DSCI 510
Principles of Programming for Data Science

Jose-Luis Ambite

Research Team Leader, Information Sciences Institute
Associate Research Professor, Department of Computer Science

University of Southern California

BLACK
HISTORY
MONTH
LAKERS

Midterm

- Found a bigger room: GFS 106 (sits 144)
- Same time: March 2nd, 16:00-17:50
- On the midterm you will be primarily asked:
    - ✓ Questions about terminology used in Python.
    - ✓ Questions about, and requests to define classes and functions.
    - ✓ Questions about how different operators and functions work.
- Bring your laptop to the class. The midterm will be a quiz in D2L. We'll use browser lock. I'll also bring some paper copies as backup.
- You will not be allowed any other computing device (e.g., smartphone)
- You **can** prepare and use a single, one-side page with any material you want on it

Topics

Week1	Computers, programming	Ch1
	Variables, Types, Expressions	Ch2
Week2	Conditionals: if, elif, else	Ch3
	Functions, Scope: def f(x,y)	Ch4
Week3	Iteration: while, for loops	Ch5
	Strings	Ch6
Week4	Files, Exceptions	Ch7
	Lists	Ch8
Week5	Dictionaries	Ch9
	Tuples	Ch10
	Sets	
Week6	Object-Oriented Programming	Ch14
Week 7	Object-Oriented Programming	Ch14
	Function params *args, **kargs, scope	
	Regular Expressions	Ch11

Objects in Python

Classes_Objects_with_university_students
_professors_courses.ipynb

Objects in Python

- Object method call:
    - object.method(args) == Class.method(object, args)
- Accessing Objects
    - Composing method calls
- Class membership, Inheritance
    - isinstance(var, Class)
    - issubclass(class1, class2)
- Class vs Instance Variables
- dir(), help()

Object Printing and Display __str__ vs __repr__

`object.__str__(self)`

- "Informal" or nicely printable string representation of an object.
- Return value must be a string object

`object.__repr__(self)`

- "Official" string representation of an object
- If possible, this should look like a valid Python expression that could be used to recreate an object with the same value
- If this is not possible, a string of the form <some useful description> should be returned.
- If a class defines __repr__() but not __str__(), then __repr__() is also used when an "informal" string representation of instances of that class is required.
- This is typically used for debugging, so it is important that the representation is information-rich and unambiguous.

https://docs.python.org/3/reference/datamodel.html?highlight=__str__#object.__repr__
https://docs.python.org/3/reference/datamodel.html?highlight=__str__#object.__str__

object-printing.ipynb

Object Printing

x	str(x)	repr(x)
Goal:	Readable	Unambiguous
Defined by method:	__str__	__repr__
Used by:	print()	*output*
1	'1'	'1'
'1'	'1'	"'1'"
3.14159	'3.14159'	'3.14159'
[1, 2, 3, 4]	'[1, 2, 3, 4]'	'[1, 2, 3, 4]'
datetime.datetime.now()	'2022-02-16 10:30:00.0'	'datetime.datetime(2022, 2, 16, 10, 30, 0, 0)'
Student('Bill', ‘Ho', …)	'<Student object at 0x7fcd68078e50>'	'<Student object at 0x7fcd68078e50>'

Dynamic Object Creation

dynamic_object_creation.ipynb

Function Parameters

function-parameters.ipynb
week7-1_more_function_params.ipynb

Function Parameters and Arguments

- Parameters: variables defined inside parentheses while defining a function
  - `def foo(a, b, c):`
- Arguments are the value passed for these parameters while calling a function
  - `foo(1,2,3)`
- Parameters/Arguments:
  - Positional: `foo(1,2,3)`
  - Keyword: `foo(b=2, a=1, c=3)`
  - Default: `def foo(a, b, c=7)` `foo(1,2)`
  - `*args` : pass any number of positional arguments, collected into a tuple
  - `**kwargs` : pass any number of keyword arguments, collected into a dictionary

