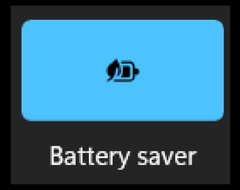
ROBOTICS CLUB @ SASTRA

COMPUTER VISION WORKSHOP

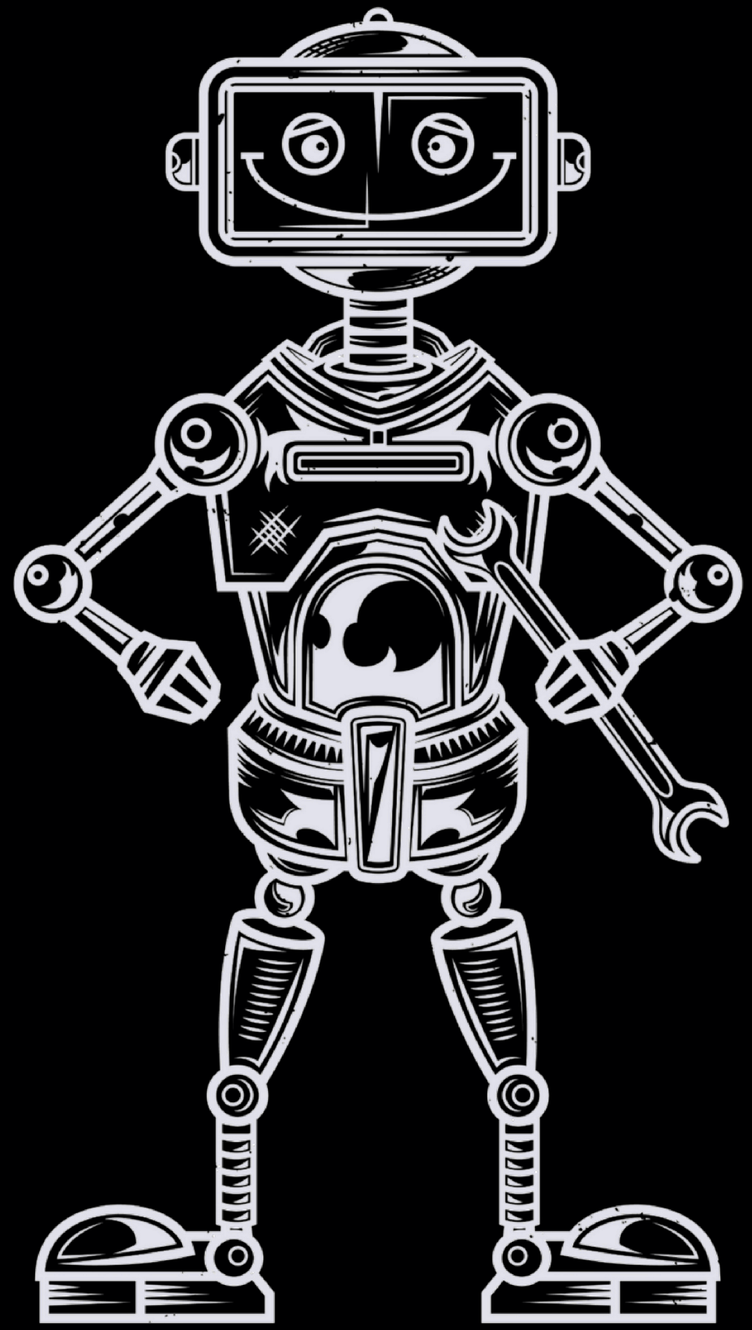
SPECIAL NOTE

Turn off your bluetooth



Turn on Battery Saver

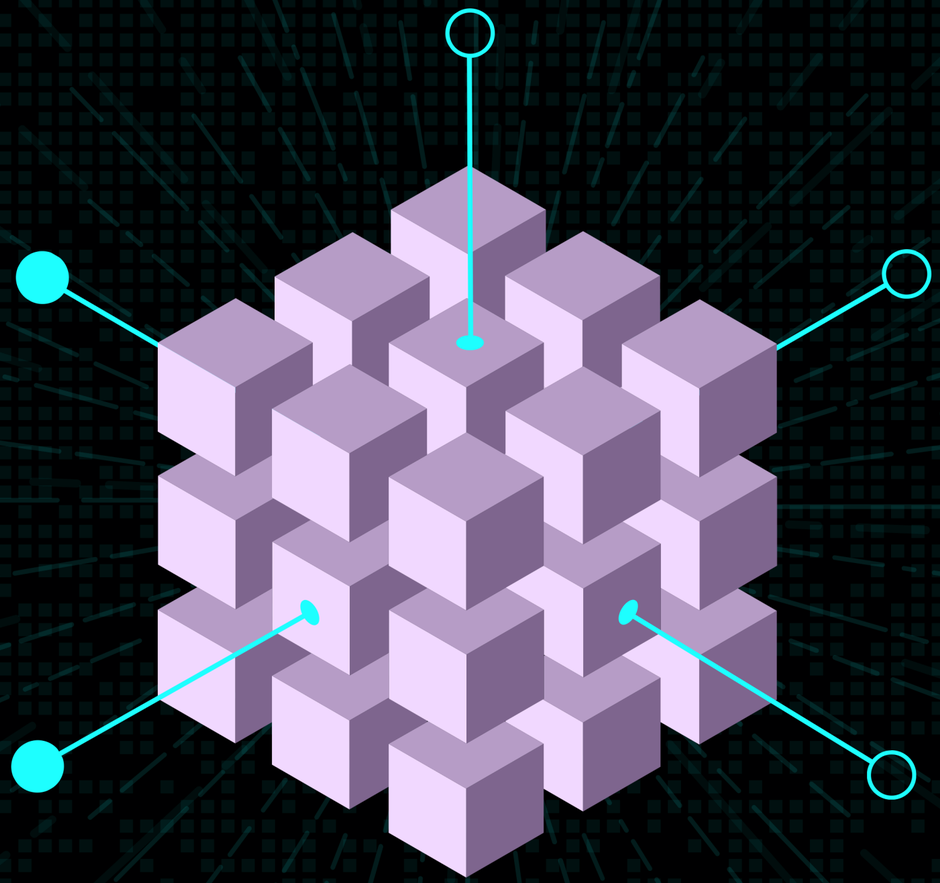
Turn on Dark Theme



WORKSHOP ROADMAP

Introduction to Python

Session 1



Pose Detection

Session 4

Image Processing

Session 2

Cats and Dogs

Session 3

Introduction to ML

Session 2

INTRODUCTION TO PYTHON

SESSION - 1

TO PRINT ON THE CONSOLE

```
print ("Welcome to RCS")
print ("%s is a string, %d is a decimal integer and %.3f is
floating number upto 3 decimal place"%("Python",4,3.33))
```

TO GET USER INPUT

```
name = input("Type your name and press enter : ")
print ("Welcome '%s' to Computer Vision Workshop." %name)
```

COMMENTS

Single line-#

Multiline-(' ' ' ' ' ') [three single quotes]

BUILT-IN DATA TYPES IN PYTHON

int :

```
integer_number = 52
print (integer_number)
print (type(integer_number))
```

float:

```
float_number = 56.78
print (type(float_number))
```

complex :

```
cmp_number = 3 + 4j
z = complex(3,4)
print ("Complex Number:",cmp_number,"\tType:",type(cmp_number))
print ("Complex Number:",z,"\tType: ",type(z))
```

STRINGS

- Created by enclosing characters in quotes – single, double or triple quotes.
- Python treats single quotes the same as double quotes

Repetition:

```
string_1 = 4 * "Hello"
print (string_1)
```

STRINGS

Concatenation:

```
string_1 = "Hello Python."
string_2 = 'Welcome to programming'
print (string_1 + string_2)
print (string_1+ " " + string_2 + " world.")
```

