Preparatory Course Informatics for Life Scientists

An Introduction to Python 1:

Basics

Philipp Thiel

September 13, 2022

Python

- **High-level** programming language:
  It abstracts (hides) details, such as memory, hard drives, CPUs
- **General purpose** programming language:
  Can be used for many things, such as implementing large applications, statistics, machine learning, simple scripting ...
- **Interpreted** programming language:
  Source code is not directly executed on a computer but by an intermediate application: the Python Interpreter
- **Easy to learn** programming language:
  Syntax is not cluttered and source code readability a major goal

Python

- It is a very extensible programming language
- A large developer community contributes free extensions
- These so-called **packages** are easy to install

  The term **module** is also frequently used to refer to packages
- Packages add functionality to your Python installation

  A bare Python installation is like a new mobile phone and packages the apps
- Installing packages modifies a Python installation
- Programmers often need multiple Python installations with different extensions

  One for statistics, one for web development, one for bioinformatics, ...

Python

- We only talk about and work with **Python 3**
- Often already installed on Linux
- Usually not installed on Windows and macOS
- There are different ways to get a Python installation
  - Official Python installer https://www.python.org/downloads
  - **Conda** (Miniconda or Anaconda)
  - OS specific package management tools

Miniconda (Conda)

- Package and **environment** management for Python (and more)
- Includes a stand-alone Python installation
- Environments enable multiple Python installations side-by-side
- The default environment is called **base** and is usually not touched
- Conda provides tools to create, modify, and remove environments
  This also includes tools to install packages!
- Modifications of an environment do not affect other environments
  Thus, conda allows to manage separate Python installations

How to Execute Python Code

- Shell v1: interactive mode of the Python interpreter
- Shell v2: execute python scripts by the Python interpreter
- From an IDE: write and execute your code from program
- Jupyter Notebooks (we probably come to that)

Integrated Development Environment (IDE) for Python

- Software with GUI that facilitates the development of software
- Ready to use building blocks (very good for beginners)
- Management of larger projects
- Highlights syntax (we will see many examples)
- Code completion (we also will see examples here)
- Debugging

Practice Time ... 001

1. Download and install Miniconda with Python 3.9
   → https://docs.conda.io/en/latest/miniconda.html
2. Getting started with conda
   → https://docs.conda.io/projects/conda/en/latest/user-guide/getting-started.html
   → Create a new conda environment called 'prepcourse'
   → Activate the newly created environment
   → Install the package 'jupyter'
3. Start the Python interpreter in interactive mode
   → https://realpython.com/run-python-scripts/#how-to-run-python-code-interactively
   → Use the 'print()' function to write a welcome message
4. Write a Python script called 'myscript.py'
   → Type in (implement) the previous 'print()' statement
   → Execute the Python script from the shell (command line)
5. Download and install PyCharm
   → https://www.jetbrains.com/de-de/pycharm/
   → Start PyCharm and create a new project that uses your new conda environment
   → Re-implement your 'print()' statement and execute it from PyCharm

Basics

myscript.py

```
1
```

```
$ python myscript.py
```

Basics

myscript.py

```
1
```

```
$ python myscript.py
$
```

Lesson: No code is valid Python code

Basics

myscript.py

```
# This is a single comment line
```

```
$ python myscript.py
```

Basics

myscript.py

```
1  # This is a single comment line
```

```
1  $ python myscript.py
2  $
```

Lesson: A comment line starts with the '#' symbol

Basics

myscript.py

```
'''
Multiple lines of comments
are a good habit ...
... because commenting is good habit!
'''
```

```
$ python myscript.py
```

Basics

myscript.py

```
'''
Multiple lines of comments
are a good habit ...
... because commenting is good habit!
'''
```

```
$ python myscript.py
$
```

Lesson: Triple single quotes can enclose multiple lines of comments

Basics

myscript.py

```
# This is a single comment line

print("We already met the print() function!")
```

```
$ python myscript.py
```

Basics

myscript.py

```
1  # This is a single comment line
2
3  print("We already met the print() function!")
```

```
1  $ python myscript.py
2  We already met the print() function!
3  $
```

