Python Basic

Python introductory course for programmers

29 January 2018

Konrad Brodzik
Graphics Software Engineer, Intel VPG

Disclaimers and licensing info

The views expressed in this presentation are those of the author and do not necessarily represent or reflect the positions, strategies or opinions of his employer, Intel Corporation.

Other names and brands may be claimed as the property of others.

Names marked with an asterisk(*) are registered trademarks.

What is Python*?

- Script language released in 1991 by Guido van Rossum (BDFL)
- Presently we have 2 incompatible versions: 2.x, 3.x
- Compiles to bytecode
- Works on Windows*, Linux*, MacOS*, FreeBSD*, Android*
- Developed by Python Software Foundation

After this training you should be able to:

- Understand most Python* code
- Write simple apps
- Know some good practices
- Know some commonly used libraries

Editing and running your code

- Use the 'python' command and type code line by line
- Use Idle on Windows* to edit and F5 to run
- Use PyCharm* for complex development
- Call python some_file.py

How does Python* code look?

```
# New Python* module

def sum_list(var_list):
""" This function sums the integers in the list and returns an integer

:param var_list: list of integers
:return: sum of provided integers
"""
sum = 0
for var in var_list:
  sum += var
return sum
```

How does Python* look?

- no semicolons or curly braces to define blocks
- nested statements controlled by indents (spaces or tabs)
- some statements (if, elif, else, for, class, def) terminated with colon (:)

```
# single line comment
""" Multiline
    comment
"""
```

- Many formats of documentation strings, choose one and stick to it (PEP257)

Coding standard (PEP8)

- Variables, functions, modules - snake_case
- Class names - CamelHumps
- Constants - UPPERCASE
- Spaces around operators(+,-,=,/)
- pep8 tool to validate if you're ok

basic types & basic operations - int, str, float

- Python uses a couple of basic data types that have builtin operators and type conversion
- = assignment operator
- == value equality
- strings can be single quoted, double quoted, or triple quoted
- arithmetic (+, -, *, <, >, <=, >=, %, !=) and bitwise (<<, >>, ^, &, |) operators work like

in most programming languages.

Some interesting features:

```
5 / 2
5 // 2
5 ** 2
a, b = 5, 2
a, b = b, a
```

Some interesting features:

```
# Upgrade to float by default
>>> 5 / 2
2.5
# Force integer division
>>> 5 // 2
2
# Integrated powers
>>> 5 ** 2
25
# Multiple assignment & value swap
>>> a,b = 5,2
>>> a
5
>>> b
2
>>> b, a = a, b
>>> b
5
>>> a
2
```

Type conversions

```
a = 15
b = "15"
a == b
str(a) == b
a == int(b)
a == float(b)
```

Formatting strings

```
n = 5
m = 15.0
"Lucky numbers %d %f" % (n, m)
'This is number {0:.3f}'.format(n)
f"This is number {n}"
"""This is number {n:x}""".format(n=5)
```

Cool string methods

```
print(5*"help")
"aaa".count("a")
"info".find("o")
"info".replace("o", "os")
```

Bulitin methods

```
a = "data"
print(len(a))
dir(a)
help(a.lower)
dir(_ _ builtins _ _)
```

Double underscore generally means special names.

Bulitin methods

```
>>> a="data"
>>> len(a)
4
>>> dir(a)
['__add__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__', '__eq__', '__format__', '_
>>> help(a.lower)
Help on built-in function lower:

lower(...) method of builtins.str instance
    S.lower() -> str

    Return a copy of the string S converted to lowercase.

>>>
```

I/O - console

- print

```
a = 15
print("some text")
print(f"text with variable {a}")
print("text with variable %x" % a)
```

- input

```
>>> a = input()
15
>>> a
'15'
```

Excercise - I/O and formatting

Read a number from commandline, add 60, print in hexadecimal form (eg. print 4b for input 15)

I/O - file

```
f = open("file.txt", "w")
f.write("test")
f.close()


with open("file.txt") as f:
   data = f.read()
```

```
with open("file.txt") as f:
 for line in f:
   print(line)
```

If statement

base form:

```
a = 1
if a > 100:
  print("Greater than 100")
elif a > 1:
  print("Greater than 1")
else:
  print("Really big or really small")
```

Trinary if operator:

```
5 if a == 1 else -5
```

Collections - lists

```
a = []
a.append(1)
a.append("string")
a.append(5.0)
print(a[0])
print(a[1])
a.reverse()
print(a)
a.index(5.0)
print(len(a))
a.remove("string")
a.pop()
```

List to string and back

```
a = ["a", "b", "c"]
lst = " ".join(a)
for c in lst.split(" "):
  print(c)
```

List slicing (works also with strings)

```
a = [1,"string",5.0]
print(a[0])
print(a[-1])
print(a[1:])
print(a[:-2])
```

List nesting

```
b = [[1,2,3],["a", "b", "c"]]
print b[0][0]
```

23

Collections - dictionaries

```
a = {}
a['me'] = "user"
a[5] = "user"
a['user_details'] = {"name": "Janusz"}
print(a)
print(a["user_details"])
print(a["notfound"])
print(a.get("notfound", "Value not in dict"))
a.setdefault('me', 'admin')
print(a)
a.keys()
a.values()
a.update({"order_details": ""})
print(a)
```

