SIT 112 - Data Science Concepts

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

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

Practical Session 2.2

The purpose of this session is to revise more advanced features of Python language, including loops and function declarations, and allow you the time to practice.

1. Loops

In Python, loops can be programmed in a number of different ways. The most common is the `'for'` loop, which is used together with iterable objects like lists.

1.1 For loops

The `for` loop has the ability to iterate over the items of any sequence, such as a list or a string, and repeat executing a statement or a group of statements

In [9]:
```
list1 = [1, 2, 3, 4, 5]
for item in list1:   # iterate over the items of the list
    print(item)  # do something
```

```
1
2
3
4
5
```

In [10]:
```
# range(n) is a list of n elements [0, 1,..., n-1]
for x in range(4): # by default range start at 0
    # At each step, the statement or a group of statements is executed
    print(x)
    print(x**2)
```

```
0
0
1
1
2
4
3
9
```

In [11]:
```
course = ["an", "introduction", "to", "data", "science"]
for word in course:
    print(word)
```

```
an
introduction
to
data
science
```

In [12]:
```
# use enumerate to access a list items and their indices

course = ["an", "introduction", "to", "data", "science"]
for idx, x in enumerate(course):
    print(idx, x)
```

```
0 an
1 introduction
2 to
3 data
4 science
```

Exercise: Declare a variable named `my_var` and assign an integer to it. Print all even numbers that are smaller `my_var`

In [13]:
```
# your code here
my_var = 20
for x in range(my_var):
    if (x % 2 == 0):
        print(x)
```

```
0
2
4
6
8
10
12
14
16
18
```

1.2 List comprehension

A convenient and compact way to initialize lists:

In [14]:
```
# range(m, n) is a list: [m, m+1,...,n-1]
list2 = [x**2 for x in range(2, 6)]

print(list2)
```

```
[4, 9, 16, 25]
```

Exercise: Create a list and fill it with integers from 1 to 50. Now iterate over items of the list and if the item is divisible by 3 and not divisible by 2, print it on the screen.

In [15]:
```
# your code here
new_list = [i for i in range(1, 51)]
for x in new_list:
    if x % 3 == 0 and (x % 2 != 0):
        print(x)
```

```
3
9
15
21
27
33
39
45
```

Exercise: Create a list of integers and print the sum of all elements.

In [16]:
```
# your code here
int_list = [1, 5, 2, 10, 7, 20]
sum_value = 0
for x in int_list:
    sum_value += x

print('The sum of the list is:', sum_value)
```

```
The sum of the list is: 45
```

1.3 While loops

While loops repeat a statement or group of statements while a given condition is **True**. It tests the condition before executing the loop body.

In [17]:
```
i = 0

while i < 5: # First, the loop test if the condition "i < 5" is True.
    # end=' ' means that a space will be printed
    # ...after each number i instead of a line break
    print(i, end=' ')

    i += 1

print("done")
```

```
0 1 2 3 4 done
```

Note that the `print("done")` statement is a not part of the `while` loop body because of the difference in indentation.

Exercise: Define an integer variable named `limit`. Print the largest square number that is smaller than `limit`. For example, if `limit` is 40 then print `36`.

In [18]:
```
# your code here

limit = 40
i = 0
while i ** 2 < limit:
    i += 1

# now i * i is the bigger than or equal to limit
# we need (i - 1) ** 2
print('The largest square number smaller than limit is:', (i - 1) ** 2)
```

```
The largest square number smaller than limit is: 36
```

1.4 Nested loops

In Python, a loop can appear inside another loop. Execute the cell below:

In [19]:
```
sentence = ['Welcome', 'to', 'SIT112']
for word in sentence:
    for character in word:
        print(character, end=' ')
    print()
```

```
W e l c o m e
t o
S I T 1 1 2
```

---

2. Functions

A function in Python is defined using the keyword `def`, followed by a function name, one or several arguments within parentheses `()`, and a colon `:`. The following code, **with one additional level of indentation**, is the function body. It is recommended to use "docstring" for your functions.

