Programming Fundamentals

Modules

João Correia Lopes

INESC TEC, FEUP

13 december 2018

GOALS

By the end of this class, the student should be able to:

- Describe the contents of the `random`, `time` and `math` modules.
- Create programmer own modules.
- Describe namespaces, identifier scopes and lookup rules.

BIBLIOGRAPHY

- Peter Wentworth, Jeffrey Elkner, Allen B. Downey, and Chris Meyers, How to Think Like a Computer Scientist — Learning with Python 3, 2018 (Chapter 8) [PDF]
- The Python Tutorial, *6. Modules*, Python 3.6.7 documentation, Release 3.6.7, November 20, 2018 [HTML]

TIPS

- There's no slides: we use a script, illustrations and code in the class. Note that this PDF is NOT a replacement for **studying the bibliography** listed in the *class plan*
- "Students are responsible for anything that transpires during a class—therefore **if you're not in a class**, you should get notes from someone else (not the instructor)"—David Mayer
- The best thing to do is to **read carefully** and **understand** the documentation published in the Content wiki (or else **ask** in the recitation class)
- We will be using **Moodle** as the primary means of communication

CODE, TEST & PLAY

- Have a look at the code in GitHub:
  https://github.com/fpro-admin/lectures/
- Test before you submit at FPROtest:
  http://fpro.fe.up.pt/test/
- Pay a visit to the playground at FPROplay:
  http://fpro.fe.up.pt/play/

CONTENTS

MODULES

- A module is a file containing Python definitions and statements intended for use in other Python programs
- There are many Python modules that come with Python as part of the standard library
- We have seen some already: the `turtle` module, the `string` module, the `functools` module
- The help system contains a listing of all the standard modules that are available with Python
- Play with help!

⇒ https://docs.python.org/3/library/

RANDOM NUMBERS

- We often want to use random numbers in programs
- Python provides a module random that helps with tasks like this
- The `randrange` method call generates an integer between its lower and upper argument, using the same semantics as `range`
- All the values have an equal probability of occurring (it's a *uniform distribution*)

```
1  import random
2
3  rng = random.Random() # create an object that generates random
       numbers
4
5  dice_throw = rng.randrange(1, 7)   # Return one of 1,2,3,4,5,6
6  random_odd = rng.randrange(1, 100, 2)
```

⇒ https://github.com/fpro-admin/lectures/blob/master/22/random.py

REPEATABILITY AND TESTING

- Random number generators are based on a **deterministic** algorithm — repeatable and predictable
- So they're called **pseudo-random** generators — they are not genuinely random
- They start with a *seed* value
- Each time you ask for another random number, you'll get one based on the current seed attribute, and the state of the seed will be updated
- But, for debugging and for writing unit tests, it is convenient to have repeatability

```
1  drng = random.Random(123)  # generator with known starting state
```

PICKING BALLS FROM BAGS, THROWING DICE, SHUFFLING A PACK OF CARDS

- Pulling balls out of a bag with *replacement*

```
1  def make_random_ints(num, lower_bound, upper_bound):
2      rng = random.Random()
3      result = []
4      for i in range(num):
5          result.append(rng.randrange(lower_bound, upper_bound))
6      return result
```

- Pulling balls out of the bag *without replacement*

```
1  xs = list(range(1,13))  # Make list 1..12 (there are no
       duplicates)
2  rng = random.Random()   # Make a random number generator
3  rng.shuffle(xs)         # Shuffle the list
4  result = xs[:5]         # Take the first five elements
```

⇒ https://github.com/fpro-admin/lectures/blob/master/22/randon_ints.py

PICKING BALLS FROM BAGS, THROWING DICE, SHUFFLING A PACK OF CARDS

- Pulling balls out of a bag with *replacement*

```
1  def make_random_ints(num, lower_bound, upper_bound):
2      rng = random.Random()
3      result = []
4      for i in range(num):
5          result.append(rng.randrange(lower_bound, upper_bound))
6      return result
```

- Pulling balls out of the bag *without replacement*

```
1  xs = list(range(1,13))   # Make list 1..12 (there are no
       duplicates)
2  rng = random.Random()    # Make a random number generator
3  rng.shuffle(xs)          # Shuffle the list
4  result = xs[:5]          # Take the first five elements
```

⇒ https://github.com/fpro-admin/lectures/blob/master/22/randon_ints.py

THE TIME MODULE

- The `time` module has a function called `clock` that can be used for *timing* programs
- Whenever `clock` is called, it returns a floating point number representing how many seconds have elapsed since your program started running

