Intro tp programming 4

Henri Vandendriessche
henri.vandendriessche@ens.fr

2023-10-09

Terminal cheat sheet reminder

- Bash commands to navigate directories
  - Print Working Directory. Print the path of the current directory

```
pwd
```

  - List all files of the current directory

```
ls folder
```

  - Moving into folder1 and subfolder2 at once.

```
cd folder1/subfolder2
```

  - Moving out of a directory

```
cd ..
```

  - Going back and forth in the directory tree

```
cd ../../folder1/subfolder1
```

  - Going back to the root directory

```
cd ~
```

- "**Tab**" to use the auto-completion
- **Ctrl + C** to stop a program execution
- "**Upper arrow**" to see last commands
- Many more bash commands to use. . .

So far

- Python
- Data types:
  - integer
  - float
  - string
  - boolean
- **If**, **For** and **While** loops:
  - syntax
  - indentation
- Data collections:
  - list
  - tuple
  - set
  - dictionary
- Python Standard library
  - Python modules
  - Python built-in functions

Today

- Functions:
  - Definition
  - Parameter and argument
  - Return value
  - Scope of variable
  - Module
  - Pure functions vs function vs procedures
- Exercises

Functions intro 1/2

- A function is a named block of instructions.

Functions intro 1/2

- A function is a named block of instructions.
- When you call a function, you execute the code written within that function.

```
#  definition of a function named 'one_two'
def one_two():
  print(1)
  print(2)
  print('...')

one_two() # function calls 1
```

```
## 1
## 2
## ...
```

```
one_two() # function calls 2
```

```
## 1
## 2
## ...
```

Functions intro 2/2

- Calling a function is akin to substituting the function call with its body.

Functions intro 2/2

- Calling a function is akin to substituting the function call with its body.
- If you never call a function, it will never be executed.

Functions intro 2/2

- Calling a function is akin to substituting the function call with its body.
- If you never call a function, it will never be executed.
- Run this code in http://pythontutor.com/

```
#  definition of a function named 'one_two'
def lorem_ipsum():
  print("Lorem ipsum dolor sit amet, consectetur adipiscing elit.")
  print(" Sed non risus.")
        print("Suspendisse lectus tortor, dignissim sit amet, adipiscing nec, ul
  print('...')

lorem_ipsum() # function calls 1
lorem_ipsum() # function calls 2
```

Functions intro 2/2

- Calling a function is akin to substituting the function call with its body.
- If you never call a function, it will never be executed.
- Run this code in http://pythontutor.com/

```
#  definition of a function named 'one_two'
def lorem_ipsum():
  print("Lorem ipsum dolor sit amet, consectetur adipiscing elit.")
  print(" Sed non risus.")
        print("Suspendisse lectus tortor, dignissim sit amet, adipiscing nec, ul
  print('...')

lorem_ipsum() # function calls 1
lorem_ipsum() # function calls 2
```

- What, in your opinion, is the interest of functions?

Usefulness of functions

- Utilizing functions helps prevent code duplication (i.e., copying and pasting), making program modification and correction more efficient (errors are confined to a single place).

Usefulness of functions

- Utilizing functions helps prevent code duplication (i.e., copying and pasting), making program modification and correction more efficient (errors are confined to a single place).
- Using functions generally enhances code readability and may result in shorter code (hopefully. . .).

Definitions first !

- Functions in Python have a specific syntax for their definition: **def name():**

Definitions first !

- Functions in Python have a specific syntax for their definition: **def name():**
  - **def** : This keyword is used to define a function.

Definitions first !

- Functions in Python have a specific syntax for their definition: **def name():**
  - **def** : This keyword is used to define a function.
  - **name + ()**: The name of the function followed by parentheses. This is similar to how built-in functions are called, for example, len(string).

Definitions first !

- Functions in Python have a specific syntax for their definition: **def name():**
  - **def** : This keyword is used to define a function.
  - **name + ()**: The name of the function followed by parentheses. This is similar to how built-in functions are called, for example, len(string).
  - **:** The colon is essential to indicate the start of the function's body.

