Introduction to Python program

Olarik Surinta, PhD.

Outline

- Introduction
- Interactive Mode Programming
- Script Mode Programming
- Using Python as a Calculator
- The built-in function
- Lists
- Tuples

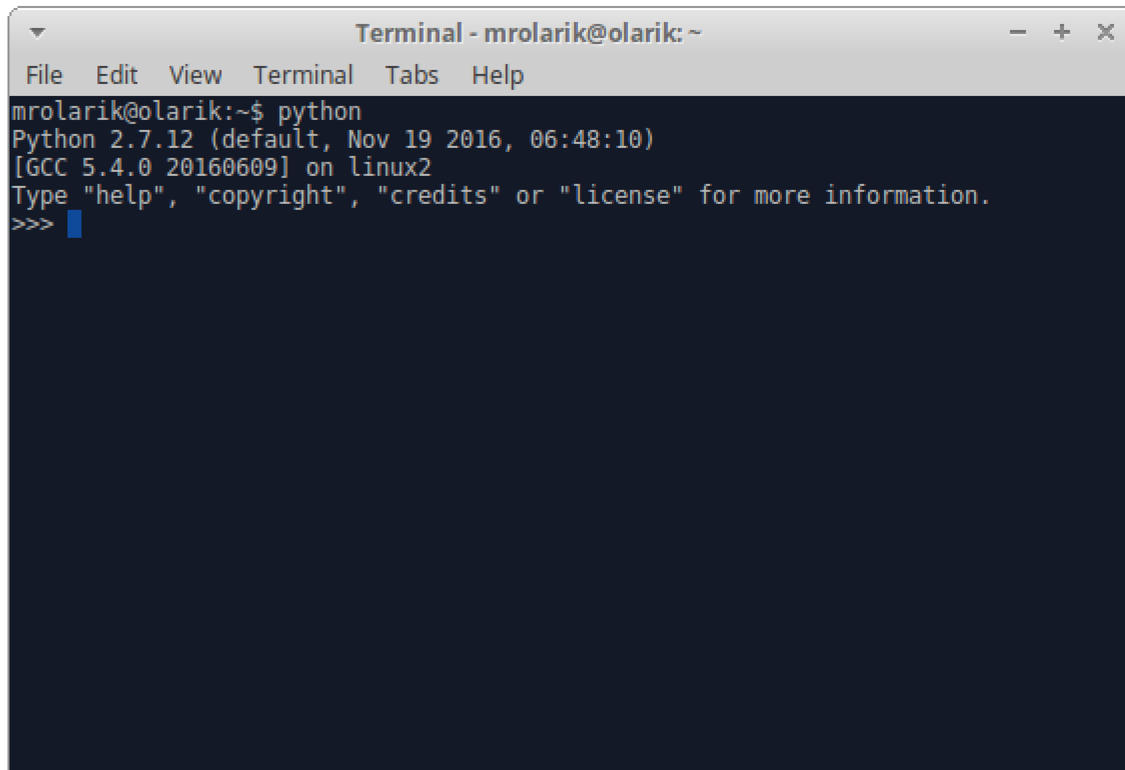
Python

- **Python** is a widely used high-level programming language for general-purpose programming.
- It was created by Guido van Rossum during 1985-1990.

