SIT 112 - Data Science Concepts

Lecturer: Sergiy Shelyag | sergiy.shelyag@deakin.edu.au
Assistant:

School of Information Technology,
Deakin University, VIC 3216, Australia.

Practical Session 1.4

The purpose of this session is twofold:

1. to give you the instruction to run Jupyter Notebooks
2. to revise key features of Python language and allow you the time to practice basic coding in python with Jupyter Notebooks

At the end of this practical session, you should be able to:

- know how to start an Jupyter Notebook server
- know how to create a Jupyter notebook in a web browser and execute the commands in the notebook
- know the basic features of python, including comparison and control structure

Instructions

1. Create a new notebook, name it as `"[yourstudentID]_prac1_sol.ipynb"`
2. As we walk through this practical session, you will be required to write your own code in the cells in your notebook. For each exercise, insert a new cell and enter your own code.
3. Although it is NOT marked, you are strongly recommended to keep these saved notebooks for your future reference.
4. The solution notebook will be available for download within 4 days after the practical session ends.

Please note that:

- This unit is *not* about teaching you how to program in Python. You are responsible for developing your own programming skills. There are many excellent resources to learn python online as well as the recommended textbooks presented in the lecture. Developing skills in programming can

take time, and let us remind ourselves:

> "... even though we can speak and write English, it does not necessarily mean we can write a best-selling novel" -- Unknown.

- You are assumed to have a basic knowledge of programming.
- This session only recaps main features of python.

Part I: Jupyter Notebook

- 3-Jupyter-Notebook (3-Jupyter-Notebook.ipynb) gives you the step by step instructions on how to start Jupyter Notebook server and how use notebooks.

Part II: Review of Python Language (continue)

---

1. Compound Types

1.1 List

List is the most versatile datatype available in Python which can be written as a sequence of comma-separated values (items) between square brackets. Important thing about a list is that items in a list need not be of the same type.

The syntax for creating lists in Python is `my_list = [item_0, item_1, ..., item_n]`:

In [ ]:
```
l = [10, 2, 30, 4, 50, 6, 70, 8]
print(type(l))
print(l)
print(len(l))
```

Elements of a list do not have to be the same type.

In [ ]:
```
l1 = [1, 'a', 1.0, 1-1j]
l2 = [1, 2, [3, 4], 5]

print(l1)
print(l2)
```

1.1.1 List indexing

You can index an item from a list using `[]` operator. `my_list[i]` returns the ith item of `my_list`:

In [ ]:
```
# indexing
l = [1, 2, 3, 4, 5, 6, 7, 8]

print(l[0])
print(l[3])
print(l[-1])    # last element
print(l[-2])    # can you guess the result?
```

Heads up MATLAB users: Indexing start at 0!

Exercise: define a list with multiple items of different types. Let the last item be another list equal to `['apple', 'orange', 'banana', 'pear']`. How can you index `'banana'`? Use list indexing and print it.

1.2 String

Strings are used for storing text messages. They are very similar to lists, lists of characters.

In [ ]:
```
s = "Hello world!"
print(type(s))

# length of the string: the number of characters
print(len(s))
```

You can index a character in a string using `[]` similar to lists.

In [ ]:
```
# indexing
s = "Hello world!"

print(s[3])
print(s[0])
```

Exercise: define a string with value equal to `"Python is fun!"`. How can you index the character `'s'`?

1.2.1 List/string slicing

You can extract a part of a list/string using the syntax `[start:stop]`, which extracts characters between index `start` and `stop`.

In [ ]:
```
# slicing
l = [1, 2, 3, 4, 5, 6, 7, 8]
s = "This is a string."

print(l[2:5])
print(s[0:6])
```

You can also define a step for slicing as in `[start:stop:step]`:

In [ ]:
```
# [start:stop:step]
print(s[2:10:1])
print()

# These two are equal
print(s[0:10:2])
print(s[:10:2])
```

`Slicing Step` can be a negative value as well:

In [ ]:
```
print(s[10:2:-1])
print()

# reversing a list or string
print(s[::-1])
```

Exercise: define a string with value equal to `"Python is fun!"`. How can you print every third character of it?

1.3 Tuples

Tuples are like lists, except that they cannot be modified once created, that is they are *immutable*.

In Python, tuples are created using the syntax `(..., ..., ...)`, or even `..., ...`:

In [ ]:
```
# TUPLE
# A tuple is a *fixed* list. We can't change its values

states = ('VIC', 'NSW', 'QLD', 'WA', 'ACT', 'NT', 'TAS', 'SA')
print("type of states: {}".format(type(states)))

print("Australian states")
for i in range(0,7):
    print(states[i])
```

Exercise: create a list with items as days of the week and call it `'my_list'`. Now create a tuple with items as days of the week and call it `'my_tuple'`. Alter values of `'my_list'` such that instead of Monday, you have 1. Can you do the same thing with `'my_tuple'`?

1.4 Dictionaries

Dictionaries are similar to lists, except that each element is a key-value pair. The syntax for dictionaries is `{key1 : value1, ...}`. Note that the values can be of any type, even another dictionary.

In [ ]:
```
params = {"parameter1" : 1.0,
          "parameter2" : "a string",
          "parameter3" : [1,2,3]}

print(type(params))
print(params)
```

You can access and manipulate the values of a dictionary using its keys.

In [ ]:
```
my_dictionary = {"a":2,
                 "python": "a programming language",
                 "phone": "a communication device"}

# access value of a dictionary dictionary['key']
print("python is", my_dictionary["python"])
print("phone is", my_dictionary["phone"])

# remove an element
del my_dictionary["phone"]
print(my_dictionary)

# clear the dictionary
my_dictionary.clear()
print(my_dictionary)
```

Exercise: create a dictionary called `'my_dict'` with keys `'first_name, last_name, age, favorite_singer'` and fill the values. Now alter the

value of `'favorite_singer'` and change it to `'Justin Bieber'`!

Can you add a key `'seriously'` and fill the value with a boolean?

---

5 Control flow

5.1 Conditional statements: if, elif, else

The Python syntax for conditional execution of code use the keywords `if`, `elif` (else if), `else`. Python code blocks are defined by their indentation level.

In [ ]:
```
statement1 = False
statement2 = False

if statement1:
    print("statement1 is True")

elif statement2:
    print("statement2 is True")

else:
    print("statement1 and statement2 are False")
```

In [ ]:
```
# nothing happens
statement1 = False

if statement1:
    print("printed if statement1 is True")

    print("still inside the if block")
```

In [ ]:
```
if statement1:
    print("printed if statement1 is True")

print("now outside the if block")
```

Exercise: Write a piece of code that declares two variables called `'is_raining'` and `'is_sunny'`, and assign a Boolean value to each of them. If both variables are True then print `'Is there a rainbow?'`.