notes

Be aware of what editors and operating systems your students are using. They might need help changing directories to find the input/output files used in this activity. Model 1 may take a little more time for setup and discussion. The source code should be provided (see `write.py` in the repository).

Model 2 switches from using an Editor to using the Shell. Help students figure out that the `write` method returns the number of characters written. Depending on the environment, they might not see the output file contents until after calling the `close` or `flush` method. If it comes up, discuss how buffering works when reporting out.

Notice that the input file for Model 3 is the `out.txt` file from Model 1 and Model 2. If students corrupt their `out.txt` file, some of their answers might be incorrect. If needed, you can give them a correct version of `out.txt` before they complete the Model 3 table. Report out after most teams reach Question #12, so that teams will have adequate time to answer the last question.

By the way, the `out.txt` data is in FASTA format, a standard in bioinformatics for representing sequences. See https://en.wikipedia.org/wiki/FASTA_format.

Copyright © 2019 T. Shepherd, C. Mayfield, and H. Hu. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

Model 1 Writing to a File

The following example creates a new file (in the current/default folder) named `out.txt` and writes several lines of output to it. Run the code, and examine the contents of the resulting `out.txt` file. In the space below, write the contents of `out.txt` to the right of the code.

```
1  outfile = open("out.txt", "w")
2  outfile.write("Example ")
3  outfile.write("output ")
4  outfile.write("text file\n")
5  outfile.write("xyz Coordinates\n")
6  outfile.write("MODEL\n")
7  outfile.write("ATOM %3d" % 1)
8  seq = "n %5.1f%5.1f%5.1f" % (0, 1, 2)
9  outfile.write(seq)
10 outfile.write("\n")
11 outfile.close()
```

out.txt

```
Example output text file
xyz Coordinates
MODEL
ATOM   1n    0.0   1.0   2.0
```

Questions (15 min)

Start time: _____

1. Based on the Python code:

   a) How many arguments are passed to `open`? What are their types? two strings

   b) What variable stores the ***file object*** returned by the `open` function? outfile

   c) Identify the names of all methods used on this file object in the code. write, close

   d) What type of data does the `write` method require for its argument? string

2. Based on the `out.txt` file:

   a) How many times was the `write` method called to create the first line of text? 3

   b) How many times was the write method called to create the second line of text? 1

   c) What does the `"\n"` character do? It ends the current line.

   d) How is the `write` method different from the `print` function? Doesn't append a newline.

3. Write a program that creates a file named `lines.txt` and writes 100 lines like this:

```
Line #1
Line #2
Line #3
Line #4
...
```

```
outfile = open("lines.txt", "w")
for i in range(1, 101):
    outfile.write("Line #%d\n" % i)
outfile.close()
```

Model 2 Appending to a File

The second argument of `open` specifies the ***mode*** in which the file is opened. When writing output to a file, there are two basic modes:

- The write (`"w"`) mode will overwrite/replace the file contents.
- The append (`"a"`) mode will add new data to the end of the file.

Either mode will create the file automatically if it does not already exist. Enter the following lines into a Python Shell, and record the output at each step.

Python code	Shell output
`afile.write("new line\n")`	NameError: name 'afile' is not defined
`afile = open("out.txt", "a")`	
`afile.write("new line\n")`	9
`afile.write(2.0)`	TypeError: write() argument must be str, not float
`afile.write("2.0")`	3
`afile.close()`	
`afile.write("new line\n")`	ValueError: I/O operation on closed file.

Questions (10 min) Start time: _____

4. Explain what happens as a result of the line: `afile = open("out.txt", "a")`

The existing out.txt file (from Model 1) is open for appending.

5. How do the arguments passed to the `open` function differ for writing a new file in comparison to appending an existing file?

The second argument is `"a"` instead of a `"w"`.

6. What does the `write` method return? Run `help(afile.write)` to check your answer.

The number of characters written (which is always equal to the length of the string).

7. Explain the reason for the error observed after entering:

   a) the first line of code: `afile.write("new line\n")` the file wasn't open

   b) the last line of code: `afile.write("new line\n")` the file wasn't open

   c) the statement: `afile.write(2.0)` you can only write strings to files

Model 3 Reading from a File

Programs often require input data from an external file source. Not surprisingly, there are methods for reading the contents of files. Enter the following lines into a Python Shell.

Python code	Shell output
`infile = open("out.txt", "r")`	
`infile.readline()`	'Example output text file\n'
`infile.readline()`	'xyz Coordinates\n'
`infile.readlines()`	list of the remaining lines
`infile.readline()`	''
`infile.close()`	
`infile = open("out.txt", "r")`	
`for line in infile:` `  print(line)`	prints each line double spaced
`infile.close()`	
`infile = open("out.txt", "r")`	
`for i in range(3):` `  infile.readline()`	
`line = infile.readline()`	
`line`	'ATOM 1n 0.0 1.0 2.0\n'
`print(line[0])`	A
`print(line[0:5])`	ATOM
`words = line.split()`	
`words`	['ATOM', '1n', '0.0', '1.0', '2.0']
`print(words[0])`	ATOM
`infile.close()`	

Questions (20 min) Start time: _____

8. Based on the output above:

   a) What type of data does the `readline` method return? string

   b) What type of data does the `readlines` method return? list of strings

9. Why did the `readline` method return different values each time?

Each time `readline` is called, it returns the next line of the file.

10. What happens if you try to read past the end of the file? Justify your answer.

From then on, `readline` returns `''`, and `readlines` returns an empty list.

11. What is the difference between the two `for` loops in Model 3?

The first loop iterates and prints every line of the file. The second for loop reads only the first three lines, making it possible to assign the fourth line in the next statement.

12. Consider the output of the first `for` loop:

    a) Why does the program display the file as if it were double spaced?

    Each line in the file ends with a newline character `\n`, and `print` adds another one.

    b) How would you change the code to avoid printing extra blank lines?

    Change the function: `print(line, end='')` Or change the string: `print(line[:-1])`

13. Based on the second half of Model 3:

    a) Why was it necessary to open the file again? The data had already been read previously.

    b) Write code that would output 1.0 using `line` print(line[17:20])

    c) Write code that would output 1.0 using `words` print(words[3])

14. Consider a file `names.txt` that contains first and last names of 100 people, with one name per line (e.g., "Anita Borg"). Write a program that prints all the last names (the second word of each line) in the file.

```
infile = open("names.txt", "r")
for line in infile:
    words = line.split()
    print(words[1])
infile.close()
```

Visualizing Data

As a general-purpose programming language, Python is incredibly useful for analyzing data and visualizing results. This activity is a first look at `matplotlib`, one of the most widely used 2D plotting libraries.

Content Learning Objectives

After completing this activity, students should be able to:

- Explain the basic structure of code for plotting a mathematical function.
- Analyze visually the behavior of the Python random number generator.
- Read data from a CSV file and generate histograms of various columns.

Process Skill Goals

During the activity, students should make progress toward:

- Navigating the documentation for a third-party library. (Information Processing)

Facilitation Notes

For each of the examples in this activity, it helps to run the code on the projector and discuss the results visually when reporting out. On Model 1, be sure to explain the `arange` function and how it works differently from `range`. For example, `range` requires an integer for the third argument, but `arange` allows floats. If time permits, explain briefly what an `array` is in `numpy`.

In addition to plotting histograms, the point of Model 2 is to give students further insight to how random numbers work. Have teams come up with an explanation of why some random numbers appear to occur more often than others (when the sample size is low). Demonstrate the results of #12 on the board to facilitate discussion.

On Model 3, students may need help counting rows and columns in the `data.csv` file. Check their answers for the first three questions before they proceed to the Scorecard Data. Questions #18 and #19 can take a lot of time if they are not savvy with Excel. You might want to answer these questions for the entire class on the projector (e.g., demonstrate how to find the range of values by sorting the data).

