Python:
Modules, Input/Output, and Classes

Hina Arora

Modules

follow along!

(Reference: https://docs.python.org/3.1/tutorial/modules.html)

Scripts and Modules

- If you quit from the Python interpreter and enter it again, the definitions you have made (functions and variables) are lost. Therefore, if you want to write a somewhat longer program, you are better off using a text editor to prepare the input for the interpreter and running it with that file as input instead. This is known as creating a **script**.

- As your program gets longer, you may want to split it into several files for easier maintenance. You may also want to use a handy function that you've written in several programs without copying its definition into each program.

- To support this, Python has a way to put definitions in a file and use them in a script or in an interactive instance of the interpreter. Such a file is called a **module**.

- Definitions from a module can be imported into other modules or into the main module (the collection of variables that you have access to in a script executed at the top level).

- A module is a file containing Python definitions and statements. The file name is the module name with the suffix .py appended.

- When a module is imported, the interpreter will look for it per sys.path.

Create two new files

fibo.py

```
# Fibonacci numbers module

# print Fibonacci series up to n
def fib(n):
    a, b = 0, 1
    while b < n:
        print(b, end=' ')
        a, b = b, a+b
    print()

# return Fibonacci series up to n
def fib2(n):
    result = []
    a, b = 0, 1
    while b < n:
        result.append(b)
        a, b = b, a+b
    return result
```

Lesson14Modules.py

```
# Now you can import the module, and then call the function defined in the module.
import fibo
fibo.fib (10)

# Or you can import the specific function from module, and then call it directly.
from fibo import fib, fib2
fib (20)
lst = fib2 (30)
print (lst)

# Or you can import all functions defined in fibo module and call them.
# This is typically not recommended since you would want to maintain control
# over what is imported into your namespace.
from fibo import *
fib (40)
```

Namespaces

- A namespace is a mapping from names to objects.

- The important thing to know about namespaces is that there is absolutely no relation between names in different namespaces; for instance, two different modules may both define a function maximize without confusion — users of the modules must prefix it with the module name.

- Namespaces are created at different moments and have different lifetimes.
  - The namespace containing the built-in names is created when the Python interpreter starts up, and is never deleted.
  - The global namespace for a module is created when the module definition is read in; normally, module namespaces also last until the interpreter quits.
  - The local namespace for a function is created when the function is called, and deleted when the function returns or raises an exception that is not handled within the function.

Scope

- A scope is a textual region of a Python program where a namespace is directly accessible. “Directly accessible” here means that an unqualified reference to a name attempts to find the name in the namespace.

- At any time during execution, there are at least three nested scopes whose namespaces are directly accessible:
  - The innermost scope, which is searched first, contains the local names
  - The scopes of any enclosing functions, which are searched starting with the nearest enclosing scope, contains non-local, but also non-global names.
  - The next-to-last scope contains the current module’s global names.
  - The outermost scope (searched last) is the namespace containing built-in names.

- If no global or non-local statement is in effect – assignments to names always go into the innermost scope. Assignments do not copy data — they just bind names to objects.

