Python for Data Science

Pre-school Lecture Series
(HEPAP-DAS, EHEP2024)

October 31, 2023 - Lecture 2

Subir Sarkar, SINP
subir.sarkar@cern.ch

Built-in Data Types

Data type	Example
Numbers	3.1415, 1234, 999L, 3+4j
String	“Hello”, “guido v”
List	[1, [2,'three'], 4]
Dictionary	{'compiled': 'Fortran,C,C++', 'interpreted': 'Perl, Python, Ruby'}
Tuple	1,'world!',3,4 or (1,'world!', 2, 3)
File	f = open('myfile.dat', 'r').readlines()

Iteration over collection

```
# list
for element in [1, 2, 3]:
  print(element, end = " ")

# tuple
for element in (1, 2, 3):
  print(element, end = " ")

# dictionary
for key in {'one':1, 'two':2}:
  print(key, end = " ")

# character string
for char in "123":
  print(char, end = " ")

# file
for line in open("myfile.txt"):
  print(line, end = " ")
```

Module/Package/Namespace

- A Python module is a file containing Python code
- A package is a collection of modules with a common purpose
- A project usually integrates different packages
- Python namespace is a combination of modules/packages

Module

Let us try to model a simple bank account

```
# bacc.py
def account():
    return {'balance': 0} # dictionary

def deposit(account, amount):
    account['balance'] += amount
    return account['balance']

def withdraw(account, amount):
    account['balance'] -= amount
    return account['balance']
```

http://anandology.com/python-practice-book/object_oriented_programming.html

A module in use

```
>>> from bacc import
account,deposit,withdraw
>>> a = account()
>>> b = account()
>>> deposit(a, 100)
100
>>> deposit(b, 50)
50
>>> withdraw(b, 10)
40
>>> withdraw(a, 10)
90
```

A Module w/ test code

```
def account():
    return {'balance': 0} # dictionary

def deposit(account, amount):
    account['balance'] += amount
    return account['balance']

def withdraw(account, amount):
    account['balance'] -= amount
    return account['balance']

def balance(account):
    return account['balance']

if __name__ == '__main__':
    a = account()
    b = account()
    deposit(a, 1000)
    deposit(b, 500)
    withdraw(b, 100)
    withdraw(a, 100)
    print('Balance -> A/C a:', balance(a), 'A/C b:', balance(b))
```

Data type and Structure

Type System

- static vs dynamic
  - type checking at compile time vs leaving the responsibility to runtime
- strong vs weak
  - how runtime treats types
  - 1 + “1” gives error for strongly typed language (e.g Python)
- explicit vs implicit
  - how type conversion takes place

```
>>> int("Hello")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: 'Hello'
>>> int('5')
5
>>> float('5.2')
5.2
```

Datatype conversion

int(x [,base]) - converts x to an integer. Base specifies the base if x is a string
long(x [,base]) - converts x to a long integer. Base specifies the base if x is a string
float(x) - converts x to a floating-point number
complex(real [,imag]) - creates a complex number
str(x) - converts object x to a string representation
repr(x) - converts object x to an expression string
eval(str) - evaluates a string and returns an object
tuple(s) - converts s to a tuple
list(s) - converts s to a list
set(s) - converts s to a set
dict(d) - creates a dictionary. d must be a sequence of (key,value) tuples
chr(x) - converts an integer to a character
ord(x) - converts a single character to its integer value
hex(x) - converts an integer to a hexadecimal string
oct(x) - converts an integer to an octal string

True or False

- The empty string is mapped to `False`, every other string is mapped to `True`

```
>>> s = ''
>>> if s:
...     print('hello')
...
>>> s = 'rob'
>>> if s:
...     print('hello')
...
hello
```

- For integers, 0 is mapped to `False` and every other value to `True`
- For floating point numbers, 0.0 is mapped to `False` and every other value to `True`

```
>>> a = 1
>>> if a:
...     print('hello')
...
hello
```

int type

```
>>> a,b = 1,2
>>> type(a)
<class 'int'>
>>> type(b)
<class 'int'>
>>> c = a + b
>>> d = a.__add__(b)
>>> c,d
(3, 3)
>>> type(c), type(d)
(<class 'int'>, <class 'int'>)
```

Control flow and loop

Control Flow

```
x = int(input("Enter an integer: "))

if x < 0:
   x = 0
   print('Negative changed to zero')
elif x == 0:
   print('Zero')
elif x == 1:
   print('One')
else:
   print('More than one')
```

standard input

- Note
  - if-elif-else construct slightly unconventional
  - indentation is mandatory

pass statements

The pass statement does nothing. It can be used when a statement is required syntactically but the program requires no action

```
>>> while True:   # infinite loop
...    pass
...
```

```
i = int(input("Enter an integer (>=0): "))
if i > 0:
  pass   # place-holder for future code
else:
  print("i is zero")
```

Loop: for & while

```
>>> list(range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> for i in range(10):
...     print(i, end = " ")
...
0 1 2 3 4 5 6 7 8 9
```

for

```
>>> index = 0
>>> while index < 10:
...     print(index, end = " ")
...     index += 1
```

while

Loop - continue & break

```
for i in range(10):
    if i%2 == 0: continue
    print(i, end = " ")
```

```
for i in range(10):
    if i**2 > 25: break
    print(i, end = " ")
```

pass by reference

```
def square(items):
    for i,x in enumerate(items):
        items[i] = x * x # modify items in-place

a = [1, 2, 3, 4, 5]
print(a)
square(a) # changes a to [1, 4, 9, 16, 25]
print(a)
```

List

List

- A list is a sequence of values that could be of any type
- The values in a list are called elements or items
- A list is mutable

List creation:

```
L = []                          # empty list
L = list()                      # empty list
L = list(sequence)
L = list(expression for variable in sequence)

# functional programming
L1 = map(function, L)
L2 = filter(function, L)

# (List) Comprehension
L = [expression, ...] # list display
L = [expression for variable in sequence]
```