LLP109 - Digital Application Development

Introduction to python™

- Control flow & Conditional

Dr. Hyun Lim

h.lim@lboro.ac.uk

Institute for Digital Technologies

Loughborough University London

Summary

- Data structures: ***string, tuple, list***
- Methods for a list: ***.append()***, ***.extend()***, ***.insert()***
- ~~*zip()*~~ is **not** defined for **Python 3.x**, but **Python 2.x**
- A single element ***tuple*** & ***list***:
  - a_tp1 = (1,) # *cf.* a_tp2 = (1) # It is not a tuple, # but a single variable
  - a_lst = [1] # okay
- How to use a **compiler** mode
- Study Exercises **on your own**

Questions?

Contents

- Introduction
- Data Types/Structure
- Control flow & conditional

A **compiler** mode can be more useful to execute multiple lines of codes, instead of the interactive mode:

Go to **Python** > Run **IDLE**

File > **New file** > (write your code) > **Save as** > xxxx.py > **F5** (run a program)

True/ False/ not

decision

Things that are **False**

- The boolean value: ***False***
- The number zero: **0**
- The empty string **""**, tuple **()**, list **[]**

Things that are **True**

- The boolean value: ***True***
- All non-zero numbers
- A non-empty data structure

not True : ***False***, ***not False*** : ***True***

Conditional - *if*

```
>>>A = "Hello"    # non-empty string: True
>>>bool(A)
True
>>>not bool(A)
False

# if (boolean type condition) : (intended actions)
>>>if not A :       # not A, i.e. 'empty' is False
    print ("A is empty")  # If the condition is True, do it.
    # Python is indentation-sensitive

>>>if A : print ("A is '%s' " %A)
else: print ("A is empty")      # else: is optional

# It can be useful for the coursework.
```

elif to chain subsequent conditions

```
>>> mode = 'f'          # initialise a variable - str
>>> if mode == 'a':
 x = 'a'
elif mode == 'b':       # when the previous condition is False,
 x = 'b'                # and the current condition is True
elif mode == 'c':
 x = 'c'
else:                   # when all the previous conditions are False
 print ('unknown mode')
```

Boolean comparison (<, >, ==, !), logical (*and*, *or*) operators

```
>>> a=4; b=15
>>> if (a < 5 and b >= 10 or b == 500 and a != 5):
 print ('excellent!')

>>> time=15
>>> if 9 <= time <= 17:
 print ('Office hour')
```

Loop - *for*

# executes a set of statements in the loop, once for each item in a data structure. ***for*** in Python is a kind of ***iterator***.

```
>>>names = ["James", "Chen", "Kim", "Sergey"]        # a list

>>> for x in (0,1,2,3): print(names[x])

        # x = 3, print(names[3])

>>>for x in names :        # for loop with elements of a list
    print(x)               # Python is indentation-sensitive
James      # x=names[0], print(names[0])
Chen       # x=names[1], print(names[1])
Kim        # ...
Sergey     # x=names[3], print(names[3])
```

Tuple assignment in a *for* loop

```
>>>data = [ ("C20", 308) ,
            ("C22", 316),
            ("C24", 416),
            ("C14", 311),
            ("C15", 232) ]
```

# (, ,) is a *tuple*

# [, ,] is a *list*

```
>>>for (x, y) in data:
      print ("The molecular weight of %s is %d" %(x, y))
         # %s is a type specifier for strings, %d for decimal numbers.
```

```
The molecular weight of C20 is 308
The molecular weight of C22 is 316
The molecular weight of C24 is 416
The molecular weight of C14 is 311
The molecular weight of C15 is 232
```

```
>>>for x in data:    # cf.
      print (x)
```

Summary
- *Control flow & conditional*

- **Comparison** (**<**, **>**, **==**, **!**, **!=**), **logical** (**and**, **or**) operators
- **Conditional**

  if (True/False)**:** **elif** (True/False) **:** **elif**(True/False) **:** …… **else:**
- **Loop**

  for x **in** (0,1,2,3)**:**
- Python is **indentation**-sensitive
- Use the **compiler mode**
- Study Exercises on your own and try to **put them together** logically >> **Coursework**