SLICING AND INDEXING IN STRINGS

- Use square brackets for slicing [ ]
- Indexing starts from 0
- variable name[a : b : c] ------ a : starting index
  b : ending index
  c : step size (default 1)
- negative indexing - the last element is also index as -1 going through -2,-3.. as we go from right to left

SLICING AND INDEXING IN STRINGS

```
string_1 = "Hello Python"
print (string_1[0])        # print first character print (string_1[-1])
print (string_1[0 : 2])    # print character at 0 and 1 index
print (string_1[-1 : ])    # printing from backward. print character at last index
print (string_1[1 : -2])   #printing from 2nd character to last second character
print (string_1[ : : 2])   # printing every alternate digit
print (string_1[ : : -1])  # printing reverse string
print (type(string_1[0]))
```

IN-BUILT STRING FUNCTIONS

- capitalize() - return string with first letter in upper case
- upper() - return string with all letter in upper case
- lower() - return string with all letter in lower case
- count(str) - count occurrence of 'str' in string
- len() - return length of the string

IN-BUILT STRING FUNCTIONS

- find(str) - find 'str' in string. If found: return start index of first occurrance str, else return -1
- join(seq) - join string represented as sequence into a string with a separator string between each string
- len(str) - returns length of string
- replace() - returns a copy of the string in which the occurrences of old have been replaced with new, optionally restricting the number of replacements to max.

WHAT ARE MODULES IN PYTHON ?

A file containing a set of functions you want to include in your application.

```
import math as m        # Here "m" is used as short hand for "math"
                                #module which is imported.

print (m.ceil(2.33))     # return greater integer nearer
print (m.floor(2.33))   # return least integer nearer
print (m.exp(1))          # exp(x) - return e**x
```

WHAT ARE THE MODULES IN PYTHON?

```
print (m.pi)              # return pi value as 3.14159. . . . . .
print (m.log(2,2))        # return the value log 2 base 2
print (m.sqrt(4))         # return the square root of 4
```

MATH OPERATORS

Addition:	print(x+y)
Subtraction:	print(x-y)
Multiplication:	print(x*y)
Division:	print(x/y)
Floored quotient:	print(x//y)
Remainder:	print(x%y)
Power:	print(x**y)
Remainder and Quotient:	print(divmod(x,y)) print(type(divmod(x,y))

CONDITIONAL STATEMENTS

IF :

```
if (condition):
        print ("Statement")
```

CONDITIONAL STATEMENTS

IF-ELSE:

```
if (condition):
        print ("Statement")
else:
        print ("Statement")
```

CONDITIONAL STATEMENTS

IF-ELIF-ELSE:

```
if (condition 1):
        print ("Statement")
elif(condition 2):
        print("Statement")
else:
        print ("Statement")
```

CONDITIONAL STATEMENTS

NESTED IF-ELSE:

```
if (condition 1):
        print ("Statement")
elif(condition 2):
        if(condition 3):
                print("Statement")
        else:
                print ("Statement")
```

LOOPS

</>

While Loop:

```
while(condition):
    code_block
```

Eg:

```
i = 0
while i<5:
    print (i * i)
    i += 1
```

Output:

```
0
1
4
9
16
```

LOOPS

For Loop :

Range function :

Syntax : range (arg1, arg2, arg3)

arg1: start value

arg2: stop value

arg3: step-size (default 1)

LOOPS

For Loop :

Eg :

```
for i in range (0 , 5):
    print (i)
```

Output:

```
0
1
2
3
4
```

TUPLES

A tuple is a sequence of comma-separated values enclosed within parenthesis.
Tuple is also an immutable Python data structure.
Eg : ("a", "c", "t", "r", "e").
Representation:

```
tup = ( ) #empty tuple
```

TUPLES

</>

- Tuple are immutable –elements inside tuples can't be changed.
- A tuple can store elements of different datatypes – string, integer, etc.
- Tuple can be converted to other data types such as string.