The NULL values in the Scorecard Data will likely confuse students. Monitor team discussions, and as needed, point out that "NULL" is a literal string in the file contents. It's not interpreted as `None` in Python.

Copyright © 2019 C. Mayfield and T. Shepherd. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

Model 1 Simple Plot

When analyzing data, it's helpful to create charts, plots, and other visualizations. Doing so allows you to see important numerical relationships. Enter the following code into a Python Editor, and run the program.

```
1  import matplotlib.pyplot as plt
2  import numpy as np
3
4  def model_one():
5      x = np.arange(0.0, 2.0, .01)
6      y = np.sin(2 * np.pi * x)
7      plt.plot(x, y)
8      plt.xlabel('time (s)')
9      plt.ylabel('volts (mV)')
10     plt.show()
11
12 if __name__ == "__main__":
13     model_one()
```

Questions (15 min) Start time: _____

1. Identify in the source code which line numbers:

   a) generated the data? 5–6

   b) set the axes properties? 8–9

   c) displayed the window? 10

   d) plotted the actual data? 7

2. Describe in your own words what is being plotted.

It plots a sine wave with an amplitude of 2, a period of 1, and a vertical shift of 1. The x and y values range from 0 to 2.

3. Modify the code to plot only one cycle of the sine wave (instead of two). Write the edited line of code below.

`x = np.arange(0.0, 1.0, .01)` OR `y = np.sin(np.pi * x)`

4. Change the third argument of `np.arange` from 0.01 to 0.15. What is the result?

Less curvature; it almost looks like a triangle.

5. Add `"o"` as a third argument to the `plot` function. What is the result?

Plots points instead of a line.

6. How does the third parameter of `np.arange` affect how the plot looks?

It determines how many points to generate, which makes the plot look smoother, but it also takes longer to draw.

7. How would you modify the code to plot the function $y = x^2 - 1$ instead? Show the results from -2 to +2.

Replace Line 5 with `x = np.arange(-2.0, 2.0, 0.01)` and Line 6 with `y = x ** 2 - 1`.

8. Which three Python libraries are used in Model 1? Quickly search the Internet and find their websites. Write a one-sentence description about each library.

- `matplotlib`: 2D plotting library which produces publication quality figures.
- `pyplot`: Provides a MATLAB-like plotting framework (part of matplotlib).
- `numpy`: Fundamental package for scientific computing (part of SciPy).

Model 2 Histograms

Recall that you can generate a sequence of numbers using the `random` module. Merge the code below into your program from Model 1. Run the program, and view the output.

```
1  import matplotlib.pyplot as plt
2  import random
3
4  def model_two(npts):
5      numbers = []
6      for _ in range(npts):
7          numbers.append(random.random())
8      plt.hist(numbers)
9      plt.show()
10
11 if __name__ == "__main__":
12     model_two(100)
```

Questions (10 min) Start time: _____

9. Based on the Python code:

   a) What is the range of values generated by the `random` function? [0, 1)

   b) How many random values are generated? 100

10. Based on the figure plotted:

   a) How many bars are displayed? 10

   b) What is the width of each bar? 0.1

   c) What is the sum of the heights of the bars? 100

11. Based on your answers above, what are appropriate labels for the x and y axes?

The x-axis could be "random number value", and the y-axis could be "frequency" (number of times generated).

12. Increase the argument of `model_two` to 1000, 10000, and 100000. Describe how the output plot changes when you run the program.

The bars get skinnier and more uniform. The y-axis also increases in range. And it takes a lot longer to plot.

13. Add the number 50 as second argument to the `hist` function. What is the meaning of the result?

It plots 50 bars instead of 10, and it looks look less uniform. There are more ranges in which the numbers may fall.

14. In general, describe what the `hist` function does with the list of random numbers to create this type of plot.

It groups them into bins (based on the second argument of `hist`), and draws a bar for the number of values in each bin.

Model 3 CSV Data

"Comma Separated Values" is a common file format when exporting data from spreadsheets and databases. Each line of the file is a row, and each column is separated by a comma. Cells that contain commas are wrapped in quote marks.

data.csv file contents:

```
Name,Location,URL,Students
Westminster College,"Salt Lake City, UT",westminstercollege.edu,2135
Muhlenberg College,"Allentown, PA",muhlenberg.edu,2330
University of Maine,"Orono, ME",umaine.edu,8677
James Madison University,"Harrisonburg, VA",jmu.edu,19019
Michigan State University,"East Lansing, MI",msu.edu,38853
```

Python includes a `csv` module (https://docs.python.org/3/library/csv.html) that makes it easy to read and write CSV files.

```
import csv

infile = open("data.csv")
data = csv.reader(infile)
names = next(data)  # column names
for row in data:
    print(row[1])   # 2nd column
```

Program output:

```
Salt Lake City, UT
Allentown, PA
Orono, ME
Harrisonburg, VA
East Lansing, MI
```

Questions (20 min) Start time: _____

15. In the example `data.csv` file above:

    a) In what way is the first line different? It has the column names (not data)
    b) How many rows of data are there? 5
    c) How many columns are there? 4

16. Compare `data.csv` with the program output:

    a) Are quote marks included in the lines of data? Yes
    b) Are quote marks included in the program output? No
    c) What is the purpose of the quote marks? To specify a value that contains commas

17. In the Python code above:

    a) Which line of code reads the first line of the file? names = next(data)
    b) What type of data does the variable `row` contain? A list of strings

In 2013, the U.S. Department of Education released the "College Scorecard" website to help students and families compare institutions of higher education. The Scorecard data includes information like average cost of attendance, graduation and retention rates, student body demographics, etc.

18. Download the "Scorecard Data 7 MB CSV" from https://collegescorecard.ed.gov/data/ (listed halfway down under "Featured Downloads"). Open the CSV file in Excel or a similar program, and skim its contents.

a) How many rows does it have? 7058, not counting the header row

b) How many columns does it have? 190 (A to GH)

19. Column CH is named UGDS, which means "Enrollment of undergraduate certificate / degree-seeking students".

a) What is the range of values in this column? 0 to 77,269

b) Which school has the most students enrolled? University of Phoenix-Arizona

c) Do all rows have an integer value for UGDS? No, many are NULL

20. Based on the code in Model 2 and Model 3, write a program that plots a histogram of the UGDS column. Complete the following steps to consider each part of the program.

a) What two import statements will you need at the top?

```
import csv
import matplotlib.pyplot as plt
```

b) What three statements prepare the csv file for reading?

```
infile = open("Most-Recent-Cohorts-Scorecard-Elements.csv")
data = csv.reader(infile)
names = next(data)  # column names
```

c) What code is necessary to read the entire column into a list? (Note: Column CH in Excel is `row[85]` in Python.)

```
ugds = []
for row in data:
    ugds.append(row[85])
```

d) By default, data from text/csv files are read as strings. Write the code to convert the `row[85]` values to integers. Be sure not include the `"NULL"` values in the final list.

```
if row[85] != 'NULL':
    size = int(row[85])
    ugds.append(size)
```

e) Write the last two lines that plot and show the histogram.

```
plt.hist(ugds)
plt.show()
```

21. Run the program, and compare your results with another team's. What does the histogram tell you about undergraduate enrollments in the United States?

There are thousands of schools with less than 1,000 students. Very few schools have 15,000 students or more.

22. What other questions could you ask about this data? How would you answer them using histograms, line charts, and scatter plots?

Answers will vary. For example, it would be nice to see the number of schools per state.

Defining Classes

In this activity, we'll take a first look at object-oriented programming. Classes provide a means of bundling data and functionality together.

Content Learning Objectives

After completing this activity, students should be able to:

- Write a class definition that has several attributes and methods.
- Explain what a constructor is, when it is called, and what it does.
- Discuss what "object-oriented" means using concrete examples.

Process Skill Goals

During the activity, students should make progress toward:

- Developing and testing the design of a program incrementally. (Problem Solving)

Facilitation Notes

NOTE: This activity was designed with an "objects late" approach in mind. It might not be suitable for an "objects early" course, given the order in which the example class is presented.