Collections - sets

- no element repeats, set algebra included
- set cannot be modified

```
a = set([1,2,3,3,5])
b = set([1])
a & b
a | b
a ^ b
b.issubset(a)
```

Loops

```
# Python style loop
numbers = [3,2,1]
for element in numbers:
  print(element)
# Legacy style loop
for i in range(len(numbers)):
  print(numbers[i])
# Infinite while loop + break condition
while True:
  a=input()
  if a == "yes":
    break
```

Loop + list = list comprehension

```
[a.upper() for a in "swag" if a != "a"]
```

- Also comes in dict flavour

```
{letter : index for letter, index in enumerate("swag")}
```

Excercise - files and collections

Open "lorem_ipsum_1k.txt", read it, count how many unique words it contains, count how many times the word "lorem" shows up. (30 minutes)
(Hint: split file content by spaces, lowercase, remove all dots, commas and newlines, create set from list)

Functions

```
# Positional and keyword arguments
def add_integers(left, right=0):
  return left + right
add_integers(2)
add_integers(2, 5)
add_integers(left=2, right=5)

# Variable arguments
def add_integers(*integers):
  for integer in intergers:
    print(integer)
  # Returns None by default
```

- Functions are just objects.. Try it!

```
a = add_integers
a(2)
```

Functions - passing reference or value?

It depends whether the argument is immutable (string, int, float, bool) or mutable (everything else).

Immutable:

```
a = 5
def change_argument(number):
  global a
  number = number + 5
  a += 1
change_argument(a)
print(a)


Others:


 a = [1,2]
 def change_argument(my_list):
   my_list.append(3)
 change_argument(a)
 print(a)
```

Modules and imports

- File module

Create a file called "constants.py" with value TIMEOUT=5

Run python console:
# Import module
import constants
print(constants.TIMEOUT)
# Import object to namespace
from constants import TIMEOUT as timeout
print(timeout)

Directory as a module

Put constants.py in a directory called "my_module"
Try:
import my_module.constants
Try:
create _ _ init_ _.py file in "my_module" directory
import my_module.constants

Modules and imports

- every .py file is a module and can be imported
- directory with __init__.py inside is a module
- file names - **DON'T** use simple names like file, os, console - you can override system libraries

External modules

- Search Python Package Index (http://pypi.python.org) - most useful libraries are there
- install via pip install package_name

Writing commandline tools - argparse

```
import argparse

ap = argparse.ArgumentParser()
ap.add_argument("-ip", "--ip_address", help="IP to convert", required=True)
parsed = ap.parse_args()
print(parsed.ip_address)
```

```
C:\>python ap.py
usage: ap.py [-h] -ip IP_ADDRESS
ap.py: error: the following arguments are required: -ip/--ip_address
```

Excercise - IP converting utility

- Read dotted decimal ip address (e.g. 192.168.1.1) from command line argument using argparse and print it in hex (e.g. c0a80101)

Working with HTTP requests - requests

```
import requests
r = requests.get("http://www.wp.pl")
print(r.status_code)
print(r.text)
```

Excercise - fetch and parse a json file from the Web

Fetch Gdansk weather page from Yahoo APIs and print a list of day + expected temperature

API link (https://query.yahooapis.com/v1/public/yql?

q=select%20*%20from%20weather.forecast%20where%20woeid%20in%20%28select%20woeid%20from%20geo.places%281%29%20where%20text%3D%22gdansk%2C%20poland%22%29&forr

```
import requests
data = requests.get("paste API link here")
data.json() # find data here. Hint: look at the file structure in your browser with JSON extension insta
```

Resolving order - LEGB

- Local, Enclosing, Global, Builtin
- global scope - top of file
- you can define a variable in a if/for statement, and use it outside it, e.g.

```
if True:
  a = 1
print(a)
```

But watch out:

```
import requests
requests=""
requests.get("http://wp.pl")
```

Classes

- Special keyword - self - reference to object instance (like 'this' in Java)
- Keyword - super - refer to parent class
- Multiple inheritance
- protected members with _method, private members with __method - but everything is public, really
- Magic methods - _ _ init _ _(), _ _ eq _ _(), _ _ ne _ _(), _ _ str _ _()

Classes

```
class Parent:
  def __init__(self, a):
    self.a = a
  def method(self):
    print(self.a)

obj = Parent(2)
obj.method()

class Child(Parent):
  def __init__(self,a):
    super(Child, self).__init__(a)
    self.b = "data"
  def __eq__(self, right):
    return self.a == right.a

obj = Child(2)
obj.method()
print(obj.b)
obj2 = Child(2)
print(obj == obj2)
obj2 = OtherChild(2)
print(obj == obj2)
```

Classes - excercise

- Implement _ _ ge _ _() magic method so that obj >= obj2 works
- Implement _ _ str _ _ () magic method and try str(obj) before and after adding it (similar to toString() method in Java/C#)
- Create a class called OtherChild (not inheriting from Parent) also having self.a set - see if _ _ eq _ _ will work

