Introduction to python 3

Kevin Sturm and Winfried Auzinger

Outline

The basics

Python references

- Good python book *Python 3 (2017 edition)* by Johannes Ernesti and Peter Kaiser
- online documentation: `https://docs.python.org/3.6/`

Historical facts

- developed in the nineties by Guido van Rossum in Amsterdam at Centrum voor Wiskunde en Informatica
- the name "python" comes from the comedy "*Monty Python*"
- python **version 3.0** was released in December 2008
- one of the most popular programming languages
- designed for *functional* and *object oriented* programming
- programs that partially use python:
  - Google Mail
  - Google Maps
  - YouTube
  - Dropbox
  - reddit
  - Battlefield 2
  - BitTorrent

Why python?

What does python offer?

- Interactive
- Interpreted
- Modular
- Object-oriented
- Portable
- High level
- Extensible in C++ & C

Why is python good for scientifc computing?

- open source / free
- many libraries, e.g.,
- scientific computing: numpy, scipy
- symbolic math: sympy
- plotting: matplotlib
- excellent PDE solver software: ngsolve, FEniCs, Firedrake, ...

How to start python?

- Python can either be used **interactively**: simply type "python3" or "ipython3" (to start IPython) into the shell
- we can also **execute python code** written in a file "file.py" by typing "python3 file.py" into the shell

Let's start with a hello world example:

Listing 1: hello_world.py

```
1 """ This is our first program """
2
3 print("Hello world!")
```

Float

declaration of floats

```
>>> x = 987.27
>>> x
987.27
```

division

```
>>> y = 2.27
>>> x/y
434.92070484581495
```

floor division

```
>>> x//y
434.0
```

addition and subtraction

```
>>> x = 987.27
>>> y = 2.0
>>> x+y
989.27
>>> x-y
985.27
```

powers

```
>>> x**2
974702.0529
>>> x**3
962294095.766583
>>> x**0.5 # square root
31.4208529483208
```

multiplication

```
>>> x*y
1974.54
>>> x*-y
-1974.54
```

Integers

calculator

```
>>> 1+3
4
>>> 3-10
-7
>>> 30*3
90
```

declaration of integer

```
>>> x = 987
>>> x
987
>>> z = int(10.0)
>>> z
10
```

multiplication and division

```
>>> y = 2
>>> x/y
493.5
>>> 5/3
1.6666666666666667
```

floor division

```
>>> x//y
493
```

conversion of float to integer

```
>>> x = 1.4
>>> y = int(x)
>>> y
1
>>> x + 3
4.4
```

- remember: float + int = float

Complex number

- imaginary unit in python is j
- recall $(a + ib) * (c + id) := ac - db + i(bc + ad)$

```
>>> z = 1.0 + 5j  # complex number with real 1 and imag 5
>>> z.conjugate()  # conjugate complex number
(1-5j)

>>> z = complex(1,5) # equivalent to 1+5j

>>> z.imag # return imaginary part
5.0
>>> z.real # return real part
1.0
```

Complex number (continued)

multiplication of complex numbers

```
>>> z1 = 1 + 4j
>>> z2 = 2 - 4j

>>> z1*z2  # multiply z1 and z2
(18+4j)

>>> # Let us verify this is correct

>>> a, b, c, d  = z1.real, z1.imag, z2.real, z2.imag

>>> a*c - b*d
18.0
>>> b*c + a*d
4.0
```

Strings

declaration of strings

```
>>> a = "hello" # assign hello
>>> a
'hello'
```

addition of strings

```
>>> a+a
'hellohello'
>>> a+" cool"
'hello cool'
```

referencing letters

```
>>> fourth =   a[3]  # 4th letter
>>> fourth
'l'
>>> last = a[-1]  # last letter
>>> last
'o'
```

conversion of float and integer to string

```
>>> x = 987.27
>>> s1 = str(x)
>>> s1
'987.27'
>>> n = 10
>>> s2 = str(n)
>>> s2
'10'
```

Strings (continued)

lower and upper case

```
>>> a = "hello" # assign hello
>>> a.upper()
'HELLO'

>>> a = "HELLO"
>>> a.lower()
'hello'
>>> a
'HELLO'

>>> a = "Hello"
>>> a.swapcase()
'hELLO'
>>> a
'Hello'
```

accessing letters

```
>>> s = "This is a long sentence!"
>>> s[::3] # every third letter
'Tss nstc'
>>> s = "z"
>>> 10*s
'zzzzzzzzzz'
```