The source code for Model 1 should be provided to students (see `atom.py` in the repository). When reporting out, focus the discussion on Questions #5 and #7. It may be helpful to step through the code with a debugger. Students with prior experience might wonder why Line 19 doesn't raise an error (since the attribute wasn't "defined" in a constructor).

Model 2 solves the problem introduced in Question #7, namely that different objects might have different attributes. Don't let students get bogged down with the chemistry examples; it's not essential that they understand the details. (An isotope is one of two or more atoms with the same atomic number but with different numbers of neutrons.)

By Model 3, students should have a complete version of the `Atom` class implemented with all four methods. You may need to guide the discussion and report out both during the middle and the end of these questions. Give multiple teams the opportunity to explain object-oriented programming in their own words.


Copyright © 2019 T. Shepherd, C. Mayfield, and H. Hu. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

Model 1 Attributes and Methods

Previously you have used built-in types like `int`, `str`, and `list`. Each of these types comes with its own ***methods***, such as `isdigit` and `append`. You can create new types of data, and methods to go with them, by defining a `class`. Classes specify the ***attributes*** (instance variables) and methods (functions) that each object belonging to the class will have.

```
1  class Atom:
2      """An element from the periodic table."""
3
4      def neutrons(self):
5          """Returns the number of neutrons the element has"""
6          number = self.isotope - self.atomic
7          print("%s has %d neutrons" % (self.symbol, number))
8          return number
9
10     def grams_to_moles(self, grams):
11         """Converts the mass of an element in grams to moles"""
12         moles = grams / self.mass
13         print("%.1f g is %.1f moles of %s" % (grams, moles, self.symbol))
14         return moles
15
16 if __name__ == "__main__":
17
18     oxygen = Atom()  # create an Atom object
19     oxygen.symbol = 'O'
20     oxygen.atomic = 8
21     oxygen.mass = 15.999
22     oxygen.isotope = 16
23     carbon = Atom()  # create another Atom object
24     carbon.symbol = 'C'
25     carbon.mass = 12.001
26     oxygen.neutrons()
27     oxygen.grams_to_moles(24)
28     carbon.grams_to_moles(24)
```

Questions (15 min) Start time: _____

1. Examine the ***class definition*** (the top half of the code):

   a) What is the name of the class? Atom

   b) What are the names of the two methods? `neutrons` and `grams_to_moles`

   c) What is the name of the first parameter for all methods? self

2. Now examine the `"__main__"` block of code:

   a) How many different `Atom` objects were created? two

   b) Identify the variable name of each object. oxygen and carbon

   c) How many attributes were assigned in the `oxygen` object? List the names.

      4: symbol, atomic, mass, isotope

   d) How do the number of arguments for each method call differ from the number of parameters specified in the method definition? one less

3. How does the syntax referencing an attribute differ inside vs. outside the class definition?

inside class: self.identifier vs. outside class: object.identifier

4. When the `grams_to_moles` method is called (in the last two lines), what is the value of the `self` parameter?

In this example, oxygen or carbon (depending on which object was before the method call).

5. Enter the expression `type(oxygen)` in a Python Shell. Explain the meaning and significance of the output.

The output is `<class '__main__.Atom'>`, meaning that `oxygen` belongs to the `Atom` class defined in the `__main__` module. It's significant because we are the ones who defined this data type.

6. Write code to create a new `Atom` object called `hydrogen`, and assign one of the attributes listed in Question #2c.

```
hydrogen = Atom()
hydrogen.symbol = 'H'
```

7. Call the `neutrons` method on carbon in a Python Shell. What is the reason for the error?

The attributes `self.symbol`, `self.isotope`, and `self.atomic` have not yet been assigned.

Model 2 Constructors

For each class defined, you can provide a ***constructor*** that initializes attributes of a new object. In Python, the constructor is always named `__init__` (with two underscores before and after the word `init`). The constructor is called automatically when you create a new object.

Add the following constructor to the top of your `Atom` class. By convention, the constructor is typically the first method in a class definition. Also edit the `"__main__"` block of code as shown.

```
class Atom:
    """An element from the periodic table."""

    def __init__(self, symbol, atomic, mass, isotope=12):
        """Constructs an Atom with the given values."""
        self.symbol = symbol
        self.atomic = atomic
        self.mass = mass
        self.isotope = isotope

    ... previous methods from Model 1 ...

if __name__ == "__main__":

    oxygen = Atom('O', 8, 15.999, 16)
    carbon = Atom('C', 6, 12.001)
    oxygen.neutrons()
    carbon.neutrons()
    oxygen.grams_to_moles(24)
    carbon.grams_to_moles(24)
```

Questions (15 min) Start time: _____

8. What is always the name of the constructor?

`__init__`

9. Although there is no direct call to the constructor, explain how you know this method is executed when an object is created.

The instance variables have a value when used in the methods `neutrons` and `grams_to_moles`. There is no error message this time.

10. Consider your answer to Question #7. What is one advantage of defining a constructor for a class?

It automatically initializes the attributes so that methods don't error if they aren't called in the correct order.

11. In a Python Shell, try to create a new `Atom` object called `hydrogen` with only two arguments. Write your statement in the space below. What is the reason for the error you see?

`hydrogen = Atom('H', 1)` Based on the definition of `__init__`, at least three parameters are required to define an `Atom` object.

12. When creating an object of the `Atom` class, what is the value of `isotope` if:

a) three arguments are given? 12

b) four arguments are given? the value of the last argument

13. Print the value of `self.isotope` in a Python shell.

a) What is the reason for the error? NameError: 'self' is not defined

b) In order to eliminate this error, what should be printed instead? oxygen.isotope

14. For each line below, what is the value of `self`?

a) oxygen = Atom('O', 8, 15.999, 16) The object that will be assigned to `oxygen`

b) carbon = Atom('C', 6, 12.001) The object that will be assigned to `carbon`

c) oxygen.neutrons() The object currently referenced by `oxygen`

d) carbon.neutrons() The object currently referenced by `carbon`

15. Recall that a variable may be "local" (defined within a function), "global" (defined in the non-indented or `"__main__"` block of code), or "built-in" (part of Python itself).

a) Explain why the `isotope` attribute is not a global variable.

it's defined in the `__init__` function (not `__main__`)

b) Explain why the `isotope` attribute is not a local variable.

it's used outside the constructor in other methods

c) How is each method of the class able to access the `isotope` attribute?

Through the `self` parameter, which is a local variable in each method.

Model 3 Object-Oriented

Edit the `Atom` class further to include the variable `avogadros`, the method `grams_to_atoms`, and the modified `"__main__"` block of code. Note that ***class variables*** (like `avogadros`) are typically defined before the `__init__` method.

```
class Atom:
    """An element from the periodic table."""

    avogadros = 6.02E23

    ... previous methods from Model 2 ...

    def grams_to_atoms(self, weight):
        """Converts the mass of an element in grams to number of atoms."""
        answer = Atom.avogadros * self.grams_to_moles(weight)
        print("%.1f g is %.1e atoms of %s" % (weight, answer, self.symbol))
        return answer

if __name__ == "__main__":

    oxygen = Atom('O', 8, 15.999, 16)
    carbon = Atom('C', 6, 12.001)
    oxygen.neutrons()
    oxygen.isotope = 18
    oxygen.neutrons()
    oxygen.grams_to_atoms(24)
    carbon.grams_to_atoms(24)
```

Questions (15 min) Start time: _____

16. Examine the `grams_to_moles` method (from Model 1):

    a) Identify the three main variables used in `grams_to_moles`:

    self, grams, moles

    b) For each variable, what is its scope? (local or global)

    local, local, local

17. What determines whether a variable is defined as an attribute or a local variable?

The `self` parameter defines (stores) all the attributes. Local variables, like `moles`, are defined in methods directly.

