PROSA/Enigma Webinar

Python Refresh

Aslak Johansen asjo@mmmi.sdu.dk

Apr 28, 2021

SDU

Part 1:
Getting Started

Two Modes of Execution

aslak@thera: ~

```
aslak@thera:~$ python3
Python 3.8.5 (default, Jul 28 2020, 12:59:40)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print('Hello, World!')
Hello, World!
>>>
```

Open hello.py /tmp Save

```
1 #!/usr/bin/env python3
2
3 print('Hello, World!')
4
```

Python 3 Tab Width: 4 Ln 4, Col 1 INS

Imports

```
import os
from sys import argv
from sys import exit as bye

print(os.name)
bye()
```

First Steps

```python
#!/usr/bin/env python3
import sys

print("Hello, world!")
sys.exit() # this is really not necessary
```

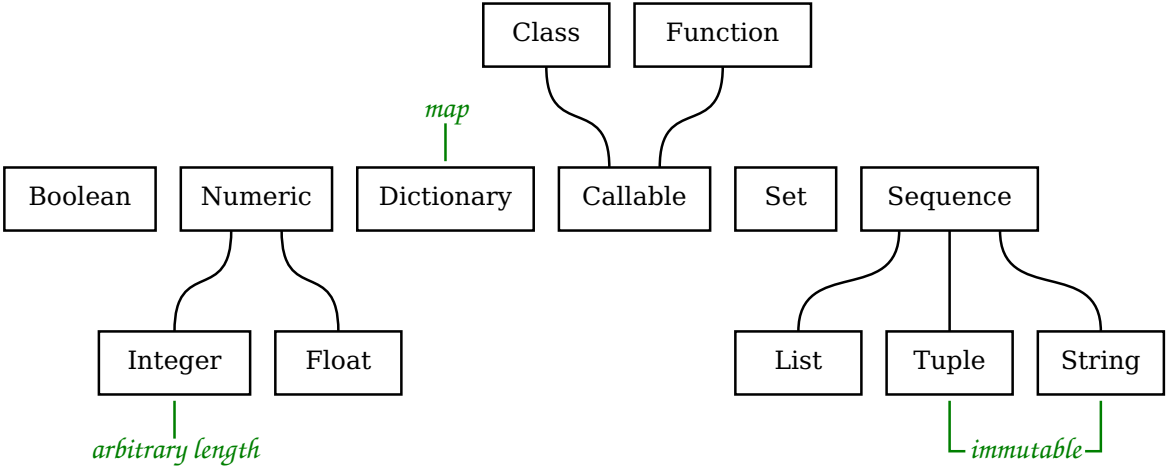
Command-Line Arguments

```python
from sys import argv, exit

if len(argv) != 3:
    print('Syntax: %s INPUT_FILENAME OUTPUT_FILENAME' % argv[0])
    print('        %s log.txt analysis.csv' % argv[0])
    exit(1)
input_filename  = argv[1]
output_filename = argv[2]

print('%s -> %s' % (input_filename, output_filename))
```

```
aslak@thera:~/vcs/git/dm-course/src/python$ python3 process.py
Syntax: process.py INPUT_FILENAME OUTPUT_FILENAME
        process.py log.txt analysis.csv
aslak@thera:~/vcs/git/dm-course/src/python$ python3 process.py log.txt analysis.csv
log.txt -> analysis.csv
aslak@thera:~/vcs/git/dm-course/src/python$
```

Part 2:
Basic Datatypes and -Structures

Types


Class
Function
map
Boolean
Numeric
Dictionary
Callable
Set
Sequence
Integer
Float
List
Tuple
String
arbitrary length
immutable

Boolean Operators

In python boolean operators are spelled out.

```
if page < pagecount and good_book:
    print('Enjoy!')

while not there:
    print('Are we there yet?')

if finished or not started:
    print('Not much is happening :-(')
```