Splitting and concatenation

```
>>> name = "This is a long sentence."
>>> name.split()
['This', 'is', 'a', 'long', 'sentence.']
>>> name
'This is a long sentence.'
```

inserting strings

```
>>> 'Insert here: {}'.format('Inserted string')
'Insert here: Inserted string'
```

Lists

declaration of list

```
>>> l = []  # empty list
>>> l
[]
>>> l = [1, 2, 3] # integers list
>>> l
[1, 2, 3]
>>> l = [1.0, 3.0, 3,0] # float list
```

lists can contain anything

```
>>> l1 = [1,2,3]
>>> l2 = ["hello", [], "new"]
>>> l = [l1, l2]
>>> l
[[1, 2, 3], ['hello', [], 'new']]
```

other ways to generate lists

```
>>> l1 = [1]*5
>>> l1
[1, 1, 1, 1, 1]
>>> l2 = [k for k in range(5)]
>>> l2
[0, 1, 2, 3, 4]
```

The last command is similar to the mathematical definition $\{k : \ k = 0, 1, 2, 3, 4\}$.

addition of lists

```
>>> l1 = [1,2,3]
>>> l2 = [4,5,6]
>>> l1+l2
[1, 2, 3, 4, 5, 6]
```

multiplication not supported

```
>>> l1*l2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can't multiply sequence by n
```

More on lists

The *list* class has the following methods:

- append
- clear
- copy
- count
- extend
- index
- insert
- pop
- remove
- reverse
- sort

```
>>> l = [1, 2, 3, 4, 4]
>>> l
[1, 2, 3, 4, 4]

>>> l.reverse()
>>> l
[4, 4, 3, 2, 1]

>>> l.pop(3)
2
>>> l
[4, 4, 3, 1]

>>> # print every 2nd element
>>> # start with index 1
>>> # go until end of list -1
>>> # the : operation is called slicing
>>> l[1:-1:2]
[4]
```

Tuple

- Tuple are essentially uneditable lists. We use round parenthesis.
- referencing possible, but no assignment
- to be used when list should not be modified

declaration of list

```
>>> l = ()  # empty tuple
>>> l
()
>>> l = (1, 2, 3) # tuple of integers
>>> l
(1, 2, 3)
>>> l = tuple([1.0, 3.0, 3,0]) # same
>>> l
(1.0, 3.0, 3, 0)
```

adding tuples

```
>>> l+l
(1.0, 3.0, 3, 0, 1.0, 3.0, 3, 0)
>>> 4*l
(1.0, 3.0, 3, 0, 1.0, 3.0, 3, 0, 1.0, 3.0, 3, 0, 1.0, 3.0, 3, 0)
```

Bool and logical operators

bool True or False

```
>>> t = True
>>> t
True
>>> f = False
>>> f
False
>>> f == t
False
```

"and", "or", and "not

```
>>> t and f
False
>>> t or f
True
>>> not f == t
True
```

Possibilities for "or":

x	y	x or y
True	True	True
True	False	True
False	True	True
False	False	False

Possibilities for "and":

x	y	x and y
True	True	True
True	False	False
False	True	False
False	False	False

If-else

simple if-else statement

Listing 2: if_else.py

```
if condition:
    command
else:
    another command
```

When we have more than one condition we use *elif*:

Listing 3: if_else2.py

```
if condition1:
    first command
elif condition2:
    second command
else:
    third command
```

If-else example

Listing 4: if_else_ex.py

```
1 if x == 1:
2     print("x has value 1")
3 elif x == 2:
4     print("x has value 2")
```

Listing 5: if_else_ex2.py

```
1 if x == 1:
2     print("x has value 1")
3 else:
4     print("x has another value")
```

for loop

Listing 6: for_loop.py

```
1 for n in range(10):
2     print(n)
```

- Here *n* ranges from 0 to 9 and is printed after each loop.
- general syntax is *range(start, stop, steps)*
- *start* and *steps* are optional

Listing 7: for_loop2.py

```
1 l = [0, 1, 'hello', True, False]
2
3 for n in l:
4     print(n)
```

for loop (continued)

- use *enumerate* to count the element in the loop

Listing 8: for_loop_en.py

```
1 l = ['one', 'two', 'three', 'four', 'five']
2
3 for n, s in enumerate(l):
4     print('Item number ', n, ' item itself ', s)
```

While loop

The syntax of a python while loop is as follows.

```
while statement:
    do stuff
```

- "do stuff" is executed as long as statement is true.
- notice again the indention!
- use *break* to leave a while loop
- use *continue* to go to the next loop

Listing 9: while_loop.py

```
counter = 10

while counter > 0:
    print("counter is", counter)
    counter -= 1
```

Functions

Let's have a look at an example function.