18. Now examine the `grams_to_atoms` method (from Model 3).

a) What variable was initialized in the `Atom` class outside the constructor and methods?

Atom.avogadros

b) How does the syntax of a class variable differ from an attribute (instance variable)?

It uses the class name in the identifier rather than `self` or the object name.

19. Would it be possible to rewrite the `grams_to_atoms` method as a global function instead? If so, explain how the function would differ.

Yes; simply define the function outside of the class. Rename the parameter `self` to `atom` to avoid confusion.

20. How would you rewrite the line `oxygen.grams_to_atoms(24)` to call the global function defined in the previous question?

```
grams_to_atoms(oxygen, 24)
```

21. Consider the built-in `str` class:

a) Given the statement `s = "Hello"`, what data is stored in the `str` object?

The word Hello.

b) Show an example line of code that calls the `upper` method on the object `s`.

print(s.upper())

c) If the `upper` method were defined as a global function instead, how would you call it?

print(upper(s))

22. Based on the previous two questions, explain what the term "object-oriented" means.

It means that methods are built into objects, as opposed to being standalone functions. We call those methods through the objects.

23. Summarize the advantages you perceive for writing code as methods in classes instead of global functions.

Either way, we have to pass the object to the function. It's easier to think about the object first, and then call methods that take the remaining parameters.

Extending Classes

A major benefit of object-oriented programming is the ability to inherit classes and eliminate duplicate code. Inheritance allows you to define a new class based on an existing class.

Content Learning Objectives

After completing this activity, students should be able to:

- Read and interpret UML class diagrams for an existing code base.
- Evaluate pros and cons for designs with multiple similar classes.
- Define inheritance and demonstrate how to extend a base class.

Process Skill Goals

During the activity, students should make progress toward:

- Working with all team members to reach consensus on hard questions. (Teamwork)

Facilitation Notes

This activity involves a lot more source code than usual, but it's doable if students don't get lost in the details. Encourage teams to manage their time well and not to think too deeply about any of the questions. Point out that UML is helpful to summarize the classes without looking at all the source code. Be sure to provide students with all the files in advance (see `animals.zip`).

On #5, have them quickly write a single word for each table cell. It's not that important what they write, as long as it connects back to the source code. The important part of Model 1 is the last question. Have more than one team share their answers with the class to facilitate discussion about object-oriented design.

The attributes and methods in the source code (and UML diagrams) are sorted alphabetically, but the parameters of the `__init__` method are in a different order. You might want to point out this detail when students examine the source code from Model 2. Spend time reporting out the last two questions about the advantages and disadvantages of the single-class design.

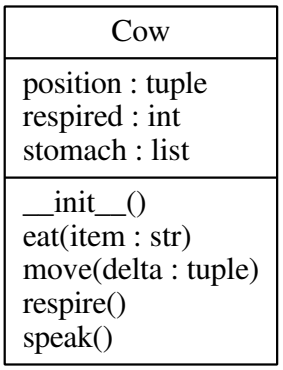
As with the other models, the most important question for Model 3 is the last one. Be sure to save time for reporting out the advantages of using inheritance. If time permits, show other examples of inheritance (with UML class diagrams). In particular, GUI libraries make extensive use inheritance (see e.g., https://docs.python.org/3/library/tk.html).

Copyright © 2019 M. Stewart and C. Mayfield. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

Model 1 UML Class Diagrams

Unified Modeling Language (UML) provides a standard way of visualizing how programs are designed (http://www.uml.org/what-is-uml.htm). For example, a ***class diagram*** is a graphical summary of the attributes and methods of a class.

```
class Cow:

    def __init__(self):
        self.position = (0, 0)
        self.respired = 0
        self.stomach = []

    def eat(self, item):
        if item == "grass":
            self.stomach.append(item)
            return "ate " + item
        else:
            return "This herbivore doesn't eat " + item

    def move(self, delta):
        self.position = (self.position[0] + delta[0],
                         self.position[1] + delta[1])
        return "This quadruped walked to " + str(self.position)

    def respire(self):
        self.respired += 1
        return "This cow respired through its nostrils."

    def speak(self):
        return "moo"
```



Questions (15 min) Start time: _____

1. Draw an arrow from each name in the diagram to where it's defined in the code.

2. What are the attributes of `Cow`? What are the methods of `Cow`?

The attributes are `position`, `respired`, and `stomach`. The methods are `__init__`, `eat`, `move`, `respire`, and `speak`.

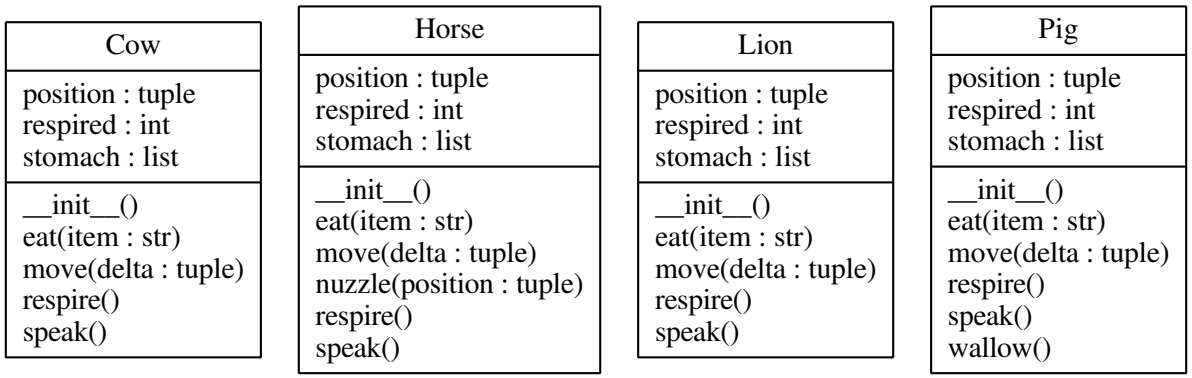
3. What is listed in each section of the UML class diagram?

a) Top section: class name

b) Middle section: attributes

c) Bottom section: methods

4. Consider the following class diagrams:



a) What attributes do the classes have in common?

All have the same three: `position`, `respired`, and `stomach`.

b) What methods do the classes have in common?

All five have `__init__`, `eat`, `move`, `respire`, and `speak`.

c) What methods are unique to a particular class?

Horse has `nuzzle`, and Pig has `wallow`.

5. Quickly examine the source code for each of the classes to identify similarities and differences. Write one or two words in each table cell to summarize your findings.

	Cow	**Horse**	**Lion**	**Pig**
eat	grass	grass	meat	everything
move	walked	galloped	walked	ran
nuzzle	N/A	moved	N/A	N/A
respire	nostrils	muzzle	nostrils	snout
speak	moo	nay	ROOOAR	oink
wallow	N/A	N/A	N/A	in water

6. Consider what it would take to add a new method named `sleep` to each of the classes.

a) Describe the process of adding the same method to each source file.

Write the method in one class, and copy/paste into the others.

b) If a mistake is found later on, how would you correct the method?

Fix the problem in one class, and copy/paste into the others.

c) What problems do you see with this approach as more classes are added?

The amount of copying and pasting and other maintenance increases.

Model 2 Single-Class Approach

Given that the classes from Model 1 are similar, we could try combining them into a single class. In order to keep track of differences, we would need to store additional attributes. We could provide the information from Question #5 when creating an object:

```
angus = Animal("Cow", "grass", "walked", "nostrils", "moo")
```

The UML diagram below outlines this approach. As a team, discuss this design and become familiar with the accompanying source code.