In [20]:
```
def func1(s):
    """
    THIS BIT IS CALLED DOCSTRING
    s is the paramter of the function
    Print a string 's' and tell how many characters it has
    """

    print(s + " has " + str(len(s)) + " characters")
```

In [21]:
```
func1("test")
```

```
test has 4 characters
```

A function can have multiple arguments. For example, the function computes the sum of two numbers:

In [22]:
```
def sum_of_2_numbers(a, b):
    """
    This function reads 2 numbers a and b and print their sum
    """
    print(a + b)
```

Use the `return` keyword for functions that return a value. A function can also return multiple values using tuples.

In [23]:
```
def powers(x):
    """
    Return a few powers of x.
    """
    return x**2, x**3, x**4
```

In [24]:
```
print(powers(3))

x1, x2, x3 = powers(3)
print(x2)
```

```
(9, 27, 81)
27
```

Exercise:

1. Write a function that takes a list of numbers and prints elements of it that are divisible by 3 and not divisible by 2.
2. Create a a list and fill it with integers from 1 to 50. Use the function you have written in the previous step to print out items of the list that are divisible by 3 and not divisible by 2.

In [25]:
```
# your code here
def print_numbers(num_list):
    """
    Print all numbers in list num_list that are divisible by 3 and not divisible by 2.
    """
    for x in num_list:
        if x % 3 == 0 and (x % 2 != 0):
            print(x)

# create a list
new_list = [i for i in range(1, 51)]
print_numbers(new_list)
```

```
3
9
15
21
27
33
39
45
```

Exercise: Write a function named `largest_square` that takes a variable `limit` and prints the largest square number that smaller than `limit`.

In [26]:
```
# your code here
def largest_square(limit):
    """
    Print the largest square number that smaller than 'limit'
    """
    i = 0
    while i ** 2 < limit:
        i += 1
    print((i - 1) ** 2)
################
# Then run this cell to see if your function is correct
largest_square(35)
largest_square(70)
```

```
25
64
```

2.1 Default Argument Values

Python allows a function to have a default value for one or more arguments. This creates a function that can be called with fewer arguments than it is defined to allow. For example:

In [27]:
```
def weather(is_sunny, complain='How bad it is!'):
    if not is_sunny:
        print(complain)

weather(False)
weather(False, 'I miss the sun!')
```

```
How bad it is!
I miss the sun!
```

You might notice that in section While loops, the `print` function has an argument `end=' '` while `print` in other examples does not. It is because `end` is an argument with '\n' (this string indicates a line break) as the default value. That means that the `print` function always prints a line break except when another string is passed. Execute the cell below and see what happend.

In [28]:
```
print('A', 'B', 'C', end='_')
print('a', 'b', 'c')

# print function has another argument called 'sep',
# ...which is used to seperate the printed items.
# The default value of 'sep' is a space.
print('A', 'B', 'C', sep='_', end=' ')
print('a', 'b', 'c', sep='')
```

```
A B C_a b c
A_B_C abc
```

2.2 Python build-in functions

The Python interpreter has a number of functions built into it. That means these functions are already defined and you can use them just like your defined functions. In the example above, function len(s) reads a string s and returns the length of s. The function str(n) returns the number n in the string type.

In [29]:
```
# the sum function returns the sum of a list
print(sum([1, 2, 3]))

# the abs function returns the absolute value of a number
print(abs(-3.2))
```

```
6
3.2
```

Exercise: See more built-in functions in Python built-in function (https://docs.python.org/2/library/functions.html) and write function named `min_max` that reads a list of numbers and returns the maximun and the minimum number in the list.

In [30]:
```
# your code here
def min_max(num_list):
    return min(num_list), max(num_list)

###############
# Then run this cell to see if your function is correct
print(min_max([3, 2, 4, 1, 10, -5]))
print(min_max([3.5, 2.7, -1.3, 6, -5.2]))
```

```
(-5, 10)
(-5.2, 6)
```

2.3 Anonymous functions

Anonymous functions are defined by the keyword `'lambda'` in Python. Functions `'f'` and `'g'` in the cell below basically do the same thing but `'g'` is an anonymous function.

