Programming in Python

MMM001 - Data Engineering

https://github.com/chbrandt/MMM001

Carlos H Brandt
c.brandt@jacobs-university.de

Table of Contents

Flow-control

- Code blocks that control -- and usually deviate the otherwise -- linear flow of processing in an algorithm.
- Python (as most other languages) provides:
  - "If-then-else" branching control
  - "For-each-item" loops
  - "Do-while-true" loops

```
if <expression-1 is true>:
        # do something
elif <expression-2 is true>:
        # do another thing
else:
        # do something else
```

```
for var_i in <iterable>:
        # do stuff for each
        # var_i
```

```
while <expression is true>:
        # do stuff until
        # <expression> is false
```

Flow-control: high-level constructions

Some control structures are applied in short, concise expressions to operate over simple data flow.

- Ternary operator

```
# Return "value" is evaluated expression is True, "another-value" otherwise

>>> result = <value> if <expression is true> else <another-value>
```

- List-comprehension

```
# Output a list from each evaluation of "val_i" provided the loop on "iterable"

>>> result_list = [ <action> for val_i in <iterable> ]
```

Flow-control & Exceptions: Python addendum

- In Python, flow/exceptions control can have a *final* statement as shown in the snippets below. Such block will *always* be executed *before* the control block is finally completed -- as the *last* step in the respective control block.

Functions

- A function is a block of *reusable* code that accomplish a specific task
- Functions is Python may have zero, one or more arguments
  - **Arguments may have *default* values**
- Functions is Python may return zero, one or more values

- Functions in Python are *first-class* objects, which means they can be handled like any other object of the language:
  - Be passed as argument to another function
  - Be returned from another function
  - Be assigned to a variable name
  - Be in a data structure, either *value or key*

Functions

- Import function from a package/module:

```
>>> from random import random
>>> random()
```

- Define a function:

```
>>> def foo(*args, **kwargs):
        print('do something')
>>> foo()
```

Functions

```
def foo(num):
    # Load 'random' function
    from random import random
    # Just return the sum
    return num + random()

foo()
```

Functions: lambda

- Lambda functions are small, atomic functions designed to be implemented *dynamically*.
- Lambda functions are originally *anonymous*, but can be binded to a variable

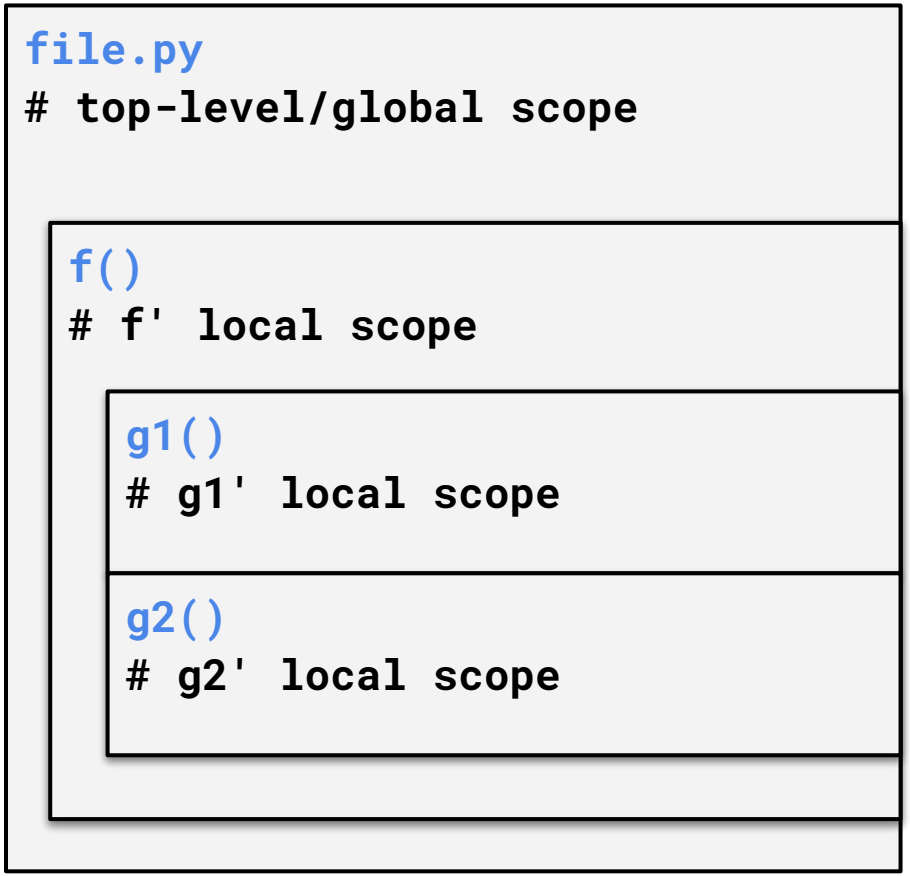
```
>>> i2s = lambda x:return str(x)
>>> i2s(10)
```

```
>>> fs = []
>>> for i in range(5):
…       fs.append(lambda i:return i*2)
>>> [f() for f in fs]
```

Functions: memory scope

- Functions define a block code with its own -- **Local** -- *scope*, meaning that objects/variables created inside a function do not exist outside the function; the function workflow is the *lifetime* for variables therein defined.
- Variables from the outer scopes (*e.g.*, **Global**) are *readable* from the function, (if there is no such variable (name) locally defined).
- **To *modify* a variable** *from an outer scope*, the variable should be explicitly labeled ***global*** or ***nonlocal*** before its use inside the function.