Animal
eats_only : str moved : str name : str nasal : str position : tuple respired : int sound : str stomach : list
__init__(name : str, eats_only : str, moved : str, nasal : str, sound : str) eat(item : str) move(delta : tuple) nuzzle(position : tuple) respire() speak() wallow()

Questions (10 min) Start time: _____

7. Circle the three original attributes that were defined in Model 1.
Circled: position, respired, and stomach

8. Circle the two methods that were NOT common to all four classes in Model 1.
Circled: nuzzle and wallow

9. Write a statement that creates an `Animal` object representing a lion. Assign it to a variable named `simba`.

```
simba = Animal("Lion", "meat", "walked", "nostrils", "ROOOAR")
```

10. What methods does `simba` now have that it did not have in Model 1?

nuzzle and wallow

11. Describe 1–2 advantages the Model 2 design has compared to Model 1.

There's only one class. You don't have several copies of the same code lying around.

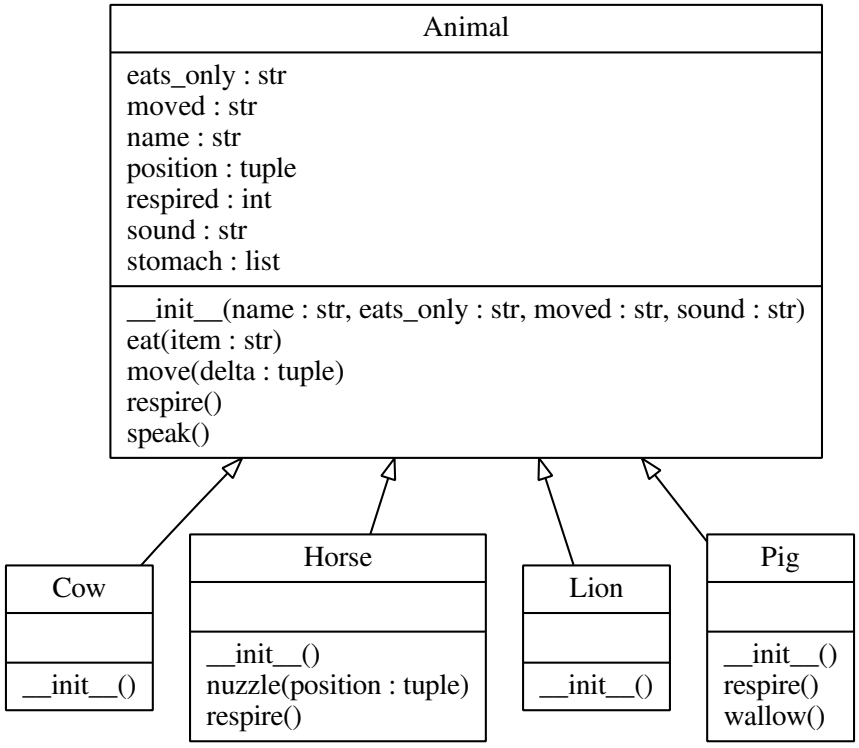
12. Describe 1–2 disadvantages the Model 2 design has compared to Model 1.

It's awkward to use the constructor. And not every object should have every method.

Model 3 Derived Classes

We can improve the code from Model 2 by using ***derived classes*** for `Cow`, `Horse`, `Lion`, and `Pig`. These classes only contain the attributes and methods specific to them. `Animal` is a ***base class*** that contains attributes and methods they all have in common. This language feature is known as ***inheritance***, because derived classes "inherit" attributes and methods from the base class.

The UML diagram below outlines this approach. As a team, discuss this design and become familiar with the accompanying source code.



Questions (20 min) Start time: _____

13. Open the `lion.py` source file. How many methods are defined in the `Lion` class? List the name of each one.

Just one: the `__init__` method.

14. Type the following code into a Python Shell (in the same location as the source files). What methods are listed in the help?

```
from lion import Lion
help(Lion)
```

In addition to the `__init__` method, it lists the methods inherited from animal.Animal (the base class).

15. Write a statement that creates a `Lion` object. Assign it to a variable named `simba`. How is this statement different from Question #9?

```
simba = Lion()
```

It's much easier than before, because the `Lion` constructor is doing the work for you.

16. In a Python Shell, what is the value of `simba.speak()`? Where did this value come from?

"ROOOAR". The constructor passed this value to `Animal`, which prints it out later.

17. Does `simba` have any methods that it did not have in Model 1? Justify your answer.

No, it does not. It only inherits the same four that it had before.

18. Based on the source files, how does the `__init__` method of `Animal` differ from the `__init__` methods of the derived classes?

`Animal` defines and stores the value of all the parameters. The other classes simply call `Animal`'s constructor.

19. What is the meaning of the built-in function `super()` that is used in the derived classes?

It refers to the base class and is used to call the `__init__` method of `Animal`.

20. Describe 1–2 advantages the Model 3 design has compared to Model 2.

Every class has its own name, and objects don't have methods they're not supposed to have. The code is also easier to extend when adding new animals.

Recursive Functions

Sometimes when solving a problem, we can compute the solution of a simpler version of the same problem. Eventually we reach the most basic version, for which the answer is known.

Content Learning Objectives

After completing this activity, students should be able to:

- Identify the base case and recursive step of the factorial function.
- Trace a recursive function by hand to predict the number of calls.
- Write short recursive functions based on mathematical sequences.

Process Skill Goals

During the activity, students should make progress toward:

- Evaluating mathematical functions to gain insight on recursion. (Information Processing)

Facilitation Notes

This activity is a first introduction to recursion using mathematical examples: factorial function, Fibonacci numbers, and summation. Students learn how to read, trace, and write recursive functions, but not yet how to formulate recursive solutions to problems.

Let students wrestle with these difficult concepts, and don't give too much help on individual questions. On Model 1, keep an eye on questions 1–3, and if a team is getting off track, have them compare answers with a neighboring team. On #4, you might have to help teams refer back to #2: they simply need to translate their previous answers into Python syntax.

When students get to Model 2, make sure they don't spend more then 2–3 minutes on the first question. It's okay for them to make a mistake and correct it during #8. Some teams may get sidetracked and spend too much time attempting to trace the recursion by hand. Report out this model by discussing the last two questions as a class.

Model 3 should move a bit faster than Model 1, given how similar they are. Ask presenters to write their team's solution to #16 on the board. Address misconceptions about variables, parameter passing, and return values. Have teams show you their working program for the last question.

Copyright © 2019 C. Mayfield and H. Hu. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

Model 1 Factorial Function

"In mathematics, the factorial of a non-negative integer n, denoted by $n!$, is the product of all positive integers less than or equal to n. For example, $5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$."

Source: https://en.wikipedia.org/wiki/Factorial

n	$n!$
0	1
1	1
2	2
3	6
4	24
5	120

Questions (15 min)

Start time: _____

1. Consider how to calculate $4! = 24$.

   a) Write out all the numbers that need to be multiplied:

   $4! =$ 4 * 3 * 2 * 1

   b) Rewrite the expression using 3! instead of $3 \times 2 \times 1$:

   $4! =$ 4 * 3!

2. Write expressions similar to #1b showing how each factorial can be calculated in terms of a smaller factorial. Each answer should end with a factorial (!).

   a) $2! =$ 2 * 1!

   b) $3! =$ 3 * 2!

   c) $100! =$ 100 * 99!

   d) $n! =$ $n * (n-1)!$

3. What is the value of 0! based on Model 1? Does it make sense to define 0! in terms of a simpler factorial? Why or why not?

0! is 1 (by convention for an empty product). We can't say $0 \times -1!$, because factorial is only defined for non-negative integers. At some point we need to define the solution in concrete terms, without referencing itself.

If we repeatedly break down a problem into smaller versions of itself, we eventually reach a basic problem that can't be broken down any further. Such a problem, like 0!*, is referred to as the* **base case**.

4. Consider the following Python function that takes n as a parameter and returns $n!$:

```
1 def factorial(n):
2     # base case
3     if n == 0:
4         return 1
5     # general case
6     product = 1
7     for i in range(n, 0, -1):
8         product *= i
9     return product
```

a) Review your answer to #2c that shows how to compute 100! using a smaller factorial. Convert this expression to Python by using the function above instead of the ! operator.

```
100 * factorial(99)
```

b) Now rewrite your answer to #2d in Python using the variable `n` and the function above.

```
n * factorial(n - 1)
```

c) In the source code above, replace the "1" on Line 6 with your answer from b). Then cross out Lines 7 and 8. Test the resulting function in a Python Shell. Does it still work?

