A very short introduction to Python3

March 11, 2020

Institute for Analysis and Scientific computing

References

Installation on Windows

- Install PYTHON 3.8.2 package from
  `https://www.python.org/downloads/windows/`
- Download NUMPY from
  `https://www.lfd.uci.edu/~gohlke/pythonlibs/`
  - 64bit: `numpy-1.18.1+mkl-cp38-cp38-win_amd64.whl`
  - 32bit: `numpy-1.18.1+mkl-cp38-cp38-win32.whl`
- Install via: `pip install numpy-1.18.1+mkl...`
- Install MATPLOTLIB via: `pip install matplotlib`

Installation on Linux

- Install PYTHON 3: `sudo apt install python3`
- Install PIP: `sudo apt install python3-pip`
- Install NUMPY: `pip3 install numpy`
- Install MATPLOTLIB: `pip3 install matplotlib`

Installation on macOS

- Install PYTHON 3.8.2 package from `https://www.python.org/downloads/mac-osx/`
- Install NUMPY: `pip3.8 install numpy`
- Install MATPLOTLIB: `python3.8 -mpip install matplotlib`

Language documentation

- Python3: `https://docs.python.org/3/reference`
- NumPy: `https://docs.scipy.org/doc/numpy`
- SciPy: `https://docs.scipy.org/doc/scipy/reference`
- MatPlotLib: `https://matplotlib.org`

Basics

Characteristics

Python is...

- ...interpreted
- ...dynamically typed
- ...garbage-collected
- ...multi-paradigm: procedural, object oriented, functional

Blocking

Python uses indentation for blocking

$\longrightarrow$ no brackets or end-statements

```
function-head
    code
    ...
    for-loop
        code
        ...
    if-statement
        code
        ...
```

Comments

Two types of comments:

```
# Single line comment

"""
Multi
line
comment
"""
```

Strings & printing

- Strings are delimited by `''`
- Every object has a string version `str()`
- Unformatted printing can be done by string concatenation

```
x = 12.345
print('The value of x = ' + str(x))
```

- Formatted printing can be done by so-called formatted strings

```
print(f'The value of x = {x:10.5f}')
```

Data structures

Objects

- Everything is an object!
  `1+2` and `(1).__add__(2)` give the same result
- `help(x)` lists all methods of object `x`
- Assignments are references
  `y = x` $\longrightarrow$ `x` and `y` point to the same object
  (Kind of an) exception: numbers
- Most objects provide methods to copy them
  `y = x.copy()`

Ranges

- for integers `start, stop, step`
- gives all integers
  `start+k*step` with `k`$\in \mathbb{N}$ in `[start, stop)`
- `range(start, stop, step)`
  `range(1, 10, 3)` $\longrightarrow$ `1,4,7`
- `range(start, stop)`
  `range(1, 5)` $\longrightarrow$ `1,2,3,4`
- `range(stop)`
  `range(5)` $\longrightarrow$ `0,1,2,3,4`
- `step` can also be negative

Lists

- collection of objects (not necessarily of same type)
  `x = [5, 'hello', [1,2]]`
- access via square brackets, 0-based indexing
  `print(x[0])` $\longrightarrow$ `5`
- negative indexing possible, index wraps around from the right
  `x[-1]` $\longrightarrow$ `[1,2]`
- `x.append(y)` adds `y` to the end of the list
- `x.pop(i)` removes element with index `i` from list
- strings are basically lists of characters

Slicing

- lists can be indexed with range-like expressions
  `x[start:stop:step]`
- `start, stop, step` work exactly as for ranges
- `:step` can be omitted
- `x[2:5]` $\longrightarrow$ all elements from 2 to 4
- `x[:5]` $\longrightarrow$ all elements from 0 to 4
- `x[2:]` $\longrightarrow$ all elements from 2 to the end of the list
- `x[:]` $\longrightarrow$ the whole list

List comprehension

- lists can be created in a very compact way

  `[<expr> for x in <iterable> if <condition>]`

  `[x*x for x in range(-3,4) if x>0]` $\longrightarrow$ `[1,4,9]`

- `<iterable>` can be any iterable object (list, range, ...)

- `if <condition>` can be skipped, multiple `if` possible

  `[x*x for x in [1,2,3]]` $\longrightarrow$ `[1,4,9]`

- alternative form allows for extended conditional

  `[<expr> if <cond> else <expr> for x in <iterable>]`

  `[x if x>0 else -x for x in range(-2,3)]`

  $\longrightarrow$ `[2,1,0,1,2]`

- list comprehensions can be nested

Tuples

- similar to lists, but with `()` instead of `[]`

  ```
  x = (5, 'hello', [1,2])
  ```

- access via square brackets, 0-based indexing, slicing possible

  ```
  print(x[0])   ⟶   5
  ```

- most notable difference: tuple elements cannot be changed

- tuples can be unpacked in a comfortable way

  ```
  a,b,c = x
  print(b)   ⟶   hello
  ```

- tuples can be concatenated by +

  ```
  (1,2,3) + (3,4,1)   ⟶   (1,2,3,3,4,1)
  ```

- tuples are often used for grouping function input/output

Dictionaries

- list of key-value pairs, where keys and values can be any object
  `{key1:value1, key2:value2, ...}`

- elements can be accessed by their key and return the value
  `x = {1:'hello', 'world':5}`
  `x[1]` $\longrightarrow$ `'hello'`
  `x['world']` $\longrightarrow$ `5`

- dictionary comprehension similar to lists possible, but with `{}` instead of `[]`

Anonymous functions

- lambda expressions serve as anonymous function
  to realize $f(x) = 2x$:
  `f = lambda x: 2 * x`
  `f(5)` $\longrightarrow$ `10`

- lambdas can take more than one parameter

- lambdas are often parameters to functions
  `reduce(lambda x,y: x**y, [2,2,2])` $\longrightarrow$ $16 = 2^{2^2}$

Control structures

Conditional statements

- basic construct

```
if <condition>:
    code
    ...
elif <condition>:
    code
    ...
else:
    code
    ...
```

- `elif` and `else` blocks are optional

Iterative loop

- `for` loop iterates over iterable object (range, list, ...)

```
for i in <iterable>:
    code
    ...
else:
    code
    ...
```

- `else` block is optional and executes after loop finishes
- stopping the loop prematurely (e.g. with `break`) prevents the `else` block from being executed

Conditional loop

- `while` loop iterates over iterable object (range, list, ...)

```
while <condition>:
    code
    ...
else:
    code
    ...
```

- executes as long as `<condition>` is true
- `else` block is optional and works as for `for` loop

Functions

- function definitions follow well-known structure

```
def function_name(parameters):
    code
    ...
    return object
```

- return object can be anything and is optional
- `pass` in function body stands for empty block
- parameters having a default value are optional

  `def f(param1, param2 = default)`

Standard library

Importing packages

- importing a library

```
import library
library.method()
```

- importing with renaming

```
import library as lib
lib.method()
```

- importing only one item (without namespace)

```
from library import method, method2, ...
method()
```

- importing everything without namespace

```
from library import *
```

Useful libraries

sys, os	system and operation system specific functions
(date)time	handle dates and times conveniently
math	all standard math operations and constants (π!)
random	random numbers of all sorts
logging	tools for automatic logging

Many libraries for common file formats: csv, xml, json, ...
Many libraries for scientific programming: NumPy, SciPy, MatPlotLib, SciKit, TensorFlow, ...