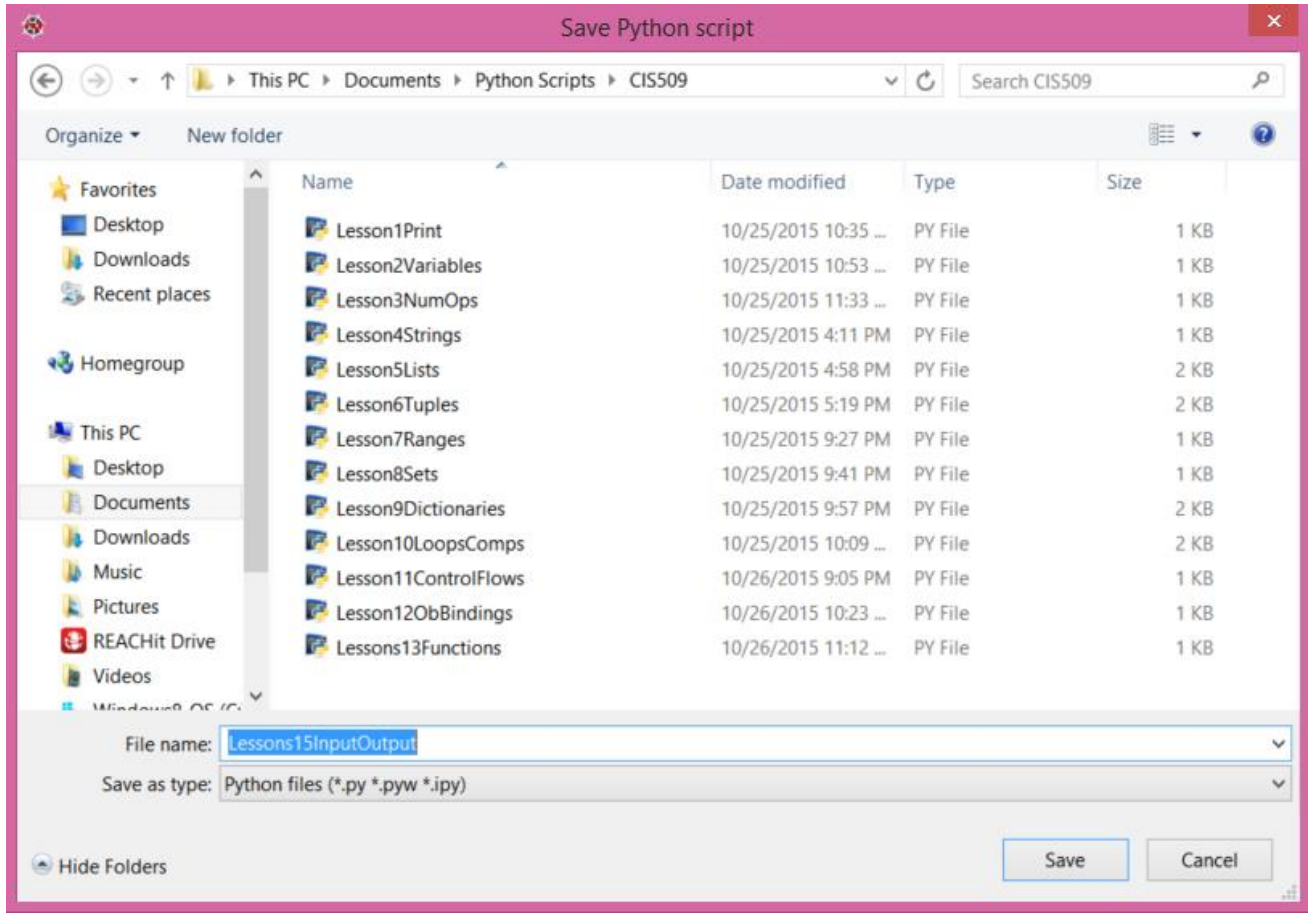
Input/Output

follow along!

(Reference: https://docs.python.org/3.1/tutorial/inputoutput.html)

Create a new file called “Lesson15InputOutput.py”

- Go to File -> New file…
- Go to File -> Save as…
- Got to CISPy directory
- Save file as “Lesson15InputOutput.py”
- Also create a file called “foo.txt” and put a few lines of text in it



```
##### input / output
##### you can read from command line, standard input, or a file
##### you can write to standard output, or a file

```

```
##### input from cmdline

# you can provide cmdline args in one of two ways:
#     - F6 to specify cmdline arguments and then F5 to execute, OR
#     - You can run following on command line: runfile('Lesson15InputOutput.py', args='a b c')

# read cmdline args into argv
from sys import argv
print (argv)

# unpack argv into cmdline args
script, var1, var2, var3 = argv
print (script, var1, var2, var3)