Yes, amazingly.

d) What specific function is being called on Line 6?

The `factorial` function that is being defined.

e) Why is the `if` statement required on Line 3?

Without the base case, it would call itself forever (until running out of memory).

5. A function that refers to itself is called **recursive**. What two steps were necessary to define the recursive version of `factorial`?

1. Write the base case, which was implemented using an if statement.
2. Write the recursive case, which was implemented using a function call.

6. Was a loop necessary to cause the recursive version of `factorial` to run multiple times? Explain your reasoning.

No; the function keeps calling itself until the base case is reached. The `for` loop was removed when we added the recursive call.

Model 2 Fibonacci Numbers

The Fibonacci numbers are a sequence where every number (after the first two) is the sum of the two preceding numbers: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ...

Source: https://en.wikipedia.org/wiki/Fibonacci_number

We can define a recursive function to compute Fibonacci numbers. Enter the following code into a Python Editor, and run the program to see the sequence.

```
1  def fibonacci(n):
2      # base case
3      if n == 1 or n == 2:
4          return 1
5      # general case
6      return fibonacci(n - 1) + fibonacci(n - 2)
7
8  if __name__ == "__main__":
9      for i in range(1, 6):
10         print(fibonacci(i))
```

Questions (10 min) Start time: _____

7. Based on the source code:

   a) How many function calls are needed to compute `fibonacci(3)`? Identify the value of the parameter `n` for each of these calls.

   3 calls: n=3, n=2, n=1

   b) How many function calls are needed to compute `fibonacci(4)`? Identify the value of the parameter `n` for each of these calls.

   5 calls: n=4, n=3, n=2, n=1, n=2

   c) How many function calls are needed to compute `fibonacci(5)`? Identify the value of the parameter `n` for each of these calls.

   9 calls: n=5, n=4, n=3, n=2, n=1, n=2, n=3, n=2, n=1

8. Check your answers for the previous question by adding the following `print` statements to the code and rerunning the program:

   - Insert `print("n is", n)` at Line 2, before the `# base case` comment
   - Insert `print("fib(%d) is..." % i)` at Line 10, before the `print` statement

9. What happens if you try to compute `fibonacci(0)` in the Python Shell?

The function is called with decreasing negative numbers until the Shell displays the error: "RuntimeError: maximum recursion depth exceeded while calling a Python object".

10. How could you modify the code so that this situation doesn't happen?

Add an `if` statement that returns -1 if $n < 1$.

Model 3 Summation

"In mathematics, summation (capital Greek sigma symbol: Σ) is the addition of a sequence of numbers; the result is their sum or total."

$$\sum_{i=1}^{100} i = 1 + 2 + 3 + \ldots + 100 = 5050$$

Source: https://en.wikipedia.org/wiki/Summation

Questions (20 min) **Start time:** _____

11. Consider how to calculate $\sum_{i=1}^{4} i = 10$.

a) Write out all the numbers that need to be added:

$\sum_{i=1}^{4} i =$ $1 + 2 + 3 + 4$

b) Show how this sum can be calculated in terms of a smaller summation.

$\sum_{i=1}^{4} i =$ $4 + \sum_{i=1}^{3} i$

12. Write an expression similar to #11b showing how any summation of n integers can be calculated in terms of a smaller summation.

$$\sum_{i=1}^{n} i = \text{n} + \sum_{i=1}^{n-1} i$$

13. What is the base case of the summation? (Write the complete formula, not just the value.)

$$\sum_{i=1}^{1} i = 1$$

Here are important questions to consider before writing a recursive function:

- *How can you define the problem in terms of a smaller similar problem?*
- *What is the base case, where you solve an easy problem in one step?*
- *For the recursive call, how will you make the problem size smaller?*

To avoid infinite recursion, make sure that each recursive call brings you closer to the base case!

14. Implement a recursive function named `summation` that takes a parameter `n` and returns the sum $1 + 2 + \ldots + n$. It should only have an `if` statement and two `return` statements (no loops).

```
def summation(n):
    if n == 1:
        return 1
    else:
        return n + summation(n - 1)
```

15. Enter your code into a Python Editor, and test the function. Make sure that `summation(100)` correctly returns 5050.

16. Implement a recursive function named `geometric` that takes three parameters (`a`, `r`, and `n`) and returns the sum "$a + ar + ar^2 + ar^3 \ldots$" where $n + 1$ is the total number of terms.

a) What is the base case?
`geometric(a, r, 0)` returns: `a`

b) What is the recursive case?
`geometric(a, r, n)` returns: `a * r ** n + geometric(a, r, n - 1)`

c) Write the function in Python:

```
def geometric(a, r, n):
    if n == 0:
        return a
    else:
        return a * r ** n + geometric(a, r, n - 1)
```

17. Enter your code into a Python Editor, and test the function. For example, if $a = 10$ and $r = 3$, the first five terms would be 10, 30, 90, 270, and 810. Make sure that `geometric(10, 3, 4)` correctly returns 1210 (the sum of those five terms).

(demonstrate to instructor)

Appendix

MANAGER

- **Helps the team get started quickly and remain focused.**
  - *"I think we have everything; are we ready to begin?"*
  - *"We're getting off topic; could we talk about that later?"*

- **Takes care of time management; keeps an eye on the clock.**
  - *"I think we need to focus on _____ so we complete this section on time."*
  - *"Let's skip this question for now until we can ask the instructor for help."*
  - *"We have _____ minutes before we need to discuss. Let's get this done."*

- **Makes sure that all voices in the team are heard and respected.**
  - *"(Name), would you be willing to read question _____ out loud?"*
  - *"(Name), what do you think about our team's answer to _____?"*

PRESENTER

- **Communicates questions and clarifications with the instructor and other teams.**
  - *"Our team is confused about how _____ relates to _____."*
  - *"Would you explain what question _____ means by _____?"*

- **Ensures that all team members reach consensus before asking outside sources.**
  - *"Does anyone in our team know the answer for _____?"*
  - *"Before we ask the instructor, could someone clarify _____?"*
  - *"Does everyone agree that we need to find out _____?"*

- **Presents conclusions of the team to the class, when requested by the instructor.**
  - *"How should I explain this idea when asked to report out?"*
    - *"Our team found the answer to number _____ by _____."*

RECORDER

- **Records the important aspects of group discussions, insights, etc.**
  - *"This seems like an important conclusion to write down."*
  - *"Let's stop for a minute so I can get this into our report."*

- **Guides consensus building process; helps team agree on responses.**
  - *"Would you all agree that _____ is a good answer for number _____?"*
  - *"Is our answer completely supported by the explanation we gave?"*
  - *"Would that response make sense to someone from another team?"*

- **Ensures that accurate revisions happen after class discussions.**
  - *"Lets go back and revise what we wrote down for question _____."*
  - *"What did other teams say that we should include in our report?"*

REFLECTOR

- **Observes team dynamics and behavior with respect to the learning process.**
  - *"I think what (name) said earlier is important; would you repeat that?"*
  - *"Let's wait for (name) to finish writing that down before we move on."*

- **Reports to the team periodically during the activity on how the team performs.**
  - *"We're doing really well right now by including all team members."*
  - *"I have a suggestion on how we could be more productive as a team."*
  - *"What process skills are we doing well? What do we need to improve?"*

- **Be ready to report to the entire class about how well the team is operating.**
  - *"Overall, how effective would you say that our team was today?"*
    - *"We found that when _____ happens, it works better if we _____."*

Meta Activity: Team Roles

Decide who will be what role for today; we will rotate the roles each week. If you have only three people, one should have two roles. If you have five people, two may share the same role.