Definitions first !

- Functions in Python have a specific syntax for their definition: **def name():**
  - **def** : This keyword is used to define a function.
  - **name + ()**: The name of the function followed by parentheses. This is similar to how built-in functions are called, for example, len(string).
  - **:** The colon is essential to indicate the start of the function's body.

- Functions must be defined before they are called

```
one_two() # function calls 1
one_two() # function calls 2

def one_two():
  print(1)
  print(2)
  print('...')
```

NameError: name 'one_two' is not defined

Definitions first !

- Functions in Python have a specific syntax for their definition: **def name():**
  - **def** : This keyword is used to define a function.
  - **name + ()**: The name of the function followed by parentheses. This is similar to how built-in functions are called, for example, len(string).
  - **:** The colon is essential to indicate the start of the function's body.

- Functions must be defined before they are called

```
one_two() # function calls 1
one_two() # function calls 2

def one_two():
  print(1)
  print(2)
  print('...')
```

  NameError: name 'one_two' is not defined

- Remarks:

Definitions first !

- Functions in Python have a specific syntax for their definition: **def name():**
  - **def** : This keyword is used to define a function.
  - **name + ()**: The name of the function followed by parentheses. This is similar to how built-in functions are called, for example, len(string).
  - **:** The colon is essential to indicate the start of the function's body.

- Functions must be defined before they are called

```python
one_two() # function calls 1
one_two() # function calls 2

def one_two():
  print(1)
  print(2)
  print('...')
```

NameError: name 'one_two' is not defined

- Remarks:
  - A given script can contain several function definitions.

Definitions first !

- Functions in Python have a specific syntax for their definition: **def name():**
  - **def** : This keyword is used to define a function.
  - **name + ()**: The name of the function followed by parentheses. This is similar to how built-in functions are called, for example, len(string).
  - **:** The colon is essential to indicate the start of the function's body.

- Functions must be defined before they are called

```
one_two() # function calls 1
one_two() # function calls 2

def one_two():
  print(1)
  print(2)
  print('...')
```

NameError: name 'one_two' is not defined

- Remarks:
  - A given script can contain several function definitions.
  - As a convention, all functions definitions must be at the beginning of the script.

Arguments

- You can provide inputs to your function, which are called parameters.

```
def hello(name):
    print('Hello, ' + name)

hello('Alice')
```

```
## Hello, Alice
```

```
hello('Bob')
```

```
## Hello, Bob
```

Arguments

- You can provide inputs to your function, which are called parameters.

```python
def hello(name):
    print('Hello, ' + name)

hello('Alice')
```

```
## Hello, Alice
```

```python
hello('Bob')
```

```
## Hello, Bob
```

- When you call a function like hello('Alice'), the argument 'Alice' is stored in the variable named inside the function.

Arguments

- You can provide inputs to your function, which are called parameters.

```
def hello(name):
    print('Hello, ' + name)

hello('Alice')
```

```
## Hello, Alice
```

```
hello('Bob')
```

```
## Hello, Bob
```

- When you call a function like hello('Alice'), the argument 'Alice' is stored in the variable named inside the function.

- Run it in http://pythontutor.com/

Arguments

- You can provide inputs to your function, which are called parameters.

```python
def hello(name):
    print('Hello, ' + name)

hello('Alice')
```

```
## Hello, Alice
```

```python
hello('Bob')
```

```
## Hello, Bob
```

- When you call a function like hello('Alice'), the argument 'Alice' is stored in the variable named inside the function.
- Run it in http://pythontutor.com/
- Note: The variable 'name' is created and exists only during the execution of the function hello(); it is local to hello().

Multiple arguments

- If you can pass one argument, you can also pass two or ten, depending on the function's definition.

```
def print_if_divisible(n, div):
  if (n % div == 0):
    print(n, ' is a divisible by ', div)

print_if_divisible(10, 5)
```

```
## 10  is a divisible by  5
```

```
print_if_divisible(11, 5)
```

Multiple arguments

- If you can pass one argument, you can also pass two or ten, depending on the function's definition.

```
def print_if_divisible(n, div):
  if (n % div == 0):
    print(n, ' is a divisible by ', div)

print_if_divisible(10, 5)
```

```
## 10  is a divisible by  5
```

```
print_if_divisible(11, 5)
```

- As an exercise, you can use the provided function to find the divisors of numbers like 10, 15, 27, 33, 64, and 100.