Lessons:

- The function print() produces some visible output
- It writes something to an output destination: here text (**string**) to **stdout**
- Empty source code lines are ignored
- Python comes with a set of ready to use functions: **built-in functions**
- Python version 3.10 has 81 built-in functions (has been growing)
- We will use and discuss some of them

Basics

myscript.py

```
print("We already met the print() function!")
 print("We already met the print() function!")
```

```
$ python myscript.py
```

Basics

myscript.py

```
print("We already met the print() function!")
 print("We already met the print() function!")
```

```
$ python myscript.py
  File "myscript.py", line 2
    print("We already met the print() function!")
    ^
IndentationError: unexpected indent
$
```

Basics

myscript.py

```
print("We already met the print() function!")
print("We already met the print() function!")
```

```
$ python myscript.py
```

Basics

myscript.py

```
print("We already met the print() function!")
print("We already met the print() function!")
```

```
$ python myscript.py
We already met the print() function!
We already met the print() function!
$
```

Lessons:

- A **block** is a set of lines (statements) that belong together
- Statements in a block must have the same **indentation**
- Blocks can be nested, which will be introduced later
- The outermost block must not be indented
- The error in this case is a so-called **syntax error**

Basics: Strings

- Strings (**str**) are sequences of symbols (characters): **it's text!**
- Explicit strings in source code are called a **string literals**
- String literals are enclosed in **single** or **double quotes**

myscript.py

```
1  print("I am a string literal :)")
2  print('I am a string literal, too!')
```

```
1  $ python myscript.py
```

Basics: Strings

- Strings (**str**) are sequences of symbols (characters): **it's text!**
- Explicit strings in source code are called a **string literals**
- String literals are enclosed in **single** or **double quotes**

myscript.py

```
1  print("I am a string literal :)")
2  print('I am a string literal, too!')
```

```
1  $ python myscript.py
2  I am a string literal :)
3  I am a string literal, too!
4  $
```

Lesson: the print() function enforces a line break

Basics: Strings

myscript.py

```
1  print("This symbol (") is a double quote.")
```

```
1  $ python myscript.py
```

Basics: Strings

myscript.py

```
1  print("This symbol (") is a double quote.")
```

```
1  $ python myscript.py
2    File "myscript.py", line 1
3      print("This symbol (") is a double quote.")
4                                ^
5  SyntaxError: invalid syntax
6  $
```

- Again, Python raised a syntax error … but why?
- Explicit quotes need to be different from the enclosing ones
- Another option: use a **backslash** to ’escape’ the quote character

Basics: Strings

myscript.py

```
1  print('This symbol (") is a double quote.')
2  print("This symbol (\") is a double quote.")
```

```
1  $ python myscript.py
```

Basics: Strings

myscript.py

```
print('This symbol (") is a double quote.')
print("This symbol (\") is a double quote.")
```

```
$ python myscript.py
This symbol (") is a double quote.
This symbol (") is a double quote.
$
```

Basics: Strings

myscript.py

```
1  print('This symbol (") is a double quote.')
2  print("This symbol (\") is a double quote.")
```

```
1  $ python myscript.py
2  This symbol (") is a double quote.
3  This symbol (") is a double quote.
4  $
```

myscript.py

```
1  print("")
```

```
1  $ python myscript.py
```

Basics: Strings

myscript.py

```
1  print('This symbol (") is a double quote.')
2  print("This symbol (\") is a double quote.")
```

```
1  $ python myscript.py
2  This symbol (") is a double quote.
3  This symbol (") is a double quote.
4  $
```

myscript.py

```
1  print("")
```

```
1  $ python myscript.py
2
3  $
```

Lesson: a string can be empty!