In [31]:
```
def f(x):
    return x**2

g = lambda x: x**2

print(f(8), g(8))
```

```
64 64
```

Note how we used anonymous functions in the cell below to create the parametric function 'n_increment()'. Now We create new functions by passing 'n' to 'n_incremenet()'. For example 'f5' and 'f9' are functions that add 5 and 9 to their inputs respectively.

In [32]:
```
def n_increment(n):
    return lambda x: x+n

f5 = n_increment(5)
f9 = n_increment(9)

print(f5(2), f9(2))
```

```
7 11
```

3.More on Complex Data Structures

3.1 Matrix as nested lists

A matrix of size 3x4 can be implemented by a list of 3 lists of length 4. Execute the cell below:

In [33]:
```
matrix = [[0, 1, 2, 3, 4],
          [5, 6, 7, 8, 9],
          [10, 11, 12, 13, 14]]

print(matrix)

# Remember that python indices from 0
print('Element at row 1, column 2:', matrix[1][2])
```

```
[[0, 1, 2, 3, 4], [5, 6, 7, 8, 9], [10, 11, 12, 13, 14]]
Element at row 1, column 2: 7
```

The `matrix` in the example above can be created by the following way:

In [34]:
```
matrix = [[row * 4 + column for column in range(4)] for row in range(3)]

print(matrix)
```

```
[[0, 1, 2, 3], [4, 5, 6, 7], [8, 9, 10, 11]]
```

Exercise:

1. Write a function named `matrix_sum` that takes a matrix as an input and returns the sum of all elements in the matrix.
2. Declare a matrix A of size 3x3 with arbitrary values and print the `matrix_sum` of A.

In [35]:
```
# Your code here
def matrix_sum(matrix):
    # using nested for loops:
    sum_value = 0
    for row in matrix:
        for value in row:
            sum_value += value
    return sum_value

A = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
print(matrix_sum(A))
```

```
45
```

In [36]:
```
# Your code here
# Another way is using the built-in function 'sum'
def matrix_sum(matrix):
    return sum([sum(row) for row in matrix])

A = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
print(matrix_sum(A))
```

```
45
```

3.2 Built-in methods

Python includes a number of methods for complex structures such as strings, lists and dictionaries. Execute these cells below:

In [37]:
```
# Split method for string
str1 = 'I-love data science'
space_split = str1.split()
hyphen_split = str1.split('-')
print('Space splitting:', space_split)
print('Hyphen splitting:', hyphen_split)
# More on split method: https://docs.python.org/3/library/stdtypes.html#str.split

# Check if a string is a list of digits
print('123:', '123'.isdigit())
print('1a3:', '1a3'.isdigit())
print('1.2:', '1.2'.isdigit())
```

```
Space splitting: ['I-love', 'data', 'science']
Hyphen splitting: ['I', 'love data science']
123: True
1a3: False
1.2: False
```

In [8]:
```
list3 = [1, 2, 3]

# append an element to list
list3.append(4)
print(list3)

list3.append('I love data science')
print(list3)

# insert an element at an index
list3.insert(3, 'this is an insertion')
print(list3)
```

```
[1, 2, 3, 4]
[1, 2, 3, 4, 'I love data science']
[1, 2, 3, 'this is an insertion', 4, 'I love data science']
```

In [45]:
```
price_dictionary = {'bananas': 2,
                    'kiwis': 5,
                    'apples': 4,
                    'cherries': 15}

# the method returns the keywords in the dictionary
print(price_dictionary.keys())

# the method returns the values in the dictionary
print(price_dictionary.values())
```

```
dict_keys(['bananas', 'kiwis', 'apples', 'cherries'])
dict_values([2, 5, 4, 15])
```

4. Further reading

Python functions (https://docs.python.org/3.5/tutorial/controlflow.html#defining-functions) More about python functions on the official Python tutorial website.