Motivation

• One of the motivations for functions was to reduce redundant code
• But if we need a function in multiple programs, we have been copying it to another program
• Modules allow us to avoid this duplication

Modules

• A file containing functions and/or classes
• Have extension .py
• Another program gains access using:
  ```python
  import moduleName
  ...
  moduleName.moduleFunction()
  ```
  • By default, file defining module needs to be in same directory

Module Details

• Importing a module gives you access, but you still need to say where to find the function you’re interested in:
  ```python
  from moduleName import moduleFunction
  ...
  moduleFunction()
  ```
• Can also add to current namespace:

Common Structure

• Often, a class defines a new type that will be helpful in a number of different programs
• Put the class in its own module!
• It can then be imported into multiple other programs

Script vs Module Motivation

• When designing a class, we want to test its functionality
  • i.e. main code section of our file
• We also want to use the class in other programs
  • Do not want to rerun the tests in order to use it somewhere else
  • `import` “runs” the file
  • So tests are run too!
Avoid running tests

```python
class Something:
    ...
    if __name__ == "__main__":
        # do the tests
```

- When this file is executed (pressing play):
  - Tests are run
- When this file is imported:
  - Tests are skipped

File I/O

- Motivation to use files:
  - Avoid manually entering data for each run
  - Allow programs to write permanent data

- Key ideas:
  - Open a file for reading or writing
  - Read the contents of a file into a program
  - Write data to a file
  - Use the `os` module to create portable code
  - Use `with` to automatically close files

A program is designed to retrieve some data from a file, process it, and output the revised data to another file. Which of the following functions/methods will **not** be called in the program?

A. `open`
B. A loop or method for reading
C. `write`
D. `close`
E. All should be called

Reading Files

- `readline`: reads and returns next line; returns empty string at end-of-file
- `read`: reads the entire file into one string
- `readlines`: reads the entire file into a list of strings

- How do you choose?

89
39
99

```python
scores.txt

```f = open('scores.txt')
lines = f.readlines()
for x in lines:
    print(x, 'boo')
```

- What is the output?
  - 89
  - boo
  - 39
  - boo
  - 99

Reading with Loops

```python
f = open('scores.txt')
for line in f:
    print(line.strip())
```

... or using a while-loop:

```python
f = open('scores.txt')
line = f.readline()
while line:
    print(line.strip())
    line = f.readline()
```
Assuming that f refers to the file at the top and has just been opened, what does skip return?

```
Songs Chosen
#Game names for the songs
#All ridiculously good soundtracks
Sep 17, Chrono Cross
Sep 19, Zelda 3

def skip(f):
    line = f.readline()
    line = f.readline()
    while line.startswith('# '):
        line = f.readline()
    return f.readline()
```

Using the IMDB list of top movies, “TopMoviesIMDB.txt”, find and print the number of titles that have “cat” or “dog” in them.

```
Using the IMDB list of top movies, “TopMoviesIMDB.txt”,
find and print the number of titles that have “cat” or “dog” in them.
```

Multi-Field Records

- Reading entire lines is common, but often the lines contains multiple forms of information
- Critical string method: split
  - No parameters: splits around whitespace
  - With parameter: splits around the given string

```python
class Soldier:
    def __init__(self, str1):
        str1 = str1.split()
        self._name = str1[0]
        self._rank = str1[1]
        self._serial_number = int(str1[2])
        self._nick_names = []
        for i in range(3, len(str1)):
            self._nick_names.append(str1[i])
    def __str__(self):
        return (self._name + self._rank + "\n" + str(self._serial_number))
```

Writing to Files

- Which of the following is valid? Assume outFile is a valid file reference.

```
A. outFile.write(42)
B. outFile.write(‘42’ % 42)
C. outFile.write(The meaning of life is 42)
```

- Write must be given a string argument!