Manager: Helen Hu	keeps track of time, all voices are heard
Presenter: Clif Kussmaul	asks questions, gives the team's answers
Recorder: Chris Mayfield	quality control and consensus building
Reflector: Aman Yadav	team dynamics, suggest improvements

Questions (15 min) **Start time:** _____

1. What is the difference between **bold** and *italics* on the role cards?

The bold points describe what the responsibilities are. Examples of what that person would say are in italics.

2. Manager: invite each person to explain their role to the team. Recorder: take notes of the discussion by writing down key phrases next to the table above.

3. What responsibilities do two or more roles have in common?

Both the presenter and the recorder help the team reach consensus. The manager and reflector both monitor how the team is working.

4. For each role, give an example of how someone observing your team would know that a person is <u>not</u> doing their job well.

- Manager: The team is constantly getting behind.
- Presenter: The student doesn't know what to say.
- Recorder: Some team members aren't taking good notes.
- Reflector: The student never comments on team dynamics.

Meta Activity: Team Disruptions

Common disruptions to learning in teams include: talking about topics that are off-task, teammates answering questions on their own, entire teams working alone, limited or no communication between teammates, arguing or being disrespectful, rushing to complete the activity, not being an active teammate, not coming to a consensus about an answer, writing incomplete answers or explanations, ignoring ideas from one or more teammates.

Questions (10 min)

Start time: _____

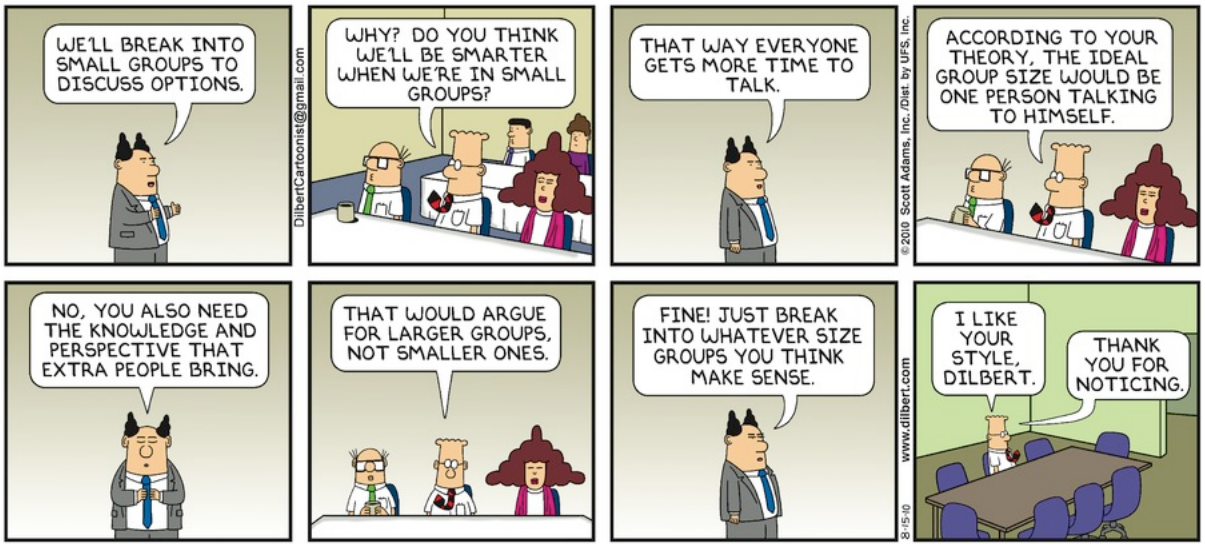
1. Pick four of the disruptions listed above. For each one, find something from the role cards that could help improve the team's success. Use a different role for each disruption.

a) Manager: limited communication between teammates

b) Presenter: ignoring ideas from one or more teammates

c) Recorder: writing incomplete answers or explanations

d) Reflector: teammates answering questions on their own



Dilbert by Scott Adams. © Andrews McMeel Syndication. http://dilbert.com/strip/2010-08-15

Meta Activity: What Employers Want

The following table is from the *Job Outlook 2019* survey by the National Association of Colleges and Employers (NACE). A total of 172 organizations responded to the survey.

Attributes Employers Seek on a Candidate's Resume

	Attribute	% of respondents
1.	Communication skills (written)	82.0%
2.	Problem-solving skills	80.9%
3.	Ability to work in a team	78.7%
4.	Initiative	74.2%
5.	Analytical/quantitative skills	71.9%
6.	Strong work ethic	70.8%
7.	Communication skills (verbal)	67.4%
8.	Leadership	67.4%
9.	Detail-oriented	59.6%
10.	Technical skills	59.6%

Source: https://www.naceweb.org/talent-acquisition/candidate-selection/

Questions (10 min) Start time: _____

1. Consider the top three attributes. Why do you think they are the most sought-after?

Answers will vary. Most employers will look for these skills at career fairs and job interviews. Being able to communicate, solve problems, and work with others is essential.

2. How is communication (written and verbal) related to problem solving and teamwork?

Solving problems in teams involves talking to other people and trying different approaches. Writing solutions down is necessary to solidify the details and share them with others.

3. Which of these skills do each of you individually need to work on? Justify your answers.

Ideally, all of them. Students will definitely learn technical computer programming skills. But working in teams provides the opportunity to develop many other employable skills.

4. How will you develop these skills in college: in courses, or other activities? Be specific.

Answers will vary. Ideally, students will learn these skills both in courses and in other activities. POGIL activities provide an opportunity to develop these skills during class.

Meta Activity: Group vs Team

Throughout the course, you will need to examine and process information, ask and answer questions, construct your own understanding, and develop new problem-solving skills.

Questions (10 min) **Start time:** _____

1. What are some advantages to working in groups/teams?

You get to know other people and make new friends. Different people have different backgrounds and skills. The responsibility is shared.

2. What are some disadvantages to working in groups/teams?

Some group members may decide not to contribute. One or two students may be absent. People may not always get along with each other.

3. What is the difference between a group and a team? Come up with a precise answer.

A group is students who just sit by each other and turn in the same assignment. A team actually works together toward a common goal, drawing on each other's strengths.

4. How can working as a team help you accomplish the tasks described above? Give at least two specific examples.

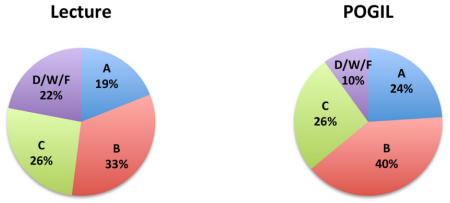
Working as a team makes it easier to examine and process information, because different people have different perspectives. We can also develop new problem-solving skills by observing how each other approaches the problems.

Meta Activity: POGIL Research

Process-Oriented Guided Inquiry Learning (see pogil.org) is a student-centered, group-learning instructional strategy and philosophy developed through research on how students learn best. The following two figures are from peer-reviewed articles published in education journals.

Grade Distributions in General Chemistry

Data (n = 905) from small (~24 students) sections of three instructors using lecture approach (1990-94) prior to implementation of POGIL pedagogy (1994-98).

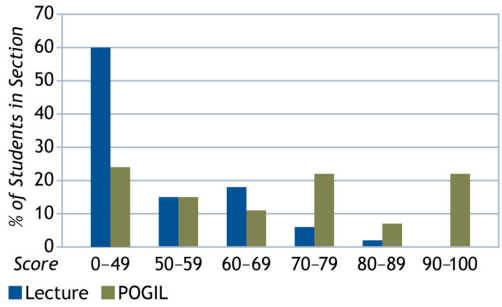


Farrell, J.J., Moog, R.S., & Spencer, J.N. (1999). A Guided Inquiry Chemistry Course. *Journal of Chemical Education*, *76*, 570–574.

Performance on Organic Chemistry 2
Unannounced First Day Pre-Quiz

All students passed Organic Chemistry 1 at this institution during the previous semester

All sections of Organic Chemistry 1 had more than 150 students.



