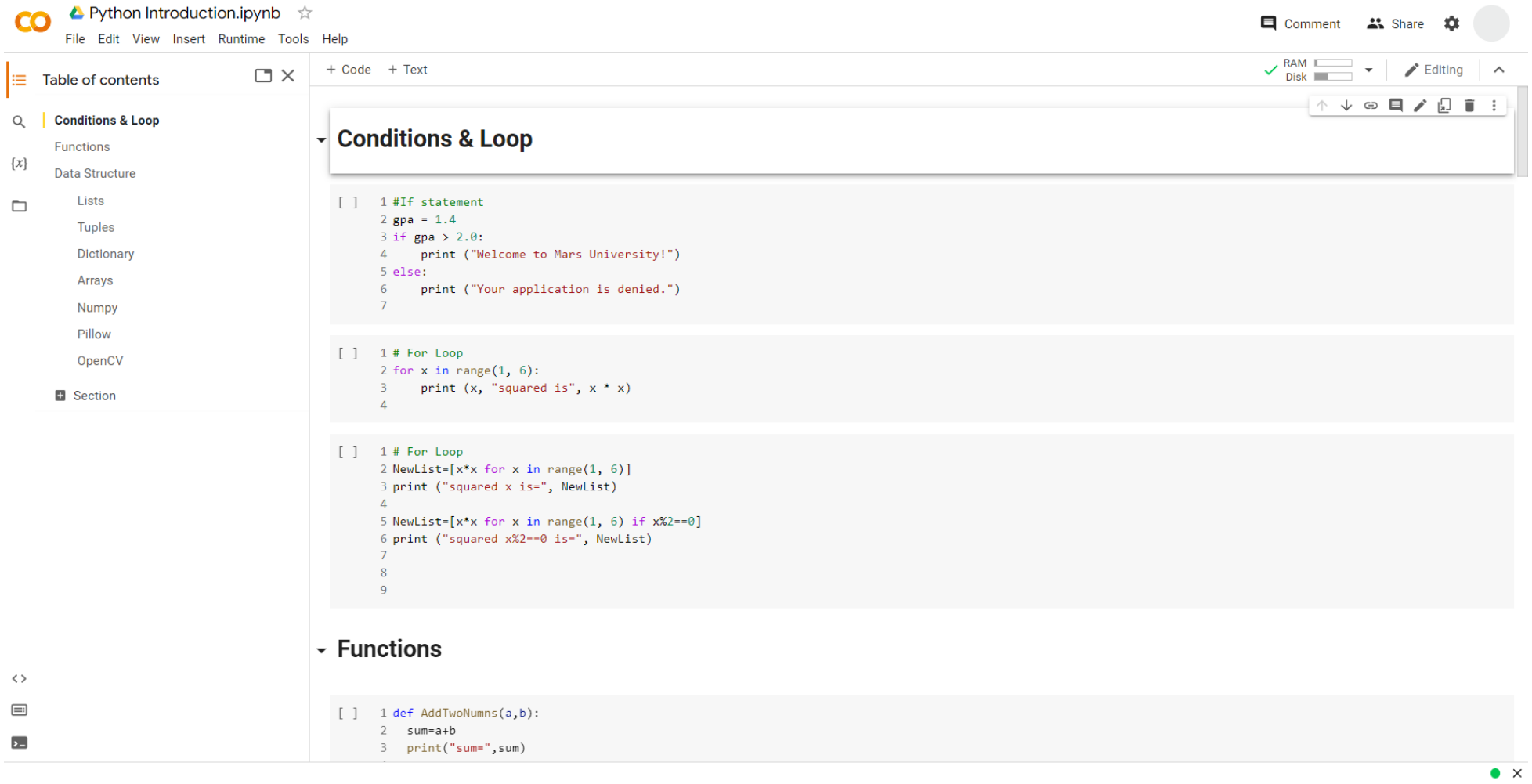
CSY3010 Media technology

Python Introduction

Google Colab



https://colab.research.google.com/

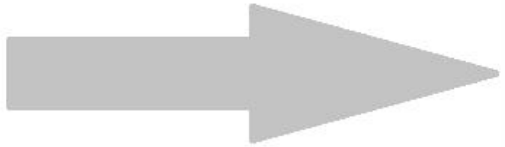
Session’s Agenda!