Interactive Mode Programming

```
$ python
Python 2.7.12 (default, Nov 19 2016, 06:48:10)
[GCC 5.4.0 20160609] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Interactive Mode Programming



Interactive Mode Programming

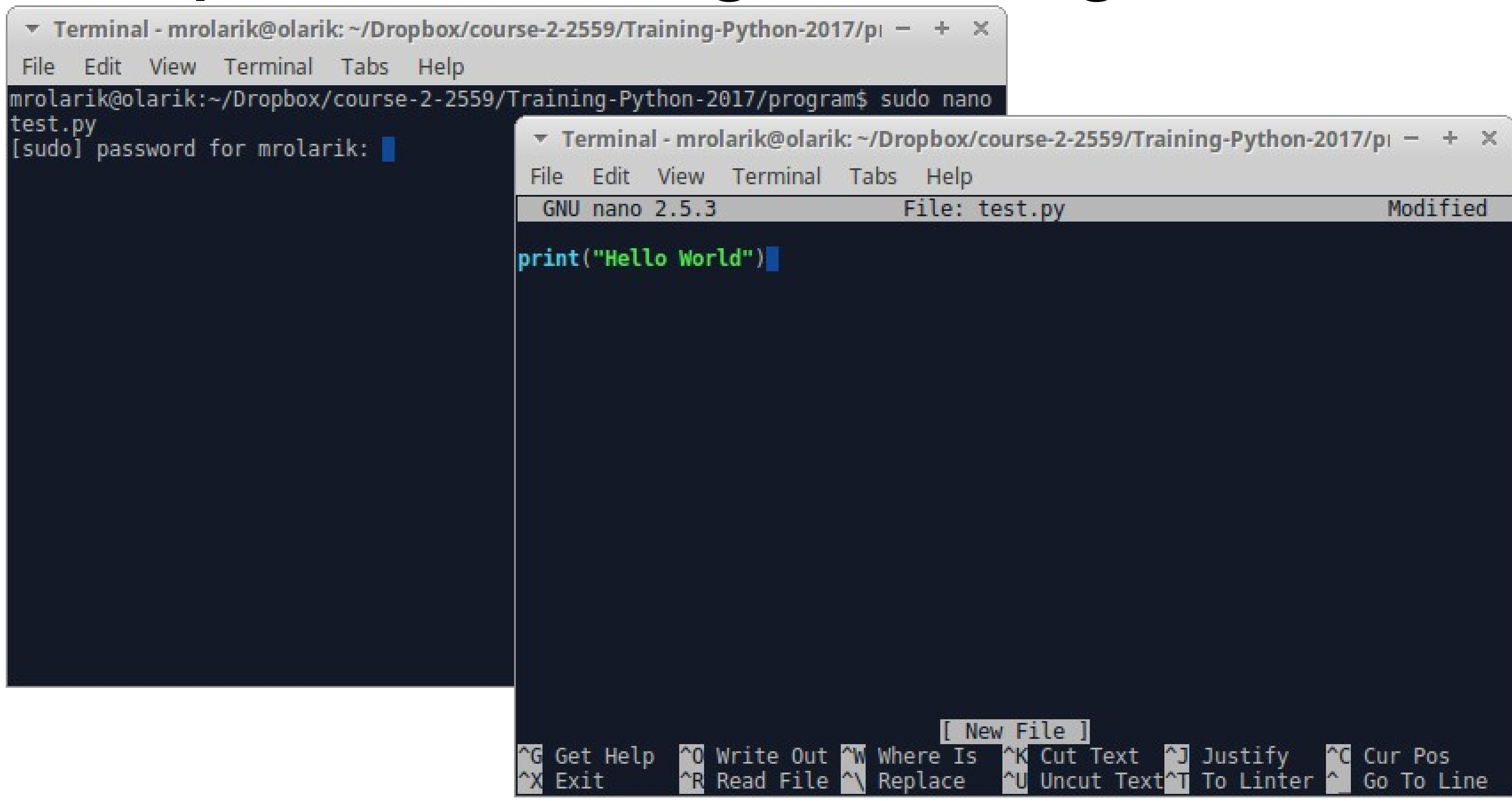
- Use **Ctrl + D** to exit the interactive mode programming.

Script Mode Programming

- Let us write a simple Python program in a script.
- Python files have extension **.py**.

$ sudo nano test.py

Script Mode Programming



Script Mode Programming

- Try to run Python program as follows.

```
$ python test.py
Hello World
```

Script Mode Programming

- Try another way to execute a Python script.
- Put this command at the first line of any Python script.

```
1    #!/usr/bin/python
2
3    print("Hello World")
```

Script Mode Programming

- Change the permission of the Python script.

$ sudo chmod +x test.py

$ ./test.py

Hello World

Script Mode Programming

```
Terminal - mrolarik@olarik: ~/Dropbox/course-2-2559/Training-Python-2017/pı
File   Edit   View   Terminal   Tabs   Help
 GNU nano 2.5.3              File: test.py                     Modified

#!/usr/bin/python

print("Hello World")

                       [ Read 3 lin
^G Get Help   ^O Write Out  ^W Where Is   ^K Cu
^X Exit       ^R Read File  ^\ Replace    ^U Un
```

```
Terminal - mrolarik@olarik: ~/Dropbox/course-2-2559/Training-Python-2017/pı
File   Edit   View   Terminal   Tabs   Help
mrolarik@olarik:~/Dropbox/course-2-2559/Training-Python-2017/program$ sudo chmod
 +x test.py
mrolarik@olarik:~/Dropbox/course-2-2559/Training-Python-2017/program$ ./test.py
Hello World
mrolarik@olarik:~/Dropbox/course-2-2559/Training-Python-2017/program$
```

Lines and Indentation

- Python provides no braces to indicate blocks of code for class and function definitions or flow control.
- Blocks of code are denoted by line indentation, which is rigidly enforced.

Multi-Line Statements

- Statements in Python typically end with a new line.
- Python does, however, allow the use of the line continuation character (\) to denote that the line should continue.

