Python for Data Science

IISER-Kolkata ML4HEP Pre-school Lecture Series
May 12, 2025 - Lecture 2

Subir Sarkar, SINP
subir.sarkar@cern.ch

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)
```

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 = " ")
```

Data type and Structure

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

Functions, Functional Programming Support

lambda, map, filter, reduce, list comprehension

lambda

- Python *lambda* supports creation of anonymous functions at runtime

```
# standard function call
def f(x):
   return x*2
print f(3) -> 6

# unnamed function assigned to a variable
g = lambda x: x*2
print(g(3)) -> 6

# all on-the-fly
print((lambda x: x*2)(3))
```

lambda

```
# two variables
>>> (lambda x, y: x*y)(3, 4)

# return a function
>>> def increment(n):
...     return lambda x: x + n
...
>>> increment(2)
>>> increment(2)(20)
```

Operation on sequence

```
>>> lang = ['Tcl', 'Perl', 'Python',
'Ruby', 'R', 'Julia']

# store string length in another list
>>> lst = []

>>> for e in lang:
...     lst.append(len(e))
...

>>> lst
[3, 4, 6, 4, 1, 5]
```

map (function , sequence , [sequence...]) → list

```
>>> list(map(len, lang)) # built-in
function
[3, 4, 6, 4, 1, 5]

>>> def to_upper(s): # user function
...     return s.upper()
...
>>> map(to_upper, lang)
['TCL', 'PERL', 'PYTHON', 'RUBY', 'R',
'JULIA']
```

map(function , sequence , [sequence...]) → list

```
# combine lambda() & map()
>>> list(map(lambda s: s.upper(), lang))
['TCL', 'PERL', 'PYTHON', 'RUBY', 'R',
'JULIA']

# operation on > 1 lists
>>> a = [ 1,  2,  3,  4]
>>> b = [11, 12, 13, 14]

>>> list(map(lambda x,y: x+y, a,b))
[12, 14, 16, 18]


>>> list(filter(lambda s:
s.startswith('P'), lang))
['Perl', 'Python']
```

filter(function or None, sequence) -> list, tuple, or string

```
>>> foo = list(range(20))
>>> filter(lambda x: x % 3 == 0, foo)
[0, 3, 6, 9, 12, 15, 18]

>>> map(lambda x: 2*x, \
    filter(lambda x: x%3 == 0, foo))
[0, 6, 12, 18, 24, 30, 36]
```

reduce(function, sequence[, initial]) -> value

```
>>> foo = list(range(20))
>>> reduce(lambda x, y: x + y, foo)
190
>>> reduce(lambda x,y: x+y,
    map(lambda x:2*x,
    filter(lambda x: x%3 == 0, foo)))
126

>>> def comp(a, b):
...   return a if (a > b) else b
...
>>> comp(2,4) # 4
>>> li = [1001, 9, 301, 450, 25]
>>> reduce(comp, li) # 1001
>>> reduce(lambda a,b: a if (a > b) else b, li)
1001
```

List Comprehension

```
>>> lang = ['Tcl', 'Perl', \
'Python', 'Ruby', 'R','Julia']

>>> [len(l) for l in lang]
[3, 4, 6, 4, 1, 5]

>>> [l.upper() for l in lang]
['TCL', 'PERL', 'PYTHON', 'RUBY',
'R', 'JULIA']
```

List Comprehension

```
import pprint
sentence = '''
Truth can be stated in a thousand different
ways, yet each one can be true.
'''
words = sentence.split()

print ">>> Use map ..."
newlist = map(lambda w: [w,w.upper(),len(w)], words)
pprint.pprint(newlist, depth=2)

print ">>> Use List Comprehension ..."
newlist_2 = [[w,w.upper(),len(w)] for w in words]
pprint.pprint(newlist_2, depth=2)
```

List Comprehension

```
# nested LC
noprimes = [j for i in range(2, 8)
            for j in range(i*2, 50, i)]
print(noprimes)

primes = [x for x in range(2, 50)
          if x not in noprimes]
print(primes)
```

Dictionary Comprehension

```
server = {'OS': 'CentOS 7',
          'IP': '10.10.0.244',
    'hostname': 'ccs2',
    'location': 'Room 237',
       'model': '4451',
         'OEM': 'Fujitsu'
}
print(server)
server_lo = {key.lower(): value for
key, value in server.items()}
print(server_lo)
```

Set Comprehension

```
>>> lst = [1,2,3,4,4,5,6,6,6,7,7,8,8]
>>> sa = {v for v in lst if v % 2 == 0}
>>> sa
{2, 4, 6, 8}

>>> a = {x for x in 'abracadabra' if x
not in 'abc'}
>>> a
{'r', 'd'}
```