- Conditions
- For Loop
- Functions
- Lists
- Tuples
- Arrays
- Numpy
- Pillow
- OpenCV

Python Indentation

- Indentation refers to the spaces at the beginning of a code line.
- Indentation should be the same for each block of code.

```
Statement
if condition:
        if condition:
                Statement
        else:
                Statement
Statement
```



```
Code block 1 begins
Code block 1 continues
        Code block 2 begins
                Code block 3 begins
        Code block 2 continues
                Code block 3 continues
Code block 1 continues
```

Condition Statements

- if statement: Executes a group of statements only if a certain condition is true. Otherwise, the statements are skipped
- Syntax:

```
if condition:
    statements
elif condition:
    statements
else:
    statements
```

```
#If statement
gpa = 1.4
if gpa > 2.0:
    print ("Welcome to Mars University!")
else:
    print ("Your application is denied.")
```

```
Your application is denied.
```

For Loop

- for loop: Repeats a set of statements over a group of values.
- Syntax:

```
for variableName in groupOfValues:
    statements
```

```
# For Loop
for x in range(1, 6):
    print (x, "squared is", x * x)
```

```
1 squared is 1
2 squared is 4
3 squared is 9
4 squared is 16
5 squared is 25
```

```
for val in "string":
    if val == "i":
        continue
    print(val)
```

```
s
t
r
n
g
```

For Loop

- Single Line For Loops

```
# For Loop
NewList=[x*x for x in range(1, 6)]
print ("squared x is=", NewList)

NewList=[x*x for x in range(1, 6) if x%2==0]
print ("squared x%2==0 is=", NewList)
```

```
squared x is= [1, 4, 9, 16, 25]
squared x%2==0 is= [4, 16]
```

Functions

- Syntax

```
def function_Name(param1,param2,..):
    Statement1
    Statement2
        ...
    Statementn
    return
```

```
def AddTwoNumns(a,b):
  sum=a+b
  print("sum=",sum)

AddTwoNumns(23,12)
```

```
sum= 35
```

```
def AddTwoNums(a,b=10):
  sum=a+b
  return sum

x=AddTwoNumns(23,20)
print("sum=",x)
x=AddTwoNumns(23)
print("sum=",x)
```

```
sum= 43
sum= 33
```

Built-in Functions

- Built-in functions that are provided as part of Python - input(), type(), float(), int()…
  - print( )
  - type( )
  - input( )
  - abs( )
  - pow( )
  - sorted( )
  - max( )
  - …

Data Structures

Python comes with a general set of built in data structures:

- Strings
- Lists
- Tuples
- Dictionaries
- Arrays
- others...

Lists

- A list contains items separated by commas and enclosed within square brackets [10,17,43, “John”,24].
- List may contain different DATA TYPES!
- List acts as Strings: list can be accessed using the slice operator [ ]
  - indexes starting at 0 in the beginning of the list(example list[0])
  - -1 refers to the last item in the list (example list[-1])
  - The plus (+) sign is the list concatenation operator, and the asterisk (*) is the repetition operator(like in Strings).

Lists

- Create List

```
myList=[]

print(type(myList))
print(len(myList))
```

```
<class 'list'>
0
```

```
list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]
tinylist = [123, 'john']

print(list) # Prints complete list
print(list[0]) # Prints first element of the list
print(list[1:3]) # Prints elements starting from 2nd till 3rd
print(list[2:]) # Prints elements starting from 3rd element
print(tinylist * 2) # Prints list two times
print(list + tinylist) # Prints concatenated lists
```

```
['abcd', 786, 2.23, 'john', 70.2]
abcd
[786, 2.23]
[2.23, 'john', 70.2]
[123, 'john', 123, 'john']
['abcd', 786, 2.23, 'john', 70.2, 123, 'john']
```

Operation on Lists

- `[1, 2, 3] + [4]` ⇒ `[1, 2, 3, 4]` #Concatenate
- `[1, 2, 3] * 2` ⇒ `[1, 2, 3, 1, 2, 3]`#Repeat *
- `1 in [1, 2, 3]` ⇒ `True`#Membership (the in operator)
- `[1, 2, 3] < [1, 2, 4]` ⇒ `True`
- Slicing ([:])
- `len([1, 2, 3,11])` ⇒ 4

Pythons' Functions for List

- len(lst): number of elements in list (top level). len([1, [1, 2], 3]) ➔ 3
- min(lst): smallest element. Must all be the same type!
- max(lst): largest element, again all must be the same type
- sum(lst): sum of the elements, numeric only

Iteration

- We can iterate through the elements of a list like we did with a string

```
Items = [ 'abcd', 786 , 2.23, 'john', 70.2 ]
for item in Items:
  print(item)
```

```
abcd
786
2.23
john
70.2
```

Can you guess the output?

```
Items = [ 'abcd', 786 , 2.23, 'john', 70.2 ]
for item in Items[0]:
  print(item)
```