Discussion

- ***.sort( )*** method does not support the nested list **<u>including both lists and strings</u>** anymore after Python 3.6.4, but .reverse( ). Refer to **pp_30_32_33.py**

break

```
for x in [3, 1, 4, 1, 5, 9, 2]:
  print ("Checking", x)
  if x > 8:
    print ("Exit from the loop")
    break
  elif x < 3:
    print ("Ignoring")
  else:
    print("The square is", x**2)
```

break terminates the *for*-loop

```
Checking 3
The square is 9
Checking 1
Ignoring
Checking 4
The square is 16
Checking 1
Ignoring
Checking 5
The square is 25
Checking 9
Exiting for loop
```

# x**2 : x*x (square)
# Be careful, **^2** is not 'square' in Python

range

- '***range***' creates **a sequence** of numbers in a regular interval in a specified range between the *start* and *stop***-1**

range(*start*, *stop*, *step*)

When step is given, it specifies the increment (or decrement), default step: 1

```
>>>range(5)                                                     default start: 0
      # creates a sequence of numbers from 0 to 4 (stop-1), default step: 1
      # 0, 1, 2, 3, 4

>>>range(1, 10, 2)
      # from 1 to 9 (stop-1), step: 2
      # 1, 3, 5, 7, 9

>>>>range(1, 10, 2)[0]        # print(range(1, 10, 2)[0])
>>>>range(1, 10, 2)[1]        # [ ] calls an element from a data set
>>>>range(1, 10, 2)[-1]
```

- '*range*' creates an *immutable* sequence **integer number array**. It is neither a tuple, nor a string, but an int number array.

- Test 1

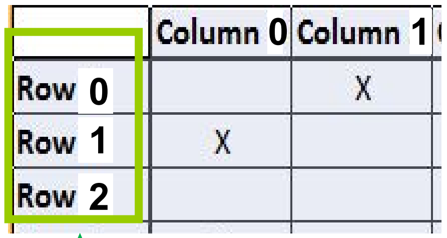
```
names = ["James0", "Chen1", "Kim2", "Sergey3"]
for x in (1,3,2,0): print(names[x])
```

- Test 2

```
data=[("C00",100) ,("C01",101),("C02",102),("C03",103),("C04",104) ]

for x in (2,4,1,3,0):
 print ("The molecular weight of %s is %d" (data[x][0], data[x][1]))
```

Exercise

```
a_lst = [['author0', 'title0'], ['author1', 'title1'], ['author2', 'title2']]
number = 3   # How can you calculate the total number of items?  len(a_lst)

# When we have a lot of books, such as for i in (0,1,2,....,10000) ??
# If we want to print the details of a_lst[0] ~ a_lst[number]

for i in range(number):                 # range(number): 0, 1, ... , number-1
 print('Author:',a_lst[i][0])           # cf.   for i in (0,1,2):
 print('Title:',a_lst[i][1],'\n')        # 2D array [row no.][column no.]
```



	Column 0	Column 1
Row 0		X
Row 1	X	
Row 2		

Each item number

Summary

- *if* - *elif* - *else* & *break*
- *for* loop with *range*()
- Use the **compiler** mode

Questions?

User-defined Function

Template

The **heading** contains ***def***, the name of the function, parentheses **()**, and a colon **:**

```
def my_func_name():
    <your statements 1>
    <your statements 2>
    ……
```

The function **body** is **indented** by a consistent amount.

```
def my_func_name(inputs):
    <your statements with the inputs>
    outputs = ……
    return outputs
```

Exercise

```
a_lst=[('John','Book I'),('Alice','Book II'),('Jamie','Book III')]

#Show function
def my_show(i):
    print('Display book information\n')
    print('Nr[', i ,']')
    print('Author:',(a_lst[i])[0])    # (a_lst[i])[0]) == a_lst[i][0]
    print('Title:', a_lst[i][1],'\n')

my_show(0)
my_show(1)
```

Data (Keyboard) *input*

```
>>>author=input("Insert Author: ")
Insert Author: John
>>>author
'John'
```

```
>>>year=input("Insert Year: ")
Insert Year: 2020
>>>year
'2020'
```

# An ***Input*** value is a (character) *string*.

```
>>>if year=='2020': print('Year is ',year)
(Does it work?)
```

Exercise   # Design your insert function including **user-inputs**, such as

```
author = input("Insert Author: ")
title = input("Insert Title: ")
   ...
i_lst=[author, title, ...]
all_lst=[i_lst[0], i_lst[1], ..., i_lst[-1]]    # nested lists
```

Loop - *while*

```
while condition == True:
  statement 1
  statement 2
  ….
```

# A loop will be repeated while the condition is **True**. If the condition is **False** or the process meets **break**. it gets out of the loop.