Basics: Reading Input

myscript.py

```
1  print( input() )
```

Basics: Reading Input

myscript.py

```
1  print( input() )
```

```
1  $ python myscript.py
2  I call it the parrot program : )
3  I call it the parrot program : )
4  $
```

Basics: Reading Input

myscript.py

```
1  print( input() )
```

```
1  $ python myscript.py
2  I call it the parrot program : )
3  I call it the parrot program : )
4  $
```

Lessons:

- The function **input()** belongs to the built-in functions
- It reads character sequences (strings) from the command line
- It waits until <enter> is pressed
- The function input() it obviously **returns** the input as a string
- And wow: we can stick functions into functions

Basics: Variables

- To do meaningful things we need to store content
- In programming we can store content in so-called **variables**

Basics: Variables

- To do meaningful things we need to store content
- In programming we can store content in so-called **variables**

myscript.py

```
1  # We assign the variable 'input_text'
2
3  input_text = input()
4  print(input_text)
```

Basics: Variables

- To do meaningful things we need to store content
- In programming we can store content in so-called **variables**

myscript.py

```
# We assign the variable 'input_text'

input_text = input()
print(input_text)
```

```
$ python myscript.py
A parrot program with memory ; )
A parrot program with memory ; )
$
```

Basics: Variables

- To do meaningful things we need to store content
- In programming we can store content in so-called **variables**

myscript.py

```
1  # We assign the variable 'input_text'
2
3  input_text = input()
4  print(input_text)
```

- A variable is a **unique name** that refers to some content
- The '=' symbol is used to **assign** content to a variable
- The content is in fact stored in a piece of **memory**
- The variable stores the address to that piece of memory
- Variables need to be assigned immediately

Basics: Variables

myscript.py

```
1  var_1 = "first"
2  var_2 = "second"
3  print(var_1)
4  print(var_2)
```

Basics: Variables

myscript.py

```
1  var_1 = "first"
2  var_2 = "second"
3  print(var_1)
4  print(var_2)
```

```
1  $ python myscript.py
```

Basics: Variables

myscript.py

```
1  var_1 = "first"
2  var_2 = "second"
3  print(var_1)
4  print(var_2)
```

```
1  $ python myscript.py
2  first
3  second
4  $
```

Lesson: String literals can be assigned to a variable :)

Basics: Variables

- Traditionally, variable names in Python ...
  - can have arbitrary length
  - can contain uppercase and lowercase letters (A-Z, a-z)
  - can contain digits (0-9) except the first character
  - can contain the underscore character '_'
- Developers usually follow some **style guide** (coding convention)
- Guidelines on how to program in a specific language
- Strictly following a style guide is a feature of software quality
- Some examples:
  - how to construct variable names
  - how to format comments
  - which quotes for string literals
  - ...

Basics: Variables

- Traditionally, variable names in Python ...
  - can have arbitrary length
  - can contain uppercase and lowercase letters (A-Z, a-z)
  - can contain digits (0-9) except the first character
  - can contain the underscore character '_'

myscript.py

```
1  # Camel case variable naming
2  theFirstVariable = "first"
3
4  # Pascal case variable naming
5  TheFirstVariable = "first"
6
7  # Snake case variable naming
8  the_first_variable = "first"
```

Basics: Variables

myscript.py

```
1  firstVar = "first"
2  print(firstVar)
3  print(firstvar)
```

```
1  $ python myscript.py
```

Basics: Variables

myscript.py

```
1  firstVar = "first"
2  print(firstVar)
3  print(firstvar)
```

```
1  $ python myscript.py
2  first
3  Traceback (most recent call last):
4    File "myscript.py", line 2, in <module>
5      print(firstvar)
6  NameError: name 'firstvar' is not defined
7  $
```

Basics: Variables

myscript.py

```
1 firstVar = "first"
2 print(firstVar)
3 print(firstvar)
```

Lessons:

- Python is a **case sensitive** language: 'var' and 'Var' are different names
- Names (here variables) need to be defined (assigned) before they can be used
- Variables in Python must be assigned: no define without assign :)
- Python error messages are very informative

Basics: Variables

- Every programming language has a set of **keywords**
- Keywords have a special function in the programming language
- Keywords are **reserved** and cannot be used as (variable) name
- Python reserves 33 keywords

False	None	True	and	as	assert	break	class	continue
def	del	elif	else	except	finally	for	from	global
if	import	in	is	lambda	nonlocal	note	or	pass
raise	return	try	why	with	yield			