⇒ https://github.com/fpro-admin/lectures/blob/master/22/timing.py

THE MATH MODULE

- The `math` module contains the kinds of mathematical functions you'd typically find on your calculator
- Functions: `sin`, `cos`, `sqrt`, `asin`, `log`, `log10`
- Some mathematical constants like `pi` and `e`
- Angles are expressed in radians rather than degrees
- There are two functions `radians` and `degrees` to convert between these two popular ways of measuring angles
- Mathematical functions are "pure" and don't have any *state*

⇒ https://github.com/fpro-admin/lectures/blob/master/22/math.py

CREATING YOUR OWN MODULES

- All we need to do to create our own modules is to save our script as a file with a `.py` extension
- Suppose, for example, this script is saved as a file named `seqtools.py`

```
1  def remove_at(pos, seq):
2      return seq[:pos] + seq[pos+1:]
```

- We can now use our module, both in scripts we write, or in the interactive Python interpreter
- To do so, we must first import the module

```
1  >>> import seqtools
2  >>> s = "A string!"
3  >>> seqtools.remove_at(4, s)
4  'A sting!'
```

__NAME__ (RECAP)

- Before the Python interpreter executes your program, it defines the variable `__name__`
  - The variable is automatically set to the string value `"__main__"` when the program is being executed by itself in a standalone fashion
  - On the other hand, if the program is being imported by another program, then the `"__name__"` variable is set to the name of that module
- This ability to conditionally execute our main function can be extremely useful when we are writing code that will potentially be used by others

⇒ https://github.com/fpro-admin/lectures/blob/master/09/mymath.py

⇒ https://github.com/fpro-admin/lectures/blob/master/09/import.py

NAMESPACES

- A `namespace` is a collection of identifiers that belong to a module, or to a function
- Each module has its own namespace, so we can use the same identifier name in multiple modules without causing an identification problem

```
1  # module1.py
2
3  question = "What is the meaning of Life, the Universe, and
       Everything?"
4  answer = 42
```

```
1  # module2.py
2
3  question = "What is your quest?"
4  answer = "To seek the holy grail."
```

⇒ https://github.com/fpro-admin/lectures/blob/master/22/namespaces.py

FUNCTION NAMESPACES

- Functions also have their own namespaces:

```
1   def f():
2       n = 7
3       print("printing n inside of f:", n)
4
5   def g():
6       n = 42
7       print("printing n inside of g:", n)
8
9   n = 11
10  f()
11  g()
```

Python takes the module name from the file name, and this becomes the name of the namespace: `math.py` is a filename, the module is called `math`, and its namespace is `math`.

⇒ https://github.com/fpro-admin/lectures/blob/master/22/fnamespaces.py

FUNCTION NAMESPACES

- Functions also have their own namespaces:

```
1   def f():
2       n = 7
3       print("printing n inside of f:", n)
4
5   def g():
6       n = 42
7       print("printing n inside of g:", n)
8
9   n = 11
10  f()
11  g()
```

Python takes the module name from the file name, and this becomes the name of the namespace: `math.py` *is a filename, the module is called* `math`*, and its namespace is* `math`*.*

⇒ https://github.com/fpro-admin/lectures/blob/master/22/fnamespaces.py

SCOPE AND LOOKUP RULES

- The **scope** of an identifier is the region of program code in which the identifier can be accessed, or used
- There are three important scopes in Python:
  - **Local scope** refers to identifiers declared within a function: these identifiers are kept in the namespace that belongs to the function, and each function has its own namespace
  - **Global scope** refers to all the identifiers declared within the current module, or file
  - **Built-in scope** refers to all the identifiers built into Python — those like `range` and `min` that can be used without having to import anything, and are (almost) always available
- Functions `locals`, `globals`, and `dir` to see what is the scope
- Python uses precedence rules: the innermost, or local scope, will always take precedence over the global scope, and the global scope always gets used in preference to the built-in scope

⇒ https://github.com/fpro-admin/lectures/blob/master/22/scope.py

THREE IMPORT STATEMENT VARIANTS

- Here are three different ways to import names into the current namespace, and to use them:

```
1   # math is added to the current namespace
2   import math
3   x = math.sqrt(10)
4
5   # names are added directly to the current namespace
6   from math import cos, sin, sqrt
7   x = sqrt(10)
8
9   # import all the identifiers from math
10  from math import *
11  x = sqrt(10)
12
13  #
14  import math as m
15  m.pi
```

EXERCISES

- Moodle activity at: LE22: Modules