String Operations

```
>>> s1 = "Alice was beginning to get very tired of sitting by her sister on the bank"
>>> s1
'Alice was beginning to get very tired of sitting by her sister on the bank'
>>> s2 = 'and of having nothing to do'
>>> s2
'and of having nothing to do'
>>> s = s1+", "+s2
>>> s
'Alice was beginning to get very tired of sitting by her sister on the bank, and of having nothing to do'
>>> s = s.replace(',', '')
>>> s
'Alice was beginning to get very tired of sitting by her sister on the bank and of having nothing to do'
>>> words = s.split(' ')
>>> words
['Alice', 'was', 'beginning', 'to', 'get', 'very', 'tired', 'of', 'sitting', 'by', 'her', 'sister', 'on', 'the',
'bank', 'and', 'of', 'having', 'nothing', 'to', 'do']
>>> '%d: %s' % (3, 'March')
'3: March'
>>> '_'.join(words)
'Alice_was_beginning_to_get_very_tired_of_sitting_by_her_sister_on_the_bank_and_of_having_nothing_to_do'
```

Functions

```
def add (a, b, c=0):
    return a+b+c

print(add(1,2,3))
print(add(1,2))

a = add
print(a(1,2))
```

```
6
3
3
```

Type Introspection

```
>>> t = type(True)
>>> t
<class 'bool'>
>>> type(t)
<class 'type'>
>>> type(bool)
<class 'type'>
>>> t == bool
True
```

Type Introspection

```
>>> def fun(var): return var
...
>>> type(fun)
<class 'function'>
>>> f = fun
>>> type(f)
<class 'function'>
>>> f(1)
1
>>> g = lambda a: a
>>> type(g)
<class 'function'>
>>> g(1)
1
```

Object-Orientation

```
>>> def fun(var): return var
...
>>> type(fun)
<class 'function'>
>>> f = fun
>>> type(f)
<class 'function'>
>>> f(1)
1
>>> g = lambda a: a
>>> type(g)
<class 'function'>
>>> g(1)
1
```

Part 3:
Flow Control

Branching

```
if len(lines)>0 and len(line[0])>0 and line[0][0]=='#':
    print('First line is a comment')

parts = line.split(' ')
command = parts[0]
if   command=='load':
    load_file()
elif command=='save':
    save_file()
elif command=='quit':
    quit()
else:
    print('Unknown command "'+command+'"')
```

Missing For-Loop

Python does not have a *for* loop.

Python has a *foreach* loop.

Iterating over a list:

```
for line in lines:
    print(line)
```

Iterating over a list with access to the index:

```
for i in range(len(lines)):
    line = lines[i]
    print(str(i)+': '+line)
```

Generating Ranges of Integers

The `range` function returns a generator for a sequence of integers.

```
>>> range(5)
range(0, 5)
>>> list(range(5))
[0, 1, 2, 3, 4]
>>> list(range(1,5))
[1, 2, 3, 4]
>>> list(range(1,5,2))
[1, 3]
>>> for i in range(1,5,2):
...     print(i)
...
1
3
```

Part 4:
Lists

List Operations

```
>>> l = [1,2,3]
>>> l
[1, 2, 3]
>>> l.append(4)
>>> l
[1, 2, 3, 4]
>>> l.extend([7,6,5])
>>> l
[1, 2, 3, 4, 7, 6, 5]
>>> sorted(l)
[1, 2, 3, 4, 5, 6, 7]
>>> l
[1, 2, 3, 4, 7, 6, 5]
>>> l.sort()
>>> l
[1, 2, 3, 4, 5, 6, 7]
>>> len(l)
7
>>> l[2], l[-1]
(3, 7)
>>> l[2:]
[3, 4, 5, 6, 7]
>>> l[2:4]
[3, 4]
>>> l[:4]
[1, 2, 3, 4]
>>> 4 in l, 42 in l
(True, False)
```