```
>>> total = 1 + \
... 2 + \
... 3
>>> total
6
```

Multi-Line Statements

- Statements contained within the [], {}, or () brackets do not need to use the line continuation character.

```
>>> days = ['Monday', 'Tuesday',
... 'Wednesday', 'Thursday', 'Friday']
>>> days
['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday']
```

Quotation in Python

- Python accepts single ('), double (") and triple (''' or """) quotes to denote string literals, as long as the same type of quote starts and ends the string.
- The triple quotes are used to span the string across multiple lines.

```
word = 'word'

sentence = "This is a sentence."

paragraph = """This is a paragraph. It is

made up of multiple lines and sentences."""
```

Comments in Python

- A hash sign (#) that is not inside a string literal begins a comment.
- All characters after the # and up to the end of the physical line are part of the comment and the Python interpreter ignores them.

```
#!/usr/bin/python

#First comment
print("Hello world")
```

Multi-line Comments in Python

'''

This is a multi-line

comment.

'''

Using Python as a Calculator: Numbers

- The interpreter acts as a simple calculator: you can type an expression at it and it will write the value.
- Expression syntax is straightforward: the operators +, -, *, and /; parentheses ( ) can be used for grouping.

```
>>> 2 + 2
4
>>> 50 - 5*6
20
>>> (50 - 5.0*6) / 4
5.0
```

- The integer numbers (e.g. 2, 4, 20) have type **int**.
- The ones with a fractional part (e.g. 5.0, 1.6) have type **float**.

Using Python as a Calculator: Numbers

```
>>> 17 / 3  # int / int -> int
5
>>> 17 / 3.0  # int / float -> float
5.666666666666667
>>> 17 // 3.0  # explicit floor division discards the fractional part
5.0
>>> 17 % 3  # the % operator returns the remainder of the division
2
>>> 5 * 3 + 2  # result * divisor + remainder
17
```

Using Python as a Calculator: Numbers

- With Python, it is possible to use the ** operator to calculate powers

```
>>> 5 ** 2  # 5 squared
25
>>> 2 ** 7  # 2 to the power of 7
128
```

Using Python as a Calculator: Numbers

- The equal sign (=) is used to assign a value to a variable.

```
>>> width = 20
>>> height = 5 * 9
>>> width * height
900
```

Using Python as a Calculator: Numbers

- If a variable is not “Defined” (assigned a value), trying to use it will give you an error:

```
>>> n  # try to access an undefined variable
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'n' is not defined
```

Using Python as a Calculator: Numbers

- In interactive mode, the last printed expression is assigned to the variable _.
- This means that when you are using Pythin as a desk calculator, it is somewhat easier to continue calculations.

```
>>> tax = 12.5 / 100
>>> price = 100.50
>>> price * tax
12.5625
>>> price + _
113.0625
>>> round(_, 2)
113.06
```

Using Python as a Calculator: Strings

- Python can also manipulate strings, which can be expressed in several ways.

```
>>> 'spam eggs'  # single quotes
'spam eggs'
>>> 'doesn\'t'  # use \' to escape the single quote...
"doesn't"
>>> "doesn't"  # ...or use double quotes instead
"doesn't"
```

```
>>> '"Yes," he said.'
'"Yes," he said.'
>>> "\"Yes,\" he said."
'"Yes," he said.'
>>> '"Isn\'t," she said.'
'"Isn\'t," she said.'
```

Using Python as a Calculator: Strings

- If you don't want characters prefaced by \ to be interpreted as special characters, you can use raw strings by adding an "**r**" before the first quote.

```
>>> print 'C:\some\name'  # here \n means newline!
C:\some
ame
>>> print r'C:\some\name'  # note the r before the quote
C:\some\name
```

Using Python as a Calculator: Strings

- Strings can be concatenated (glued together) with the **+** operator, and repeated with *

```
>>> # 3 times 'un', followed by 'ium'
>>> 3 * 'un' + 'ium'
'unununium'
```

```
>>> 't' + 2 * 'o'
'too'
```

Using Python as a Calculator: Strings

- Strings can be indexed (subscripted), with the first character having index 0.

```
>>> word = 'Python'
>>> word[0]  # character in position 0
'P'
>>> word[5]  # character in position 5
'n'
```

Using Python as a Calculator: Strings

- Indices may also be negative numbers, to start counting from the right

```
>>> word[-1]  # last character
'n'
>>> word[-2]  # second-last character
'o'
>>> word[-6]
'P'
```

Using Python as a Calculator: Strings

- In addition to indexing, slicing is also supported.