List functions

```
my_list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]
my_list.append(1)
print(my_list)
my_list.extend([1,3,2])
print(my_list)
x=my_list.pop()
print(x)
print(my_list)
my_list.insert(2,2)
print(my_list)
my_list.remove('abcd')
my_list.remove('john')
print(my_list)
my_list.sort()
print(my_list)
my_list.reverse()
print(my_list)
```

```
['abcd', 786, 2.23, 'john', 70.2, 1]
['abcd', 786, 2.23, 'john', 70.2, 1, 1, 3, 2]
2
['abcd', 786, 2.23, 'john', 70.2, 1, 1, 3]
['abcd', 786, 2, 2.23, 'john', 70.2, 1, 1, 3]
[786, 2, 2.23, 70.2, 1, 1, 3]
[1, 1, 2, 2.23, 3, 70.2, 786]
[786, 70.2, 3, 2.23, 2, 1, 1]
```

Split & Sort

- The string method split generates a sequence of characters by splitting the string at certain split-characters.
- It returns a list (we didn't mention that before)
- `split_list = 'this is a test'.split()`
- `split_list` ➔ `['this', 'is', 'a', 'test']`

- **Sort**
  - `my_list = list('xyzabc')`
  - `my_list` ➔ `['x','y','z','a','b','c']`
  - `my_list.sort()  # no return`
  - `my_list` ➔ `['a', 'b', 'c', 'x', 'y', 'z']`

```
split_list = 'this is a test'.split()
print(split_list)
my_list = list('xyzabc')
print(my_list)
my_list.sort()
print(my_list)
print(''.join(my_list))
```

```
['this', 'is', 'a', 'test']
['x', 'y', 'z', 'a', 'b', 'c']
['a', 'b', 'c', 'x', 'y', 'z']
abcxyz
```

Tuples

- Similar to the list but:
  - enclosed within parentheses.
  - Items can NOT be changed.
  - All list functions works on Tuples expect: append,extend, del
  - Create Tuples:
    - `myTuple = 1,2  # creates (1,2)`
    - `myTuple = (1,) # creates (1)`
    - `myTuple = (1)  # creates 1 not (1)`
    - `myTuple = 1,   # creates (1)`

```
tuple= ('abcd', 786 , 2.23, 'john', 70.2)
tinytuple= [123, 'john']

print(tuple) # Prints complete tuple
print(tuple[0]) # Prints first element of the tuple
print(tuple[1:3]) # Prints elements starting from 2nd till 3r
print(tuple[2:]) # Prints elements starting from 3rd element
print(tinytuple* 2) # Prints tupletwo times
```

```
('abcd', 786, 2.23, 'john', 70.2)
abcd
(786, 2.23)
(2.23, 'john', 70.2)
[123, 'john', 123, 'john']
```

Array

- Arrays is not fundamental data structure in Python, we have to import it:
  - `from array import *`
- Declaration:
  - `arrayName = array(Datatypecode,[Initializers])`
- Example:
  - `my_array = array('i',[1,2,3,4])`

'b' ==> Represents signed integer of size 1 byte
'B' ==> Represents unsigned integer of size 1 byte
'h' ==> Represents signed integer of size 2 bytes
'H' ==> Represents unsigned integer of size 2 bytes
'i' ==> Represents signed integer of size 2 bytes
'I' ==> Represents unsigned integer of size 2 bytes
'l' ==> Represents signed integer of size 4 bytes
'L' ==> Represents unsigned integer of size 4 bytes
'f' ==> Represents floating point of size 4 bytes
'd' ==> Represents floating point of size 8 bytes

Array

- Example

```
from array import *
my_array = array('i',[1,2,3,4])
for i in my_array:
    print(i)
my_array.append(6)
print(my_array)# Try other function? extend,insert
list=[11,12,13]
my_array.fromlist(list)
print(my_array)
list=my_array.tolist()
print(list)
```

```
1
2
3
4
array('i', [1, 2, 3, 4, 6])
array('i', [1, 2, 3, 4, 6, 11, 12, 13])
[1, 2, 3, 4, 6, 11, 12, 13]
```

Try more?:
count(),
reverse()

Numpy

- Numpy provides high-performance, multidimensional array object and tools for working with such arrays
- A NumPy array is an N-dimensional homogeneous collection of “items” of the same data type.
- Similar to lists, but much more capable, except fixed size!
- It is faster than lists, and requires less memory.
- Better arithmetic and better multidimensional tools.
- Example:

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
print(arr)
Output➔ [1 2 3 4 5]
```

Numpy

- Add two arrays