```

```
##### input from stdin

# read input fromm stdin
yourName = input ("what is your name? ")
yourSchool = input ("which school do you go to? ")

# see what you read
print ("your name is:", yourName)
print ("your school is:", yourSchool)

```

```
##### file open and close

# You can open a file in 4 different modes: r (default), w, r+, a.
# This returns a file handle you can use to access the file.
# Always remember to close the file once done and free up any system resources.

# file name
fname = 'foo.txt'

# check if file exists
from os.path import exists
print(exists(fname))

# open file for reading
fhr = open(fname, 'r') # OR fhr = open('file.txt')
fhr.close()

# open file for writing (overwrites)
# if files exists and we open it for write, all contents will immediately be deleted - so always check first!
if (exists(fname)==False):
    fhw = open(fname, 'w')
    fhw.close()

# open file for appending (appends to last line)
fha = open(fname, 'a')
fha.close()

# open file for reading and writing
fhrw = open(fname, 'r+')
fhrw.close()

print ()

```

```
##### file read operations

# open file for reading
fhr = open(fname, 'r')

# print the first line
print (fhr.readline(), end='')
# print another line
print (fhr.readline(), end='')
print ()

# rewind to beginign of the file
fhr.seek(0)
# read entire file at one go
print (fhr.read())
print ()

fhr.seek(0)
# loop through the file and print each line
for line in fhr:
    print(line, end='')
print ()
print ()

fhr.seek(0)
# read entire file into a list
print (list(fhr))
print ()

fhr.seek(0)
# another way to read entire file into a list
print (fhr.readlines())
print ()

# close file
fhr.close()
```

```
110 ##### file write operations
111
112 # file name
113 fname = 'bar.txt'
114
115 # open file for writing
116 fhw = open(fname, 'w')
117
118 # write a few lines
119 fhw.write ("123\n")
120 fhw.write ("456\n")
121 fhw.write ("789\n")
122 fhw.write ("0\n")
123
124 # close file
125 fhw.close()
126
127 # check what was written
128 fhr = open (fname, 'r')
129 print(fhr.read())
130 fhr.close()
```

Will these statements work?

```
134 # test
135
136 # assume foo.txt exists and has content
137 fh = open ('foo.txt', 'r')
138 print (fh.read())
139 print("line: ", fh.readline())
140 fh.close()
141
142 # assume bar.txt exists and has content
143 fh = open ('bar.txt', 'w')
144 fh.close()
145 fh= open ('bar.txt', 'r')
146 print(fh.read())
147 fh.close()
148
149 # assume bar.txt exists and has content
150 fh = open ('bar.txt', 'w')
151 #print(fh.read())
152 fh.close()
153
154 print ()
```