- We will learn and use some but not all of these keywords

Basics: Variables

myscript.py

```
1  var_1 = "first"
2  print(var_1)
3  var_1 = "second"
4  print(var_1)
```

Basics: Variables

myscript.py

```
1  var_1 = "first"
2  print(var_1)
3  var_1 = "second"
4  print(var_1)
```

```
1  $ python myscript.py
```

Basics: Variables

myscript.py

```
1  var_1 = "first"
2  print(var_1)
3  var_1 = "second"
4  print(var_1)
```

```
1  $ python myscript.py
2  first
3  second
4  $
```

Lesson: variables can be re-assigned with a new content

Basics: Variables

myscript.py

```
1  a = "first"
2  b = a
3  print(a)
4  print(b)
5
6  b = "second"
7  print(a)
8  print(b)
```

Basics: Variables

myscript.py

```
1  a = "first"
2  b = a
3  print(a)
4  print(b)
5
6  b = "second"
7  print(a)
8  print(b)
```

```
1  $ python myscript.py
```

Basics: Variables

myscript.py

```
1  a = "first"
2  b = a
3  print(a)
4  print(b)
5
6  b = "second"
7  print(a)
8  print(b)
```

```
1  $ python myscript.py
2  first
3  first
4  first
5  second
6  $
```

Lessons:

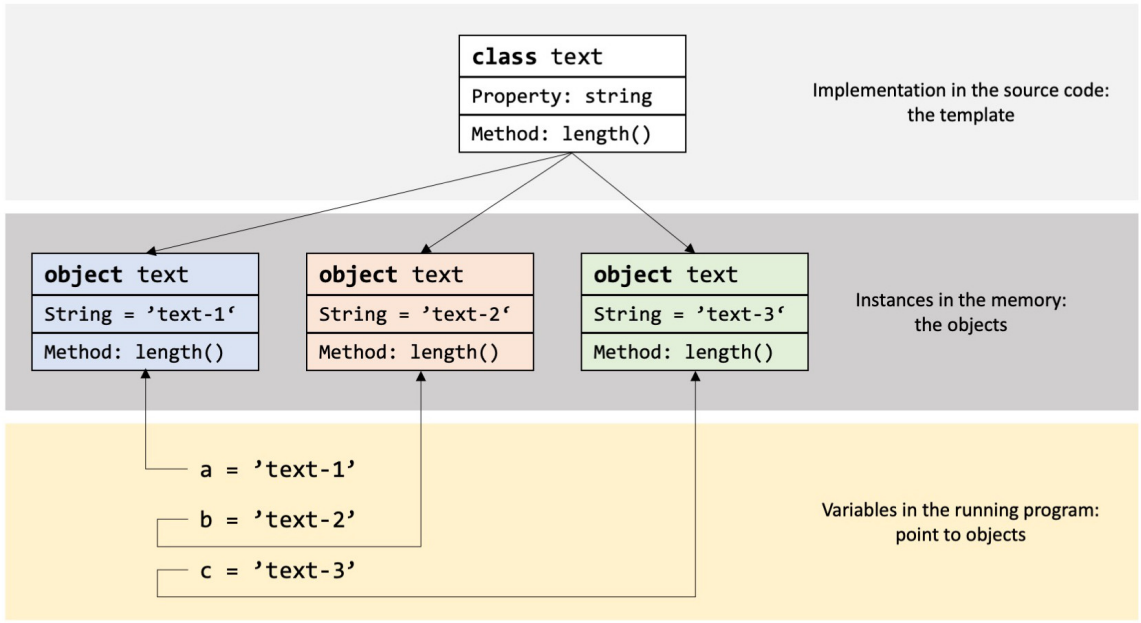
- Two variables can refer to the same content
- Re-assignment of one of these variables does not change the other content (Re-assignment in fact uses a new piece of memory)

Intermezzo: Objects

- Content in memory a variable is pointing to is an **object**
- Objects are a concept in object-oriented programming languages
- Objects are used to model something
  - Text
  - Integer number
  - Molecule
  - Employee
  - ...
- Objects have **properties** and **methods**
- Methods are in fact functions, more to that later
- Example: object to model text (string)
  - Most important property: the text that this string represents
  - Possible method: function to return the length of this string