Ruder, S.M., & Hunnicutt, S.S. (2008). POGIL in Chemistry Courses at a Large Urban University: A Case Study. In R.S. Moog, & J.N. Spencer (Eds.), *Process-Oriented Guided Inquiry Learning: ACS Symposium Series 994* (pp. 133–147). Washington, D.C.: American Chemical Society.

Questions (10 min)

Start time: _____

1. How large were the classes at each of the universities shown above?

The left university had classes of about 24, and the right had over 150 students/section.

2. What are the measures of performance shown in each of the figures?

The left figure shows grade distributions, and the right figure shows pre-quiz scores.

3. What does the figure on the left suggest about POGIL's impact on student success?

"Students in courses employing a POGIL instructional strategy achieved a significantly higher success rate (defined as receiving an A, B, or C in the course, as compared to a D, F or withdrawal) than students who had been taught by the same instructors in previous years using a more traditional lecture-oriented approach. In both cases, the students were in classes of about 24 students each, and similar exams were used for both groups of students." (Moog 2014)

4. What does the figure on the right suggest about students' retention of knowledge?

"A majority of the students (60%) in the lecture section scored below 50 percent on the quiz, and none of the students achieved a score above 90 percent. Less than five percent scored above 80 percent. In contrast, fewer than a a quarter of the students from the POGIL section scored below 50 percent on this quiz, and about 30 percent of the students scored above 80 percent, with over one-fifth of the POGIL students scoring above 90 percent." (Moog 2014)

Case Study: Panic Attack

Frank was behind in his programming assignment. He approached Martin to see if he could get some help. But he was so far behind and so confused that Martin just gave him his code with the intent that he would "just look at it to get some ideas."

In the paraphrased words of Frank: "I started the assignment three days after you put it up. But then other assignments came in and I started on them too. I felt like I was chasing rabbits and began to panic. It was already past the due date and I got really scared. That's when I went to Martin to see if he could help." Frank copied much of the code and turned it in as his own.

Questions (10 min) **Start time:** _____

1. Which, if any, of the students were at fault? Why?

Both students are at fault, assuming the assignment was supposed to be done individually. It's not okay to look at someone else's code and/or give your code to someone else.

2. Which specific Honor Code violations occurred?

[Frank] Collaborating in an unauthorized manner with one or more students on any work submitted for academic credit.

[Martin] Rendering unauthorized assistance to another student by knowingly permitting him to see a portion of work to be submitted for academic credit.

3. What should Martin have done in this situation?

This type of situation often occurs when students feel pressure before a deadline. Martin should have refused to show his code to Frank and reminded him to get help from the instructor and/or a TA.

4. What options did Frank have besides cheating?

Frank could have asked the instructor for help during office hours, met with a TA during lab hours, posted a question on Piazza, or asked a question during class. It's much easier to get help when it's not the night before.

Case Study: A Friend Indeed

Jeffrey was having trouble with one of the last programming assignments. He didn't even know where to begin and it was already late. Another student, Stephen, lived in his hall and Jeff was pretty friendly with him. Jeff went to Stephen's room and told him that his computer was acting flaky. He asked if could he borrow Stephen's laptop to finish up the program? Stephen was on his way out to dinner and told him okay. When he got back Jeffrey was gone.

While Stephen was out, Jeffrey searched for and located the code for the assignment on Stephen's machine. Jeffrey copied it onto his flash drive and took it back to his room, where he modified the code a bit before submitting it for a grade.

Questions (10 min) Start time: _____

1. Which, if any, of the students were at fault? Why?

2. Which specific Honor Code violations occurred?

3. What should Stephen have done in this situation?

4. What options did Jeffrey have besides cheating?

Case Study: Oops!

Emily was working in the lab on her programming assignment. She finished the program, submitted it, and went on to do some other work. Shortly thereafter, she left the lab.

Another student, Kyle, was working nearby. He knew that Emily had successfully submitted the assignment, and he had not been able to get his to work properly. When Emily left, he noticed that she had not logged out of her computer. He moved to her workstation, found the work under her Documents directory, and copied it onto his flash drive. He then logged out, logged in as himself, and copied the code onto his Desktop where he modified the program a bit, then successfully submitted it.

Questions (10 min) Start time: _____

1. Which, if any, of the students were at fault? Why?

Kyle is certainly at fault, and depending on the lab policy, Emily may also be penalized. But in general, students are not held accountable when files are stolen without their knowledge.

2. Which specific Honor Code violations occurred?

[Kyle] Committing the act of plagiarism: copying information, ideas, or phrasing of another person without proper acknowledgment of the true source.

[Kyle] Using computing facilities in an academically dishonest manner.

3. What should Emily have done in this situation?

To avoid this situation, Emily should have logged out or at least locked her screen when leaving the room.

4. What options did Kyle have besides cheating?

Kyle could have asked the instructor for help during class or office hours, met with a TA during lab hours, or posted a question on Piazza.

Case Study: Too Close for Comfort

Bill and Jeff were two freshman rooming together in the fall semester and were taking CS 139. Bill was an excellent student for whom the work in CS 139 seemed to come easily. Jeff seemed to struggle a bit more, but was able to do the work and turned most labs and programs in on time.

Both programs came in looking virtually the same. When confronted, Jeff claimed the work as his own and stated that he did not know how the code from the roommates was the same. Later Bill told the professor that while programming was easy for him, he had struggled with another class and then got behind with this program. He didn't think the professor would find out and so stole his roommate's code and turned that in.

Questions (10 min) Start time: _____

1. Which, if any, of the students were at fault? Why?

2. Which specific Honor Code violations occurred?

3. What should Jeff have done in this situation?

4. What options did Bill have besides cheating?

Case Study: Let's Make a Deal

It was mid-semester and the pressure was on, not only in CS but in other classes. Jamie and Pat were each working on the programming assignment in the lab, but neither was having much success. Jamie had started several days ago, but he was having trouble debugging his current work. Pat had just started that day and knew that she would be turning it in late. Regardless, Pat offered to help Jamie work out the problems with his code.

Together, and after a couple of hours of work, they got the program to work. Pat said, "Now that yours is working, can you give me the code so that I can also get credit for this assignment?" When Jamie objected, Pat said, "Hey, you wouldn't have gotten it finished if it weren't for my help, and now mine will be even later!" So Jamie turned over a copy of the code. Pat made some changes to a few sections, and then turned in the final program.

Questions (10 min) Start time: _____

1. Which, if any, of the students were at fault? Why?

2. Which specific Honor Code violations occurred?

3. What should Pat have done in this situation?

4. What should Jamie have done in this situation?

Case Study: A Friendly Assist

George was struggling with the programming assignment on the night it was due. He had gone home over the weekend before, thinking that it would be easy to do this assignment, but it turned out to be more difficult than he thought. After working on some parts of it and giving up in frustration, he turned to Shelley, a senior CS student who had taken the course several semesters before. He showed Shelley the assignment, and the two of them worked on it late into the night. They successfully submitted the program with a late penalty of only one day.

Questions (10 min) Start time: _____

1. Which, if any, of the students were at fault? Why?

2. Which specific Honor Code violations occurred?

3. What should Shelley have done in this situation?

4. What options did George have besides cheating?

Case Study: A Team Effort

Stan, Jane, and Tom were part of a project team. At the start of the semester they all agreed to be responsible for everything that was submitted by their project. As the project got underway, they divided up the work and decided to complete and submit the three parts individually. Tom was behind on his work as the deadline approached, and so as not to be late he copied several pages from files of a roommate who had completed the same work last year. The team project was submitted on time, though only 2/3 of it was original work.

Questions (10 min) **Start time:** _____

1. Which, if any, of the students were at fault? Why?

2. Which specific Honor Code violations occurred?

3. What options did Tom have besides cheating?

4. Should all three students be penalized? Why or why not?