Return values 1/3

- The functions we have seen so far perform actions.

Return values 1/3

- The functions we have seen so far perform actions.
- Functions can also return the result(s) of a computation.

```
def func(x):
  y = 2 * x + 1
  return y

print(func(0.0))
```

```
## 1.0
```

```
print(func(1.0))
```

```
## 3.0
```

```
print(func(2.5))
```

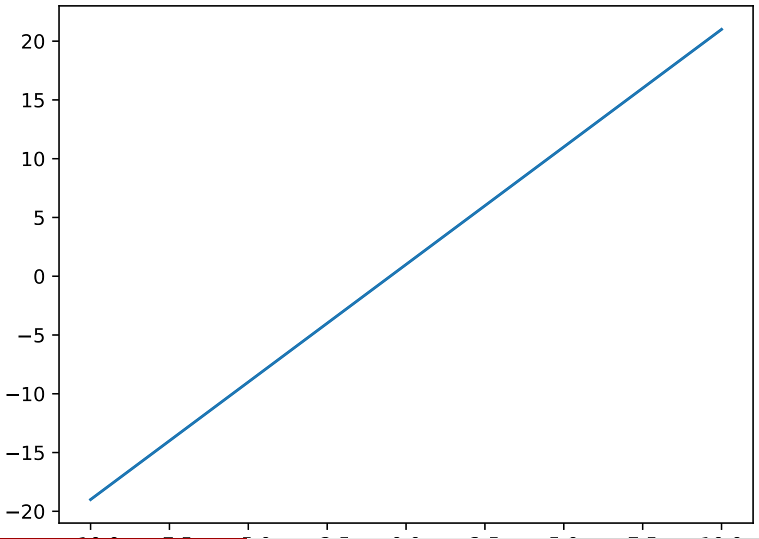
```
## 6.0
```

Return values 2/3

```
def func(x):
  y = 2 * x + 1
  return y
# compute the values of func for x in [-10, _9, -8, ..., 8, 9, 10]
xs = range(-10, 11)
values = []
for x in xs:
  values.append(func(x))

# display them on a graphics
import matplotlib.pyplot as plt
plt.plot(xs, values)
plt.show()
```

Return values 3/3



Boolean Functions

- Boolean functions return either True or False.

```
def is_divisible(x, y):
    if (x % y == 0):
        result = True
    else:
        result = False

    return result

print(is_divisible(10, 5))
```

```
## True
```

-> Question: How can one simplify (shorten) the function is_divisible?

Returning “complex” objects

- A function in Python can return various data types, including tuples, lists, dictionaries, and more.

```
def f(x):
    y1 = x + 1
    y2 = x * 3
    y3 = x ** 2 + 3
    return (y1, y2, y3)

a,b,c =f(2.0)
print(a,b,c)
```

```
## 3.0 6.0 7.0
```

Returning “complex” objects

- A function in Python can return various data types, including tuples, lists, dictionaries, and more.

```
def f(x):
    y1 = x + 1
    y2 = x * 3
    y3 = x ** 2 + 3
    return (y1, y2, y3)

a,b,c =f(2.0)
print(a,b,c)
```

```
## 3.0 6.0 7.0
```

- You can specify default values for function arguments, allowing you to provide a parameter with a default value if none are passed during the function call.