```
import numpy as np
a = np.array([1,3,5,7,9])
b = np.array([2,4,6,8,10])
c = a + b
print(c)
```

```
[ 3  7 11 15 19]
```

- 2D array

```
import numpy as np
a = np.array([[1,3,5],[7,9,2],[6,8,10]])
print(a)
```

```
[[ 1  3  5]
 [ 7  9  2]
 [ 6  8 10]]
```

Numpy

- Attributes:
  - nparray.ndim
  - nparray.shape
  - nparray.size
  - nparray.dtype
  - nparray.itemsize
  - nparray.T
  - nparray.data

```
import numpy as np
nparray = np.array([[1,3,5],[7,9,2],[6,8,10]])
print(nparray.ndim)
print(nparray.shape)
print(nparray.size)
print(nparray.dtype)
print(nparray.itemsize)
print(nparray.T)
print(nparray.data)
```

```
2
(3, 3)
9
int64
8
[[ 1  7  6]
 [ 3  9  8]
 [ 5  2 10]]
<memory at 0x7f4344e43050>
```

Numpy

- Functions
  - Sum of all the elements
  - Mean of all the elements
  - Standard deviation of all the elements
  - Maximum element
  - Minimum element

```
import numpy as np
nparray = np.array([[1,3,5],[7,9,2],[6,8,10]])
print(a.sum())
print(a.mean())
print(a.std())
print(a.max())
print(a.min())
```

```
51
5.666666666666667
2.9814239699997196
10
1
```

Numpy

- Mathematics

```
a = np.array([1,2,3], float)
b = np.array([5,2,6], float)
print("a+b=",a + b)
print("a-b=",a - b)
print("a*b=",a * b)
print("a/b=",b / a)
print("a%b=",a % b )
print("a**b=",b**a )

a = np.array([[1, 2], [3, 4], [5, 6]], float)
b = np.array([-1, 3], float)
print(a)
print(b)
print("a+b=",a + b )
```

```
a+b= [6. 4. 9.]
a-b= [-4. 0. -3.]
a*b= [ 5. 4. 18.]
a/b= [5. 1. 2.]
a%b= [1. 0. 3.]
a**b= [ 5. 4. 216.]
[[1. 2.] [3. 4.] [5.
6.]]
[-1. 3.]
a+b= [[0. 5.] [2. 7.]
[4. 9.]]
```

Numpy

- Indexing
  - You can access an array element by referring to its index number.
  - The first element has index 0

```
1  import numpy as np
2  arr = np.array([41, 27, 93, 77])
3  print("Value index  1:",arr[1])
4  print("Value index -1:",arr[-1])
```

```
Value index  1: 27
Value index -1: 77
```

```
1  arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])
2
3  print('5th element on 2nd row: ', arr[1, 4])
```

```
5th element on 2nd row:  10
```

Pillow

- Pillow is a Python Imaging Library (PIL)
- support for opening, manipulating, and saving images.
- Example:

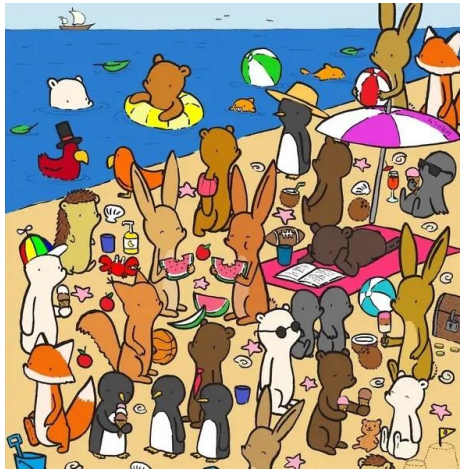
```
from PIL import Image
with Image.open('dog.jpg') as im:
    out = im.resize((128, 128))
    out = im.rotate(45)
    out.save('test.jpg', "JPEG")
```

https://pillow.readthedocs.io/

Pillow

- Compare Images