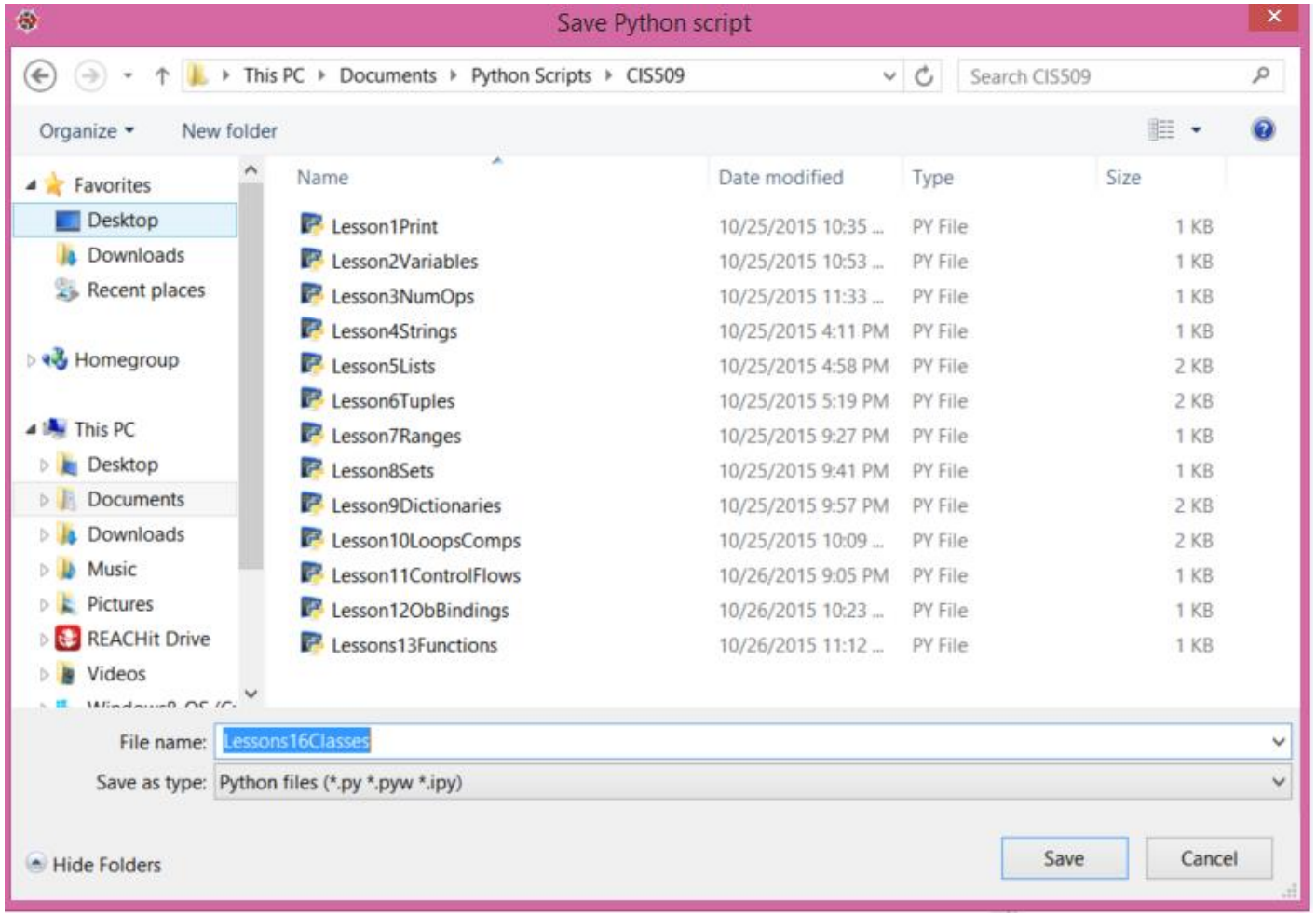
Classes

follow along!

(Reference: https://docs.python.org/3.1/tutorial/classes.html)

Create a new file called “Lesson16Classes.py”