Exceptions

- Exceptions are just normal classes inheriting from Exception
- Multiple exceptions can be caught in one block, e.g. (ValueError, TypeError)
- Multiple except blocks possible

```
try:
    int("text")
except ValueError as e:
    print(str(e))
finally:
    print("Parsing completed")
```

- To throw an exception use

```
raise Exception("Something went wrong")
```

Excercise - exceptions

- Improve IP parsing script from previous excercises to handle ValueError when text is entered instead of IP

Fast membership testing

```
names = ['John', 'James']
if 'John' in names:
  print("John is onboard")
```

```
addresses = {'John': '1 Cool Street, London'}
if 'John' in addresses:
  print("We have John's address, too ")
```

PDB

- Python debugger
- Similar to gdb in C
- Use p to print variables (you can also execute bits of code) and n move to the next line
- Most common ways of starting pdb

1. Run your program through python -m pdb your_program.py
2. Insert the line "import pdb; pdb.set_trace()" in the line where you want to stop
3. Run a function from Python console and enter pdb.pm() after it crashes

PDB explained (https://pymotw.com/3/pdb/)

Excercise - PDB

Run this program in PDB, find why it doesn't work and fix the error.

remove_file.py:

```
import os

file_to_remove = os.path.join(os.getcwd(), "non_existent_file")

print("Do you want to remove %s? Please answer yes or no" % file_to_remove)

answer = input()
answer_is_true = bool(answer)

if answer_is_true:
  print("Removing file %s" % file_to_remove)
  os.unlink(file_to_remove)
```

Explanation for the previous excercise

Truth Value Testing (https://docs.python.org/3.6/library/stdtypes.html)

Data storage - JSON files

```
import json

entries = [{"date":"30 May", "text": "Hello world"}, {"date": "1 June", "text": "New discovery"}]

with open("entries.json","w") as f:
  json.dump(entries, f)
with open("entries.json","r") as f:
  restored_entries=json.load(f)
print("After restore")
print(restored_entries)
```

It's also possible to store Python objects - use the module

Excercise - data storage

JSON files are a simple way to store data when you don't need much performance.

Write a short program that will store diary entries with dates. Use input() to read the entry and automatically add a date using datetime.datetime.now()
(You need to import the datetime module).

Function decorators

- Useful design pattern for adding extra logging, exceptions, connection closing etc.

```
import time
from functools import wraps

def logging_decorator(f):
  @wraps(f)
  def wrapper(*args, **kwds):
    print('Entering decorator')
    result=f(*args, **kwds)
    print('Exiting decorator')
    return result
  return wrapper

@logging_decorator
def do_something_long():
  print("I'm feeling sleepy")
  time.sleep(5)

do_something_long()
```

Multiprocessing

- Python Global Interpreter Lock (GIL) makes multithreading impractical
- .. but multiprocessing is easy

```
import multiprocessing
import time

def compute_squares(number):
  return number ** 2

# Python idiom - the code below will not get run if you import the module
if __name__ == '__main__':
  pool = multiprocessing.Pool(20)
  result = pool.map(compute_squares, range(1000))
  print(result)
```

Excercise - decorators + multiprocessing

1. Make the multiprocessing example a function (i.e. compute squares in parallel in one Python function)
2. Create a decorator that will measure execution time (use time.time() before and after function)
3. Measure if adding more processes helps (add decorator to your function and run it in a loop with different process numbers)

Simple HTTP server

```
from http.server import BaseHTTPRequestHandler

class GetHandler(BaseHTTPRequestHandler):
    def do_GET(self):
        self.send_response(200)
        self.send_header('Content-Type',
                         'text/plain; charset=utf-8')
        self.end_headers()
        import datetime
        message = "I'm your new website!" + str(datetime.datetime.now())
        self.wfile.write(message.encode('utf-8'))


if __name__ == '__main__':
    from http.server import HTTPServer
    server = HTTPServer(('localhost', 8080), GetHandler)
    print('Starting server, use <Ctrl-C> to stop')
    server.serve_forever()
```

Excercise - HTTP server

Use the sys and date builtin modules to print interesting information about your computer (Windows version and date)
on the website

Further reading/watching

- Dive into Python - very detailed book (http://www.diveintopython3.net/)
- PyGDA meetings - local Python community, meets every month (https://www.meetup.com/PyGda-pl/)
- Flask - simple Web framework (http://flask.pocoo.org/docs/0.12/)
- SQLAlchemy - good database engine (https://www.sqlalchemy.org/)
- Python bytecode presentation (https://www.youtube.com/watch?v=mxjv9KqzwjI&list=PLWOgCnnqygQ3XLPJIGUoKkGOQ5swogPdC&index=1)

Thank you

Konrad Brodzik
Graphics Software Engineer, Intel VPG
konrad+pygda@brodzik.it (mailto:konrad+pygda@brodzik.it)
http://github.com/Kondziowy (http://github.com/Kondziowy)
@lamafix (http://twitter.com/lamafix)