```
from PIL import Image,ImageChops
img1=Image.open('comp1.jpg')
img2=Image.open('comp2.jpg')
diff=ImageChops.difference(img1,img2)
if diff.getbbox():
    diff.show()
```



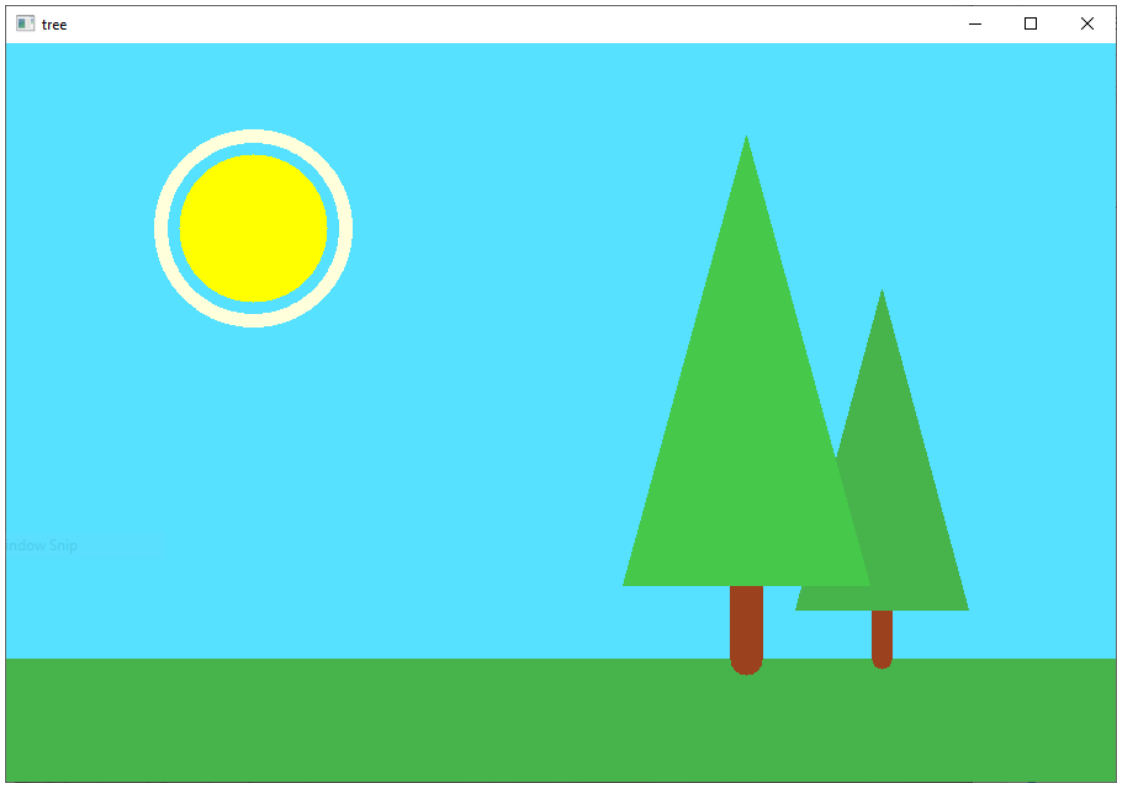
https://pillow.readthedocs.io/

OpenCV

- OpenCV: Open Source Computer Vision Library
- Great tool for image processing and performing computer vision tasks.

```python
import cv2
# Anaconda Verion
img=cv2.imread('dog.jpg',1) #0 Grayscale, 1 for color, -1 alpha channel
cv2.imshow('My Dog', img)
k=cv2.waitKey(0)
if k==27:
    cv2.destroyAllWindows()
elif k==ord('s'):
    cv2.imwrite('copyimage.jpg',img)
    cv2.destroyAllWindows()

# #Colab version
# from google.colab.patches import cv2_imshow
# img=cv2.imread('dog.jpg') #0 Grayscale, 1 for color, -1 alpha channel
# cv2_imshow(img)
# cv2.imwrite('copyimage.jpg',img)
```

```python
import cv2 as cv
import numpy as np

img=np.zeros((600,900,3),dtype=np.uint8)
#Background
cv.rectangle(img,(0,0),(900,500),(255,225,85),-1)
cv.rectangle(img,(0,500),(900,600),(75,180,70),-1)

#Sun
cv.circle(img,(200,150),60,(0,255,255),-1)
cv.circle(img,(200,150),75,(220,255,255),10)


#tree1

cv.line(img,(710,500),(710,420),(30,65,155),15)

traingle1=np.array([[640,460],[780,460],[710,200]],dtype=np.int32)
cv.fillPoly(img,[traingle1],(75,180,70))

#tree2

cv.line(img,(600,500),(600,420),(30,65,155),25)

traingle2=np.array([[500,440],[700,440],[600,75]],dtype=np.int32)
cv.fillPoly(img,[traingle2],(75,200,70))

cv.imshow("tree",img)
cv.waitKey(0)
cv.destroyAllWindows()
```



Learn more…

- https://www.w3schools.com/python/
- https://www.w3schools.com/python/numpy/
- https://numpy.org/doc/stable/
- https://www.tutorialspoint.com/python/index.htm