- Go to File -> New file…
- Go to File -> Save as…
- Got to CISPy directory
- Save file as “Lesson16Classes.py”



```
10 # Classes have two attributes - data (class variable) and method
11
12 # Notes:
13 #   - Data attributes may be referenced by methods as well as by ordinary users of an object.
14 #     In other words, classes are not usable in Python to implement pure abstract data types.
15 #     In fact, nothing in Python makes it possible to enforce data hiding – it is all based upon convention.
16 #   - Often, the first argument of a method is called self.
17 #     This is nothing more than a convention: the name self has absolutely no special meaning to Python.
18 #     Note, however, that by not following the convention your code may be less readable to other Python programmers.
```

```
class Dog:                              # class definition

    """This class defines Dogs"""       # class summary

    code = 123                          # class variable - for data shared by all instances
    kind = 'Canine'                     # class variable - for data shared by all instances

    def __init__ (self, name, age=-1): # class instantiation method
        print ("in the instantiation method")
        self.name = name                # instance variable - for data unique to each instance
        self.age = age                  # instance variable - for data unique to each instance
        self.tricks = []                # instance variable - for data unique to each instance

    def bark (self):                    # method
        print ("in the bark method")
        if 'bark' in self.tricks:
            print ('...bow-wow!...')
        else:
            print ('...silence...')

    def add_trick (self, trick):         # method
        print ("in the add_trick method")
        self.tricks.append(trick)

fido = Dog('Fido', 5)                    # instantiating an object of class Dog- with name 'Fido' and age 5
fido.add_trick('roll over')
fido.add_trick('play dead')
print(fido.name, fido.code, fido.kind, fido.age, fido.tricks)
fido.bark()
print ()

buddy = Dog('Buddy', 10)                # instantiating an object of class Dog- with name 'Buddy' and age 10
buddy.add_trick('bark')
print(buddy.name, buddy.code, buddy.kind, buddy.age, buddy.tricks)
buddy.bark()
print ()
```

```python
# Inheritance
#   - Inheritance is used to indicate that one class will get most or all of its features from a parent class.
#   - A class can be drived from a base class within the same module or a base class in a different module
#   - Derived classes may override methods of their base classes.

class Parent (): # base class

    def __init__ (self):
        print ("instantiating base class")

    def Height (self):
        print ("i am tall")

    def HairColor (self):
        print ("i have black hair")

class Child (Parent): # derived class

    def __init__ (self):
        print ("instantiating derived class")

    def HairColor (self): # overriding based class function
        print ("i have blonde hair")

mom = Parent()
mom.Height()
mom.HairColor()

daughter = Child()
daughter.Height()
daughter.HairColor()
print ()
```

Will these statements work?

```
92 class Cat ():
93     pass
94 tiger = Cat()
95
96 class Flower:
97     petals = 0
98     def __init__(self, n):
99         petals = n
100 rose = Flower(10)
101 print (rose.petals)
102
103 class Phone:
104     rings = 0
105     def __init__(self, n):
106         self.rings = n
107 myphone = Phone(5)
108 print (myphone.rings)
```

Errors and Exceptions

follow along!

(Reference: https://docs.python.org/3.1/tutorial/errors.html)

```python
# There are two distinguishable kinds of errors:
#  - Syntax Errors
#  - Excceptions

# example of syntax error (invalid syntax - missing colon):
for i in range(10)
    print(i)

# example of exception (ZeroDivisionError):
def foo ():
    print (1/0)
foo()
```

```python
# You can handle exceptions using "try - except - else - finally" blocks.
#  - First, the try clause (the statement(s) between the try and except keywords)
#    is executed.
#  - If no exception occurs, the except clause is skipped and execution of the
#    try statement is finished.
#  - If an exception occurs during execution of the try clause, the rest of the
#    clause is skipped. Then if its type matches the exception named after the
#    except keyword, the except clause is executed, and then execution continues
#    after the try statement.
#  - If an exception occurs which does not match the exception named in the
#    except clause, it is passed on to outer try statements; if no handler is
#    found, it is an unhandled exception and execution stops with a message.
#  - A finally clause is always executed before leaving the try statement,
#    whether an exception has occurred or not. It is typically used for releasing
#    external resources (such as files or network connections).

# You can also raise built-in exceptions in your program, as well as define
# custom exceptions

# Look here for more information:
# https://docs.python.org/3/tutorial/errors.html
```

Standard Library

follow along!

(Reference: https://docs.python.org/3.1/tutorial/stdlib.html)

Some useful ones you should look at

- datetime: https://docs.python.org/3/library/datetime.html#module-datetime
- regex: https://docs.python.org/3/library/re.html#module-re
- math: https://docs.python.org/3/library/math.html#module-math
- random: https://docs.python.org/3/library/random.html#module-random
- statistics: https://docs.python.org/3/library/statistics.html#module-statistics