Exercise

Design your loop using ***while*** and ***if*** conditional, such as:

```
number=0      # initialisation
while True:    # It makes an infinite loop. Or, give just any number: non-zero value.
    index=input('Choose one of 1~3:\n Increase Nr of items[1], The Nr of items[2], End[3]:')
    if index=='1':
        number=number+1
        print('You have increased the number of items.\n')
    elif index=='2':
        print('There are %d item(s).\n' %number)   # print('There are ',number, 'items.\n')
    elif index=='3':
       print('Goodbye~')
       break
    else:
     print('Your index=',index,'\n You've made a mistake! Put an appropriate number (1, 2, 3).\n')
```

Modules

Modules

- When a Python program starts it only has access to basic functions and classes, such as:

  int(), len(), print(), range(), sum(), .append()
- ***Modules*** contain additional functionality.
- Use "***import***" to ask Python to load a specific module, e.g. math:

```
>>> import math
```

✓ If other external modules have been installed, e.g. Natural Language Toolkit (**NLTK**) and **Pandas**, we can also import them as follows:

```
>>> import nltk
>>> import pandas as ps
```

math module

```
>>> import math
>>> math.pi       #  print(math.pi)
3.141592653589793
>>> math.cos(0) #  print(math.cos(0))
1.0
>>> math.cos(math.pi/3)
0.5
>>> dir(math) #  print(dir(math))
['__doc__', '__file__', '__name__', '__package__', 'acos', 'acosh',
'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copysign', 'cos',
'cosh', 'degrees', 'e', 'exp', 'fabs', 'factorial', 'floor', 'fmod',
'frexp', 'fsum', 'hypot', 'isinf', 'isnan', 'ldexp', 'log', 'log10',
'log1p', 'modf', 'pi', 'pow', 'radians', 'sin', 'sinh', 'sqrt', 'tan',
'tanh', 'trunc']
>>> help(math)
>>> help(math.cos)
```

import

```
>>> import math
>>> math.pi
3.141592653589793

>>> from math import *
>>> pi
3.141592653589793
>>> sin(pi/2)
1.0
```

Useful (external) Python Modules

- You can download and install:

  Numpy, Scipy, Pandas and **Matplotlib**

- **Pandas** provides an easy-to-use abstraction over both *Numpy* and *matplotlib*. Easier for beginners to use.

- **TensorFlow** or **PyTorch -** for *Deep Learning*
  - **PyTorch**: popular in research and industry as well
  - **TensorFlow:** easier to deploy on other devices (Android, smaller embedded systems etc)

Example

To install *Pandas* on Windows PC

- Open ***Commend Prompt*** (Press [Windows]+*R* kyes)

  > type ***cmd*** and press ***OK***)

```
C:\users\Admin>py -m pip install pandas --user
```

- If it does not work, go to the …\Python\Scripts folder, such as

```
C:\users\Admin>cd C:\Python36\Scripts
C:\Python36\Scripts>py -m pip install pandas --user
```

- ***Python*** may take place at different folders on your computer, e.g.

```
C:\Program Files (x86)\Microsoft Visual Studio\Shared\Python36_64\Scripts
C:\Users\Admin\AppData\Local\Programs\Python\Python37\Scripts
C:\Python36\Scripts
```

- Test on Python (interactive mode)

  >>> import ***pandas*** as ps

Coursework

- The coursework description is available in Coursework Section on LEARN:
  - **Basic requirements**: Use Data Structures (list, tuple), Loop, Conditionals, user defined Functions, and Data (keyboard) Input/ (screen) Output.
  - **In addition**, create your own additional functionalities, user-interface including various options, **error management functions**.
  - Similar (or a bit different) results, but lots of different approaches/ strategies are available. Creative/ different/ brave attempts will be well appreciated.
  - **Don't need to develop GUI**, such as ~~*tkinter*~~ in Python for the coursework .

Coursework

- Before submission, test your app enough **in other PCs** and **make sure it works** as it should.

  - Coursework 1: **Draft report** about your program
  - Coursework 2: a) **Program (**Python *or* C**)**
    b) **Final report**

  } Zip all together as a single .zip file