Higher-Order Functions over Lists

```
>>> l = [-17,2,5,-4,4,7,-3,-1,9,1]
>>> incr = lambda v: v+1
>>> map(incr, l)
<map object at 0x7f33c8440b20>
>>> list(map(incr, l))
[-16, 3, 6, -3, 5, 8, -2, 0, 10, 2]
>>> pos = lambda v: v>=0
>>> filter(pos, l)
<filter object at 0x7f33c8440b20>
>>> list(filter(pos, l))
[2, 5, 4, 7, 9, 1]
>>> list(map(incr, filter(pos, l)))
[3, 6, 5, 8, 10, 2]
```

Part 5:
Dictionaries

Basic Operations

```
>>> {}
{}
>>> d = {'jan': 1, 'feb': 2, 'mar': 3}
>>> d
{'jan': 1, 'feb': 2, 'mar': 3}
>>> d['jan']
1
>>> d['apr'] = 4
>>> d
{'jan': 1, 'feb': 2, 'mar': 3, 'apr': 4}
>>> d['list'] = [1,2,3]
>>> d
{'jan': 1, 'feb': 2, 'mar': 3, 'apr': 4, 'list': [1, 2, 3]}
>>> 'jan' in d
True
>>> 'may' in d
False
>>> d.keys()
dict_keys(['jan', 'feb', 'mar', 'apr', 'list'])
>>> list(d.keys())
['jan', 'feb', 'mar', 'apr', 'list']
>>> for key in d: print(key)
...
jan
feb
mar
apr
list
>>> del(d['feb'])
>>> d
{'jan': 1, 'mar': 3, 'apr': 4, 'list': [1, 2, 3]}
```

Part 6:
Strings

Strings: Basic Operations

```
>>> initial = ' once upon a time  '
>>> initial
' once upon a time  '
>>> len(initial)
19
>>> stripped = initial.strip()
>>> stripped
'once upon a time'
>>> words = stripped.split(' ')
>>> words
['once', 'upon', 'a', 'time']
>>> words[1], stripped[1]
('upon', 'n')
>>> joined = '_'.join(words)
>>> joined
'once_upon_a_time'
```

Regular Expressions

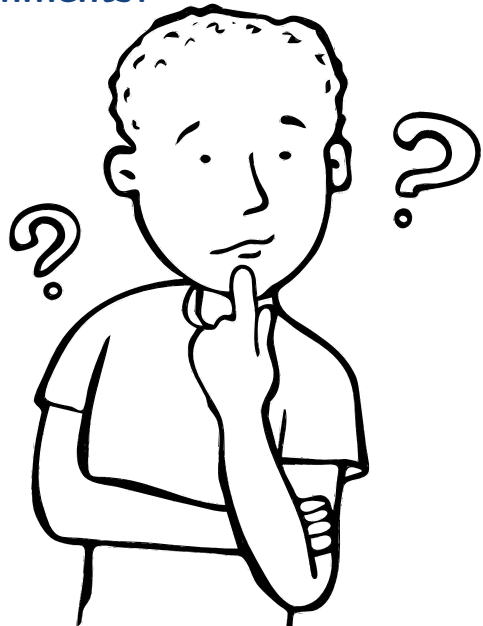
```python
import re

urls = [
    'https://www.gutenberg.org/files/11/11-h/11-h.htm',
    'https://golang.org',
    'http://www.google.com:80/',
    'definitely not a URL',
]
pattern = re.compile('([^:]+)://([^:/]+)(:\d+|)(/.*|)')

for url in urls:
    mo = pattern.match(url)

    if mo:
        print('proto="%s" domain="%s" port="%s" path="%s"' %
              (mo.group(1), mo.group(2), mo.group(3), mo.group(4)))
```

```
proto="https" domain="www.gutenberg.org" port="" path="/files/11/11-h/11-h.htm"
proto="https" domain="golang.org" port="" path=""
proto="http" domain="www.google.com" port=":80" path="/"
```

Hungry for more?

https://github.com/aslakjohansen/enigma-python-intro

Questions and Comments?



https://openclipart.org/detail/236687/boy-thinking-of-question