Returning multiple values

- A function can return multiple values

  Actually, it returns a tuple
- Example:

```
>>> def poly(x):
...     return x, x**2, x**3

>>> num, square, cube = poly(2)
>>> print(num, square, cube)
2 4 8
```

Variable Scope

variable_scope.ipynb

Revisiting Variable Scope

- Jupyter Notebook: `variable_scope.ipynb`
- Parameter/assignee variables inside functions are **local**.
- This can be overwritten by using the *global* declaration.
- The local scope does not apply to passive use.
- The local scope does not apply to manipulations inside a complex object (e.g., collections)

Modules

Modules

- As your programs get longer, you may want to split them into several parts, each containing the functions related to some specific portion of the program, or providing some specific functionality
- Each such file is a module. Definitions from a module can be imported into other modules or into the main module.
- Example from https://docs.python.org/3/tutorial/modules.html
  - suppose you use Fibonacci series a lot, you can put the relevant functions in a file (fibonacci.py) and the `import` them into your program
  - See modules.ipynb

Built-in Modules

- Python has many modules already defined:
  - https://docs.python.org/3/py-modindex.html
- Examples:

```
>>> import math   # math functions
>>> math.pi
3.141592653589793
>>> math.cos(math.pi)
-1.0
>>> import os  # Operating system functions
>>> os.getcwd()
'/Users/ambite/Documents/USC/classes/DSCI-510/DSCI-510-
Spring-2022-Ambite/Week7'
>>> import random
>>> random.randint(1,10)
3
```

Command-line arguments

Command-line Arguments

When you create a Python script,
you may wish to call it with arguments.

You may also want to control its behavior
depending on whether the script is
the main script or an imported module.

Not Very Suitable for `import`

File temperature.py:

```
def fahrenheit_to_celsius(temp_f):
    return (temp_f - 32) * 5 / 9

while temp_f_s := input("Enter temperature in Fahrenheit: "):
    temp_c = fahrenheit_to_celsius(float(temp_f_s))
    print(f'{temp_f_s}F = {temp_c}C')
```

File hospital.py:

```
from temperature import fahrenheit_to_celsius

temp_f = 98.6
temp_c = fahrenheit_to_celsius(temp_f)
print(f'{temp_f}F = {temp_c}C')
```

- Run code in terminal window

temperature.py

hospital.py

Better with `__name__` check and `sys.argv`

File temperature2.py:

```
import sys

def fahrenheit_to_celsius(temp_f):
    return (temp_f - 32) * 5 / 9

if __name__ == "__main__":
    # sys.argv is the list of arguments; sys.argv[0] is filename
    if len(sys.argv) == 1:
        while temp_f_s := input("Enter temperature in Fahrenheit: "):
            temp_c = fahrenheit_to_celsius(float(temp_f_s))
            print(f'{temp_f_s}F = {temp_c}C')
    else:
        for temp_f_s in sys.argv[1:]:
            temp_c = fahrenheit_to_celsius(float(temp_f_s))
            print(f'{temp_f_s}F = {temp_c}C')
else:
    print(f'... Importing module {__name__} ...')
```

./temperature2.py

./temperature2.py 68 86

./hospital2.py

Scripts, Modules, Packages, Libraries

- Script: Python file intended to be run directly.
  - It should do something
  - Often contain code written outside the scope of any classes or functions
- Module: Related code saved in file with extension .py
  - Can define functions, classes, or variables in a module
  - Intended to be imported into scripts or other modules
- Package: directory with a collection of modules
  - Hierarchical structure of module namespace (like directories)
  - directory must contain file named __init__.py with initialization code for the package (can be empty!)
  - Popular packages: NumPy, pandas
- Library: umbrella term for a reusable chunk of code
  - assumed to be a collection of packages
  - often used interchangeably with “package” since packages can contain subpackages
  - Popular libraries: Matplotlib, PyTorch

run test_space_package.py on command line
https://www.youtube.com/watch?v=f26nAmfJggw