Intermezzo: Objects

- Objects are created according to a template
- In Python such templates are so-called **classes**
- Classes in Python are identified by the keyword **class**
- Classes implement the properties and methods the objects will have
- Many objects of the same class can exist: so-called **instances**
- Every instance resides in a unique memory location
- A variable points to the memory address of a class instance (object)



Basics: Operators

- In programming languages so-called **operators** are defined
- Operators perform some kind of action or computation
- The action requires some input: the **operands**
- Operands are values (e.g. string literal) or objects (given by variables)
- Operators and their operands form so-called **expressions**

Basics: Operators

- In programming languages so-called **operators** are defined
- Operators perform some kind of action or computation
- The action requires some input: the **operands**
- Operands are values (e.g. string literal) or objects (given by variables)
- Operators and their operands form so-called **expressions**

myscript.py

```
1  # The assignment operator '='
2
3  a = "firstname"
4
5  # operand a:           to which a value is assigned
6  # operand "firstname": the content that is assigned to a
```

Basics: Operators

myscript.py

```
# The concatenation operator
print("Charly" + " " + "Brown")

# The membership operator
print("Brown" in "Charly Brown")
print("Red" in "Charly Brown")
```

```
$ python myscript.py
```

Basics: Operators

myscript.py

```python
# The concatenation operator
print("Charly" + " " + "Brown")

# The membership operator
print("Brown" in "Charly Brown")
print("Red" in "Charly Brown")
```

```
$ python myscript.py
Charly Brown
True
False
$
```

Basics: Operators

myscript.py

```
# The repetition operator
print("Charly" * "Charly")
```

```
$ python myscript.py
```

Basics: Operators

myscript.py

```python
# The repetition operator
print("Charly" * "Charly")
```

```
$ python myscript.py
Traceback (most recent call last):
  File "myscript.py", line 2, in <module>
    print("Charly" * "Charly")
TypeError: can't multiply sequence by non-int of type 'str'
$
```

Basics: Operators

myscript.py

```
1  # The repetition operator
2  print("Charly" * "Charly")
```

```
1  $ python myscript.py
2  Traceback (most recent call last):
3    File "myscript.py", line 2, in <module>
4      print("Charly" * "Charly")
5  TypeError: can't multiply sequence by non-int of type 'str'
6  $
```

Lessons:

- There must be something called **type** in Python
- String (str) is one of these types
- There seems to be another type called **int**
- It is not possible to multiply str (sequence) with a 'non-int' type
- Hmm: but it should be possible to multiply type str with type int … let's see!

Basics: Operators

myscript.py

```
# The repetition operator
print("Charly" * 3)
```

```
$ python myscript.py
```

Basics: Operators

myscript.py

```
# The repetition operator
print("Charly" * 3)
```

```
$ python myscript.py
CharlyCharlyCharly
$
```

Basics: Operators

myscript.py

```
# The repetition operator
print("Charly" * 3)
```

```
$ python myscript.py
CharlyCharlyCharly
$
```

Lessons:

- Type int obviously can represent some numeric value
- Indeed: int is a so-called **data type** to represent integer numbers

Practice Time ... 002

1. p001: Assign a variable and print its content to the command line. The output should be:
   ```
   $ A single quote "'", a double quote '"', and a backslash '\' : )
   ```
2. p002: Which are valid variable names:
   ```
   var-1, STR_1, 1_int, _rst2, def, _elif, 12, reallyNot, 1818
   ```
3. p003: Find the official Python documentation that lists all built-in functions.
4. p004: Find out what the built-in function 'len()' is doing and implement an example.
5. p005: Implement and execute the following program and reason / speculate about the problem:
   ```
   len = "10"
   var = "Hello!"
   print( len(var) )
   ```
6. p006: Write a program that reads your name and prints it 100 times in a row
7. p007: Assume we need a class 'Car' and you should implement it.
   Please identify important properties and some methods that such a class needs.

License and Contributors

Main Author
Philipp Thiel

Additional Contributors
n.a.