Accessing:

- Using slicing and indexing – as we did in string manipulation

TUPLES

Updating:

- Updating exsisting tuples is not allowed.
- New tuple can be creating by using exsisting tuple by using Concatenation operator

Conversion of string to tuple:

```
string_to_tuple = tuple("RCS")                    # string to tuple
print (str(string_to_tuple))          #how to convert back to string
print (len(str(string_to_tuple)))
tuple_to_string = "".join(string_to_tuple)          # tuple to string
```

TUPLES

Basic Tuple operations:

- len(tuple) #returns length of tuple
- min(tuple) #returns the minimum value of tuple
- max(tuple) #returns the maximum value of tuple

TUPLES

Basic Tuple operations:

- Membership ex :

```
tup_1 = (1, 'ABC')
print (ABC in tup_2)          # return true
print (2 in tup_2)            # return false
```

- Iteration ex:

```
tup_2 = (1, 2, 'A', 'B')
print ("Printing All elements of tuple")
for i in tup_2:
        print (i)
```

LISTS

A list is a sequence of comma-separated values enclosed within square brackets [ ].
List is also a mutable Python data structure.
Eg : [1 , 2 , 3 , 4 , 5].

- Declaring : [ ] or list()
- Accessing : Using slicing and indexing as we did in string manipulation

print (list name [0: n-1])

LISTS

- Updating the list using the :

1. INDEX :
   - You can replace old elements with new.
   - New elements can not be added to list.

alphabets = ['a','b','c'] -> print(alphabets) -> ['a','b','c']
alphabets[1] = 'd' -> print(alphabets) -> ['a','d','c']

LISTS

2. append() Method:

- Method takes one input as a parameter.
- Add new element to the end of the list:

list name.append('element').

Insertion of an element:

- list name.insert(index,"element")

- "list name" is the name of the list
- "index" -a position where insertion needs to be done

- Extending list:

list name1.extend(list name2)

LISTS

Basic List operation:

- len(list_name) #return list length
- min(list_name) #return min value of list
- max(list_name) #return max value of list
- Repetition:

```
z=4*[96 , ]
print(z) ----------> [96 , 96 , 96 , 96]
```

- Membership:

```
z=["a","b","c","d","e"]
print("a" in z) ----------> True
print("f" in z) ----------> False
```

DICTIONARIES

A dictionary is a set of key : value pairs separated by commas and enclosed within curly braces "{ }".

Eg: {1 : "Sunday", 2 : "Monday", 3 : "Tuesday"}

- Key in a dictionary is immutable, whereas the Value in it is mutable.
- Each key and value is separated by colons (:) in a dictionary.

Declaring a dictionary :

- z= { } #empty dictionary
- z= dict( ) #empty dictionary
- z= {1 : "Sunday", 2 : "Monday", 3 : "Tuesday"}

DICTIONARIES

Accessing a dictionary :

- A dictionary is accessed by its key.

Eg:

z= {1 : "Sunday", 2 : "Monday", 3 : "Tuesday"}

print(z[1]) ----------------> Sunday

Updating Dictionary :

- z[4] = "Wednesday" # adding new element

print(z) ----->{1 : "Sunday", 2 : "Monday", 3 : "Tuesday", 4 : "Wednesday"}

- z[2] = "Friday" # updating existing element

print(z) ----->{1 : "Sunday", 2 : "Friday", 3 : "Tuesday"}

DICTIONARIES

Delete element from Dictionary :

- del(z[3]) #Value corresponding to key value "3" is deleted
  print (z) ------->{2 : "Monday", 3 : "Tuesday"}
- del(z) #delete complete dictionary

Dictionary Builtin functions:

- len(z) - returns length of dictionary i.e. number of keys in dictionary
- str(z) - return string representation of entire dictionary
- type(z) - standard type() function. Return "dict" for dictionary variable

FUNCTIONS

A function is a user defined one in which users defines the flow of execution of the program according to them.

Syntax:

```
def funcName(parameters):
    # perform operations
    returnStatement
```

A semicolon (;) in python is used to denote separation and not termination. But its not necessary.....

FUNCTIONS

Eg :

```
def funcName():
        print("Welcome to CV Workshop.")

funcName();     # Function call
```

THANK YOU