Returning “complex” objects

- A function in Python can return various data types, including tuples, lists, dictionaries, and more.

```python
def f(x):
    y1 = x + 1
    y2 = x * 3
    y3 = x ** 2 + 3
    return (y1, y2, y3)

a,b,c =f(2.0)
print(a,b,c)
```

```
## 3.0 6.0 7.0
```

- You can specify default values for function arguments, allowing you to provide a parameter with a default value if none are passed during the function call.
- It is indeed possible to set default values for function arguments.

```python
def message(name, msg='Hello'):
    print(msg + ' ' + name + '!')

message("Anna")
```

```
## Hello Anna!
```

```python
message("Anna", "Gooodbye")
```

Use of position or keyword

- In a function call, parameters are usually assigned values from arguments based on either their position or their names.

```
def f(a, b):
  print('a=', a)
  print('b=', b)

f(1 ,2)
```

```
## a= 1
## b= 2
```

```
f(2, 1)
```

```
## a= 2
## b= 1
```

```
f(b=2, a=1)    # but one can also use the names of arguments
```

```
## a= 1
## b= 2
```

Methods vs Functions

- Methods are quite similar to functions.
- Each data type has its own set of methods.
- For example the **list** data type has methods such:
  - **append()**
  - **sort()**
  - **index()**
  - **reverse()**
  - and more. . .
- Unlike functions, methods are called on specific values (e.g., lists, dictionaries, sets, etc.).
- Here's an example illustrating the difference between a method and a function.

```
def appending(list1,new_element):
  list1[-1] = new_element
  return list1

element = "g"
myList = ["a", "b", "c", "d", "e" ]
myList.append(element) # method
print(myList)
```

```
## ['a', 'b', 'c', 'd', 'e', 'g']
```

```
print(appending(myList,element)) # function
```

```
## ['a', 'b', 'c', 'd', 'e', 'g']
```

Scope of variables 1/2

- Try the following code in http://pythontutor.com/

```
name = 'chris'

def hello(name):
    print('Hello, ' + name)

print(name)
```

```
## chris
```

```
hello('Alice')
```

```
## Hello, Alice
```

```
hello('Bob')
```

```
## Hello, Bob
```

```
print(name)
```

```
## chris
```

Scope of variables 1/2

- Try the following code in http://pythontutor.com/

```
name = 'chris'

def hello(name):
    print('Hello, ' + name)

print(name)
```

```
## chris
```

```
hello('Alice')
```

```
## Hello, Alice
```

```
hello('Bob')
```

```
## Hello, Bob
```

```
print(name)
```

```
## chris
```

- Two variables with the same name, **Name**, can be used for different purposes. This is possible but can be confusing because they have different scopes.

Scope of variables 2/2

- **local variables** : These are arguments or variables defined within the body of a function. They exist only while the function is being executed and are destroyed when the function finishes, freeing up the associated memory.

Scope of variables 2/2

- **local variables** : These are arguments or variables defined within the body of a function. They exist only while the function is being executed and are destroyed when the function finishes, freeing up the associated memory.
- **non-local variables** : These variables are created in the environment where the variable was called. Functions can access them (unless shadowed by a local variable). However, this practice is generally discouraged and should be avoided except in specific cases.

Scope of variables 2/2

- **local variables** : These are arguments or variables defined within the body of a function. They exist only while the function is being executed and are destroyed when the function finishes, freeing up the associated memory.
- **non-local variables** : These variables are created in the environment where the variable was called. Functions can access them (unless shadowed by a local variable). However, this practice is generally discouraged and should be avoided except in specific cases.
- The reason for avoiding it is to make functions self-contained and easy to understand based solely on their calls.

Scope of variables 2/2

- **local variables** : These are arguments or variables defined within the body of a function. They exist only while the function is being executed and are destroyed when the function finishes, freeing up the associated memory.
- **non-local variables** : These variables are created in the environment where the variable was called. Functions can access them (unless shadowed by a local variable). However, this practice is generally discouraged and should be avoided except in specific cases.
- The reason for avoiding it is to make functions self-contained and easy to understand based solely on their calls.
- You can read more about local and global scope in the section titled "Local and Global Scope" in the book : "Automate the Boring Stuff" Chapter 3 https://automatetheboringstuff.com/chapter3/

Functions can call other functions

Functions can call each other.

```
def func1():
  print(1)

def func2():
  func1()
  print(2)
  func1()

func2()
```

-> Predict the output of this script.