Functions: scope



Functions: scope

```
var = 0
print("top-level:", var)
def foo():
    print("foo: ", var)
    def yay():
        print("yay:", var)
    yay()
foo()
```

Functions: scope

```
var = 0
print("top-level:", var)
def foo():
    var = 1
    print("foo-1: ", var)
    def yay():
        var = 2
        print("yay:", var)
    yay()
    print("foo-2: ", var)
foo()
print("top-bottom:", var)
```

Functions: scope

```
var = 0
print("top-level:", var)
def foo():
    global var
    var = 1
    print("foo-1: ", var)
    def yay():
        global var
        var = 2
        print("yay:", var)
    yay()
    print("foo-2: ", var)
foo()
print("top-bottom:", var)
```

Functions: built-in

In the previous lesson we briefly saw some everyday built-in function, the necessary to move on:

- print (<arguments>)
  - Print arguments to stdout
- input (<message>)
  - Read input from stdin
- type (<object>)
  - Return object type
- id (<object>)
  - Return (unique) object identifier
- range (<[start=0,] stop [,step=1]>)
  - Generate numbers in a range
- len (<sequence>)
  - Return length of sequence
- help (<object>)
  - Online documentation of object

Functions: built-in

...And some more "everyday" (built-in) functions:

- all (<iterable>)
  - Return True if *all* items are True or empty
- any (<iterable>)
  - Return True is *any* items is True
- max (*args)
  - Return the maximum value from *args
- min (*args)
  - Return the minimum value from *args
- filter (<function>, <iterable>)
  - Return iterator result of <function> over "true" inputs. Falses are removed.
- map (<function>, <iterable>)
  - Return iterator from <function> over each item in <iterable>
- zip (*iterables)
  - Return aggregated elements from *iterables

Classes

- *Classes* are the definitions of *objects*; objects are instances of classes.
- Classes define an enclosure -- or *namespace* -- where *members* are defined in a local scope;
  - *Members* may be *data* (variables) and *methods* (functions).
    - Data members are used to keep a *state* of the object
    - Methods allow them to provide specific behaviour
- By using classes we can implement Object-Oriented programming.
  - Objects may interact with each other and specialize through inheritance

Classes

```
class HelloWorld(object):
    """This is my hello-world class"""
    def __init__(self, name=None):
        self.name = name

    def greetings(self):
        """Say hi"""
        print("Hello " + self.name + "!")
```

```
>>> helloworld = HelloWorld('People')
>>> helloworld.greetings()
Hello, People!
```

Classes: data model

Special Method Names:

- .__new__(cls [, args]) & .__del__(self) : constructor & destructor
- .__init__(self [, args]) : initializer, called after '__new__()'
- .__str__(self) : used by the 'print' function to print its returned values
- .__len__(self) : used by the 'len' function to get the object's "length"
- .__bool__(self) : used by 'bool' function to get the object's "boolean" value

The implementation of those methods are *not* mandatory, they are usually *overloaded* to respond properly for daily (*e.g.*, print(obj)) operations.

Strings

We already know what are strings: immutable sequences of characters.

- The *string* class in Python provide a rich set of methods for text processing:
  - .capitalize()
  - .find(<sub>) : search for substring "<sub>" in string and return its position
  - .upper() & .lower() : return the upper-/lower-case version of string
  - .split(<sep>) : split the string in a list at each separator "<sep>"
  - .join(<iterable>) : join "<iterable>" with a separator given by string
  - .replace(old, new) : replace "old" occurrences by "new" in string
  - .strip() : remove left/right whitespace string may have

Strings: format method

The *format* method is used to specify representation for specific types of values, *e.g.*, the precision of floating numbers. The string object calling '.format' is defined with *placeholders* where the arguments given to '.format' are meant to fill.

```
>>> message = "Hello, {}!"
>>> message.format('Mars')
Hello, Mars!
```

- For a simple reference to the different ways '.format' can be used and differences between *old* and *new* formatting schema, have a look at:
  - https://pyformat.info/

Strings: the module

The Standard library provides a module 'string' with predefined character sets that can be quite useful:

- string.ascii_letters
  - abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ
- string.digits
  - 0123456789
- string.punctuation
  - !"#$%&'()*+,-./:;<=>?@[\]^_`{|}~
- string.capwords(<str>)
  - Capitalize each word of "<str>"

References

- W3Schools: Python lambda
- Real Python: how to use lambda functions
- OO programming in Python: variables and scopes
- Datacamp: scope of variable in Python
- Python data model
  - Classes and their Special method names
  - Emulating numeric types
- Python string data type
- Python String Format

Laboratory: strings

- Consider the following string:

```
>>> words = 'This text 10 has some "words",0 shouldn't be here?'
```

- Define a function to count (and return) the number of words in a string.
- Let's do some cleaning in 'words':
  - Replace the numbers/digits by whitespaces
  - Substitute also multiple white spaces by *one* only
  - Remove all non-alpha characters
- How many words in the cleaned string?

Laboratory: functions

- This function is crashing, fix it.
  - If you check the error message, it talks about an 'UnboundLocalError'. There is a good answer to that error in Python documention[1], FAQs

```
var = 0
print("top-level:", var)
def foo():
    print("foo-1: ", var)
    var = 1
    def yay():
        print("yay-1:", var)
        var = 2
        print("yay-2:", var)
    yay()
    print("foo-2: ", var)
foo()
print("top-bottom:", var)
```

[1]: https://docs.python.org/3/faq/programming.html#why-am-i-getting-an-unboundlocalerror-when-the-variable-has-a-value

Laboratory: functions

- In function **yay()**, use **nonlocal** instead of **global**.
- What is the output from the code?
- Can we replace as well the **global** statements from **foo()**?
  - What happens when we do that?

```
var = 0
print("top-level:", var)
def foo():
    global var
    var = 1
    print("foo-1: ", var)
    def yay():
        global var
        var = 2
        print("yay:", var)
    yay()
    print("foo-2: ", var)
foo()
print("top-bottom:", var)
```

Laboratory: functions

This exercises is *non-trivial* and teaches us about the lexical and dynamic scope of Python.

- What is the expected result of the following code?

```
>>> fs = []
>>> for i in range(5):
…       fs.append(lambda i:return i*2)
>>> [f() for f in fs]
```

- Run the code and check if the output is as expected.
- What is the problem of this code? Can you fix it? (see here)

Laboratory: classes

Define a class "Point" that:

- Keeps two data members: "x" and "y", the coordinates of a point
- Support all the basic mathematical operators (+, -, *, /) through the methods:
  - __add__(self, other)
  - __sub__(self, other)
  - __mul__(self, other)
  - __truediv__(self, other)
- Implement the '__str__' method to answer to the (built-in) 'print' function
- Define docstrings for the class *and* the methods and check with 'help' function

Homework: strings & regular expressions

Did you know...

... that the first discovered fossil of the dinosaur Weewarrasaurus was noted for being preserved in green-blue opal?

... that the golden-headed cisticola (pictured) has been described as the "finest tailor of all birds"?

... that the rhythm of the call of the fulvous owl has been likened to Morse code?

Homework: strings & regular expressions

RegEx are *string expressions* coded with special characters -- *e.g.*, *, [,], **?** -- representing a *regular* pattern to be search in a text/document.

- Read the Python documentation about the `**re**` (regular expression) module
  - https://docs.python.org/3/library/re.html
- Write functions that receives a string -- the text in the previous slide -- and ...
  - Substitute -- *replace* -- all occurrences of "**...**" into "**:**".
  - Remove all newline/line-break characters ("**\n**") so the text becomes in a single line.
  - Substitute the first "**...**" into a "**:**", and the others "**...**" (at the beginning of lines) into a "*" bullet; And remove blank lines.
  - Substitute *all* first characters (of *every* word) to upper-case.

Homework: coding

- Implement the following problem and the corresponding tests

Given a *sorted* list of integer numbers, we want to find all integers that are either divisible by 3 and/or 5, but *not* 2. Every time a number divisible by 3 is found the function should print "flip"; when divisible by 5, print "flop".
For testing, a function generating the test data -- the *sorted list of integers* -- has to be implemented. The data generator function accepts one argument for the size of list.
For example: if the list of integers is [1,5,9,10,15,17] the code will output:

'flip'
'flop'
'flipflop'

Homework: coding

- Solve the problem "Question marks" from Coderbyte at:
  - https://www.coderbyte.com/editor/Questions%20Marks:Python

> Have the function QuestionsMarks(str) take the str string parameter, which will contain single digit numbers, letters, and question marks, and check if there are exactly 3 question marks between every pair of two numbers that add up to 10. If so, then your program should return the string true, otherwise it should return the string false. If there aren't any two numbers that add up to 10 in the string, then your program should return false as well.
> For example: if str is "arrb6???4xxbl5???eee5" then your program should return true because there are exactly 3 question marks between 6 and 4, and 3 question marks between 5 and 5 at the end of the string.