Programming Fundamentals

Functions

João Correia Lopes

INESC TEC, FEUP

16 October 2018

Goals

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

- Describe function definition and formal parameters
- Describe function body and local variables
- Describe function call, actual parameters or arguments and the flow of execution
- Describe void functions and fruitful functions that return values

BIBLIOGRAPHY

- Peter Wentworth, Jeffrey Elkner, Allen B. Downey, and Chris Meyers, How to Think Like a Computer Scientist — Learning with Python 3, 2018 (Chapter 4) [PDF]
- Brad Miller and David Ranum, Learning with Python: Interactive Edition. Based on material by Jeffrey Elkner, Allen B. Downey, and Chris Meyers (Chapter 6) [HTML]
- Peter Wentworth, Jeffrey Elkner, Allen B. Downey, and Chris Meyers, How to Think Like a Computer Scientist — Learning with Python 3 (RLE), 2012 (Chapter 4) [HTML]

Tips

- There's no slides: we use a script and some illustrations in the class. That is NOT a replacement for **reading the bibliography** listed in the *class sheet*
- "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 class)
- We will be using **Moodle** as the primary means of communication

Contents

FUNCTIONS

- A *function* is a named sequence of statements that belong together
- Their primary purpose is to help us organize programs into chunks that match how we think about the problem
- The syntax for a function definition is:

```
1  def <NAME>( <PARAMETERS> ):
2      <STATEMENTS>
```

- Function definitions are **compound statements**[1] which follow the pattern:
  1. A **header** line which begins with a keyword and ends with a *colon*
  2. A **body** consisting of *one or more* Python statements, each *indented* the same amount from the header line

[1] as was the case with `for` before

DRAW A SQUARE

- Suppose we're working with turtles, and a common operation we need is to draw squares
- "Draw a square" is an abstraction, or a mental chunk, of a number of smaller steps
- So let's write a function to capture the pattern of this "building block"

⇒ https://github.com/fpro-admin/lectures/blob/master/07/turles.py

DOCSTRINGS FOR DOCUMENTATION

- If the first thing after the function header is a string, it is treated as a **docstring** and gets special treatment
- Docstrings are usually formed using triple-quoted strings
- Docstrings are the key way to document our functions in Python and the documentation part is important
- docstrings are not comments:
  - a string at the start of a function (a docstring) is retrievable by Python tools *at runtime*
  - comments are completely eliminated when the program is parsed

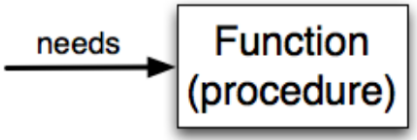
FUNCTION CALL

- Defining the function just tells Python *how* to do a particular task, not to *perform* it
- In order to execute a function we need to make a **function call**
- Function calls contain the name of the function being executed followed by a list of values, called *arguments* (**actual parameters**), which are assigned to the parameters in the function definition (**formal parameters**)
- Once we've defined a function, we can call it as often as we like, and its statements will be executed each time we call it

⇒ https://github.com/fpro-admin/lectures/blob/master/07/moreturles.py

ABSTRACTION

- The following diagram is often called a **black-box diagram** because it only states the requirements from the perspective of the user
- The user must know the name of the function and what arguments need to be passed
- The details of how the function works are hidden inside the “black-box”



A SQUARE IS A (SPECIAL) RECTANGLE

- Let's assume now we want a function to draw a rectangle
- We may use it to draw a square

⇒ https://github.com/fpro-admin/lectures/blob/master/07/rectangle.py

Flow of execution

- Execution always begins at the first statement of the program
- Statements are executed one at a time, in order from top to bottom
- Function definitions do not alter the flow of execution of the program
  - statements inside the function are not executed until the function is called
- Function calls are like a detour in the flow of execution
  - Instead of going to the next statement, the flow jumps to the first line of the called function, executes all the statements there, and then comes back to pick up where it left off.

⇒ https://github.com/fpro-admin/lectures/blob/master/07/pytutor.py

Trace a program

Moral

When we read a program, don't read from top to bottom.
Instead, follow the flow of execution.

Functions that require arguments

- Most functions require arguments: the arguments provide for generalisation

```
1  abs(5)
2  abs(-5)
3
4  pow(2, 3)
5  pow(7, 4)
6
7  max(7, 11)
8  max(4, 1, 17, 2, 12)
9  max(3 * 11, 5 ** 3, 512 - 9, 1024 ** 0)
```

FUNCTIONS THAT RETURN VALUES

- A function that returns a value is called a **fruitful function**
- The opposite of a fruitful function is **void function** — one that is not executed for its resulting value (e.g. `draw_square`)
- Python will automatically return the value `None` for void functions (aka *procedures*)
- Most of the time, calling functions generates a value, which we usually assign to a variable or use as part of an expression

```
1  biggest = max(3, 7, 2, 5)
2  x = abs(3 - 11) + 10
```

INTEREST RATES

The standard formula for compound interest:

$$A = P\left(1 + \frac{r}{n}\right)^{nt}$$

Where:

- P = principal amount (initial investment)
- r = annual nominal interest rate (as a decimal)
- n = the number of times the interest is compounded per year
- t = the number of years that the interest is calculated for

Recall your implementation without functions. Cumbersome, right?

⇒ https://github.com/fpro-admin/lectures/blob/master/07/interests.py

Variables and parameters are local

- When we create a local variable inside a function, it only exists inside the function, and we cannot use it outside
- For example, consider again this function:

```
1  def final_amount(p, r, n, t):
2      a = p * (1 + r/n) ** (n*t)
3      return a
```

- If we try to use `a`, outside the function, we'll get an error
- `a` only exists while the function is being executed — its **lifetime**
- When the execution of the function terminates, the local variables are destroyed
- Parameters are also local, and act like local variables
- Remember, `pythontutor` is your friend!

Variables and parameters are local

- When we create a local variable inside a function, it only exists inside the function, and we cannot use it outside
- For example, consider again this function:

```
1  def final_amount(p, r, n, t):
2      a = p * (1 + r/n) ** (n*t)
3      return a
```

- If we try to use `a`, outside the function, we'll get an error
- `a` only exists while the function is being executed — its **lifetime**
- When the execution of the function terminates, the local variables are destroyed
- Parameters are also local, and act like local variables
- Remember, `pythontutor` is your friend!

Variables and parameters are local

- When we create a local variable inside a function, it only exists inside the function, and we cannot use it outside
- For example, consider again this function:

```
1  def final_amount(p, r, n, t):
2      a = p * (1 + r/n) ** (n*t)
3      return a
```

- If we try to use `a`, outside the function, we'll get an error
- `a` only exists while the function is being executed — its **lifetime**
- When the execution of the function terminates, the local variables are destroyed
- Parameters are also local, and act like local variables
- Remember, `pythontutor` is your friend!

TURTLES REVISITED

- Now that we have fruitful functions, we can focus our attention on reorganizing our code so that it fits more nicely into our mental chunks
- This process of rearrangement is called **refactoring** the code

⇒ https://github.com/fpro-admin/lectures/blob/master/07/refactoring.py

EXERCISES

- Moodle activity at: LE07: Functions