Functions can call other functions

Note that functions can call each other.

```python
def func1():
  print(1)

def func2():
  func1()
  print(2)
  func1()

func2()
```

```
## 1
## 2
## 1
```

Recursive functions

Recursive functions are functions that contain calls to themselves.

For example:

```python
def fact(n):
    if n == 0:
        return 1
    else:
        return n * fact(n - 1)
```

Modules

- You can access functions in modules using the import command.

```
import random
```

Modules

- You can access functions in modules using the import command.

```
import random
```

- You can create your own modules with your own functions, just make sure not to use names already used by the Python standard library.

Modules

- You can access functions in modules using the import command.

```
import random
```

- You can create your own modules with your own functions, just make sure not to use names already used by the Python standard library.

- Functions defined in a file like myfunc.py in the current folder can be called from another Python script.

```
### file "mymodule.py"
def hello(name):
  print("Hello ", name, "!")

### file "myscript.py"
import mymodule

mymodule.hello("Chris")
```

Modules

- You can access functions in modules using the import command.

```
import random
```

- You can create your own modules with your own functions, just make sure not to use names already used by the Python standard library.

- Functions defined in a file like myfunc.py in the current folder can be called from another Python script.

```
### file "mymodule.py"
def hello(name):
  print("Hello ", name, "!")

### file "myscript.py"
import mymodule

mymodule.hello("Chris")
```

- Modules (or libraries) allow you to reuse functions and keep your scripts more organized.

Modules

- You can access functions in modules using the import command.

```
import random
```

- You can create your own modules with your own functions, just make sure not to use names already used by the Python standard library.
- Functions defined in a file like myfunc.py in the current folder can be called from another Python script.

```
### file "mymodule.py"
def hello(name):
  print("Hello ", name, "!")

### file "myscript.py"
import mymodule

mymodule.hello("Chris")
```

- Modules (or libraries) allow you to reuse functions and keep your scripts more organized.
- Python comes with many built-in modules, including random, math, and os.

if name == 'main'

- Many scripts consist of a series of functions followed by the line:

```python
if __name__ == '__main__':
```

- The behavior of the script can vary depending on whether it is the main script or if it is imported by another script.
- The condition if name == 'main': is true only if the script is executed as a Python script (not imported).
- Functions defined before the if name == 'main': block can be reused when the script is imported by other scripts.
- This structure is designed this way to allow a script to be both executable and importable as a module. It is useful when creating libraries or modules that may require configuration tests or settings but should also be used as a module by others who only need the functions.

Functions vs Procedures

- Different types of functions in programming:

  -1 A pure function always returns the same value for the same parameter and has no side effects.

```
‘‘‘python
def addition(a,b):
  return a+b
x=addition(1,2)
print(x)
‘‘‘

‘‘‘
## 3
‘‘‘
```

  -2 A function returns a value and calculates that value based on its input.

```
‘‘‘python
a=1
def addition(b):
  return a+b
x=addition(2)
print(x)
‘‘‘

‘‘‘
```

Exercises:

1- Define a function with two arguments — a string msg and a number nrepetitions — that prints msg, nrepetition times.

2- Read https://en.wikipedia.org/wiki/Fahrenheit and write a function that converts from Fahrenheit to Celsius, and another one that converts from Celsius to Fahrenheit

3- Define a function is_prime(x) which returns True if x is a prime number, else False. Use it to list all prime numbers below 1000.

4- Two taxis companies propose different pricing schemes: Company A charges 4.80€ plus 1.15€ by km traveled. Company B charges 3.20€ plus 1.20€ by km traveled. Write a first function which, given a distance, returns the costs of both companies, and a second function that returns 'company A' and 'company B', the cheapest company for a given distance.

5- Write a function are_anagrams(word1, word2) that tests if two words are anagrams, that is contain the same letters in different orders.

Even more exercises:

See

- https://pcbs.readthedocs.io/en/latest/representing-numbers-images-text.html
- https://pcbs.readthedocs.io/en/latest/building_abstractions_with_functions.html