Listing 10: func.py

```
1 def my_func(x):
2     x = x + 1.0
3     return x
```

- indention in python replaces brackets!!!
- a function always starts with *def*
- a *return* is not mandatory
- without *return* the function returns None.

Functions (continued)

- anonymous functions can be defined using *lambda* keyword

```
>>> f = lambda x: x**2 # define lambda function f
>>> f(2)
4
```

a more complicated example

```
>>> f = lambda x: x**2 if x < 0 else x**3
>>> f(2)
8
>>> f(-3)
9
```

This is equivalent to:

Listing 11: lambda_func.py

```
1 def f(x):
2     if x < 0:
3         return x**2
4     else:
5         return x**3
```

Functions (optional arguments)

- It is possible to give functions optional arguments.

Listing 12: func_opt.py

```
1  def f(x, y=None):
2
3      if y == None:
4          return x**2
5      else:
6          return x**2 + y**2
7  print(f(1))
8  print(f(1,2))
```

Dictionaries

- make a dictionary with {} and : to signify a *key* and a *value*

```
>>> value1 = 1.0
>>> value2 = 2.0
>>> my_dict = {'key1':value1,'key2':value2}

>>> print(my_dict)
{'key1': 1.0, 'key2': 2.0}

>>> my_dict['key1'] # access value1
1.0

>>> 'key2' in my_dict
True
```

Dictionaries (continued)

Accessing the values and the keys

```
>>> # Make a dictionary with {} and : to signify a key and a value
>>> value1 = 1.0
>>> value2 = 2.0
>>> my_dict = {'key1':value1,'key2':value2}


>>> print(my_dict.values())  # return values of dictionary
dict_values([1.0, 2.0])

>>> print(my_dict.items()) # return items
dict_items([('key1', 1.0), ('key2', 2.0)])

>>> print(my_dict.keys()) # return keys
dict_keys(['key1', 'key2'])
```

Sets

- sets are unordered lists

declaration of sets

```
>>> S = set([1,2,3,4])  # def. a set S
>>> S
{1, 2, 3, 4}

>>> S = {1,2,3,4} # equiv. definition
>>> S
{1, 2, 3, 4}
```

union $\cup$ and subtraction $\setminus$ of sets

```
>>> S1 = {1,2,3}
>>> S2 = {2,3,4}

>>> S1 - S2  # subtract S1 from S2
{1}
>>> S2 - S1  # subtract S2 from S1
{4}

>>> S1 | S2  # union of S1 and  S2
{1, 2, 3, 4}

>>> S1^S2  # symmetric difference
{1, 4}
```

Sets (continued)

alternative definition

```
>>> S1 = {2,3,4,5}
>>> S2 = {1,2,3,4}

>>> S1.intersection(S2)
{2, 3, 4}

>>> S2.union(S1)
{1, 2, 3, 4, 5}

>>> S1.difference(S2)
{5}
```

union $\cup$ and subtraction $\setminus$ of sets

```
>>> S1 = set([1,2,3])
>>> S2 = set([2,3,4])

>>> S1 - S2  # S1/S2
{1}
>>> S2 - S1  # S2/S1
{4}

>>> S1 | S2 # union of S1 and S2
{1, 2, 3, 4}
```

adding and deleting elements

```
>>> S1.add(10) # add 10 to list
>>> S1
{10, 1, 2, 3}

>>> S1.discard(10) # remove element 10
>>> S1
{1, 2, 3}
```

Python key words

- We already know a few python key words.
- The *keywords* are part of the python programming language.
- you cannot use these names for variables or functions

and	def	finally	in	or	while
as	del	for	is	pass	with
assert	elif	from	lambda	raise	yield
break	else	global	None	return	
class	except	if	nonlocal	True	
continue	False	import	not	try	

Figure: List of python keywords

Importing modules

- import a module with command "**import module_name**"
- a function **func** in **module_name** can be accessed by module_name.func
- including with different name use "**import module_name as mn**"
- import specific function "**from module_name import func**"
- import everything "**from module_name import ***"

Math modul

Let us consider as an example the *math* package.

```
>>> import math  # import math module and use name "math"
>>> math.pi
3.141592653589793
>>> del(math) # remove math package

>>> import math as m # import math module with name "m"
>>> m.pi
3.141592653589793
>>> del(m)

>>> from math import pi # import constant pi from math
>>> pi
3.141592653589793

>>> from math import pi as pipi # import constant pi from math with name "pipi"
>>> pipi
3.141592653589793
```