```
>>> word[0:2]  # characters from position 0 (included) to 2 (excluded)
'Py'
>>> word[2:5]  # characters from position 2 (included) to 5 (excluded)
'tho'
```

Using Python as a Calculator: Strings

>>> word[:2] + word[2:]

'Python'

>>> word[:4] + word[4:]

'Python'

Using Python as a Calculator: Strings

>>> word[:2] # character from the beginning to position 2 (excluded)

'Py'

>>> word[4:] # characters from position 4 (included) to the end

'on'

>>> word[-2:] # characters from the second-last (included) to the end

'on'

The built-in function

- **len()** returns the length of a string

```
>>> s = 'supercalifragilisticexpialidocious'
>>> len(s)
34
```

Lists

- Python knows a number of compound data types, used to group together other values.
- The most versatile is the *list*, which can be written as a list of comma-separated values between square brackets.

>>> squares = [1, 4, 9, 16, 25]

>>> squares

[1, 4, 9, 16, 25]

Lists

- Lists can be indexed and sliced.

>>> squares[0] # indexing returns the item

1

>>> squares[-1]

25

>>> squares[-3:] # slicing returns a new list

[9, 16, 25]

Lists

- Lists also supports operations like concatenation

```
>>> squares + [36, 49, 64, 81, 100]
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

Lists

```
>>> list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]
>>> type(list)
<type 'list'>
>>> list[0]
'abcd'
>>> type(list[0])
<type 'str'>
>>> type(list[1])
<type 'int'>
```

Lists

- It is possible to change their content

```
>>> cubes = [1, 8, 27, 65, 125]  # something's wrong here
>>> 4 ** 3  # the cube of 4 is 64, not 65!
64
>>> cubes[3] = 64  # replace the wrong value
>>> cubes
[1, 8, 27, 64, 125]
```

Lists

- You can add new items at the end of the list, by using the **append()** method.

```
>>> cubes.append(216)  # add the cube of 6
>>> cubes.append(7 ** 3)  # and the cube of 7
>>> cubes
[1, 8, 27, 64, 125, 216, 343]
```

Lists

```
>>> letters = ['a', 'b', 'c', 'd', 'e', 'f', 'g']
>>> letters
['a', 'b', 'c', 'd', 'e', 'f', 'g']
>>> # replace some values
>>> letters[2:5] = ['C', 'D', 'E']
>>> letters
['a', 'b', 'C', 'D', 'E', 'f', 'g']
>>> # now remove them
>>> letters[2:5] = []
>>> letters
['a', 'b', 'f', 'g']
>>> # clear the list by replacing all the elements with an empty list
>>> letters[:] = []
>>> letters
[]
```

Lists

- It is possible to nest lists (create lists containing other lists).

```
>>> a = ['a', 'b', 'c']
>>> n = [1, 2, 3]
>>> x = [a, n]
```

```
>>> x
[['a', 'b', 'c'], [1, 2, 3]]
>>> x[0]
['a', 'b', 'c']
>>> x[0][1]
'b'
```

Lists

```
>>> list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]
>>> type(list)
<type 'list'>
>>> list[0]
'abcd'
>>> type(list[0])
<type 'str'>
>>> type(list[1])
<type 'int'>
```

Tuples

- A tuple is another sequence data type that is similar to the list.
- A tuple consists of a number of values separated by commas.
- The main differences between lists and tuples are: Lists are enclosed in brackets [ ] and their elements and size can be changed, while tuples are encludes in parentheses ( ) and cannot be updated.

Tuples

```
>>> tuple = ( 'abcd', 786 , 2.23, 'john', 70.2  )
>>> type(tuple[0])
<type 'str'>
>>> type(tuple[1])
<type 'int'>
```

Python Dictionary

- Python's dictionaries are kind of hash table type.
- A dictionary key can be almost any Python type, but are usually numbers or strings.
- Dictionaries are enclosed by curly braces { } and values can be assigned and accessed using square braces [ ]

Python Dictionary

```
dict = {}
dict['one'] = "This is one"
dict[2]     = "This is two"

tinydict = {'name': 'john','code':6734, 'dept': 'sales'}

print dict['one']       # Prints value for 'one' key
print dict[2]           # Prints value for 2 key
print tinydict          # Prints complete dictionary
print tinydict.keys()   # Prints all the keys
print tinydict.values() # Prints all the values
```

References

- https://www.tutorialspoint.com/python/python_basic_syntax.htm
- https://docs.python.org/2/tutorial/introduction.html#using-python-as-a-calculator
-