Python Foundation with Program -Batch 2

Variable:

Variable is a name that is used to refer to a memory location.

PYTHON variable is also known as identifier and used to hold values.

```
price = 100

tax = 18

totalprice = price + tax

print(totalprice)


#how variables work?

x = 30

y = "Noddy"

z = 4.4


#if we say

w = z

print(w)
```

Typecasting

```
a = 10

print(type(a))

b = 10.5

print(type(b))

c = "Noddy"

print(type(c))
```

```
d = 2 + 3

#in python, we use none to indicate that we dont
# know the value yet or we didnt assign
# any value to this variable

ab = a
print(type(ab))
```

Assigning:

```
x = 10
print(x)
x = "noddy"
print(x)

#print(x) #using a variable without assigning its value causes error
```

Strings:

```
#strings
#a string is a sequence of characters enclosed in single or double quotes
#strings are immutable, meaning once created, they cannot be changed.
my_string = "Hello, World!"
my_string2 = 'Python is fun!'
my_string3 = '''This is a
 multi-line
 string.'''
print(my_string)

#string indexing
#each character in a string has an index, starting from 0 for the first character
print(my_string[0]) #H
print(my_string[-1]) #!
```

```
#string slicing
#you can extract a substring from a string using slicing
print(my_string[0:5]) #Hello
print(my_string[7:]) #World!
print(my_string[:5]) #Hello

#concatination
#you can concatenate two or more strings using the + operator
greeting = "Hello"
name = "Alice"
message = greeting + ", " + name + "!"
print(message) #Hello, Alice!

#repetition
#you can repeat a string multiple times using the * operator
laugh = "Ha" * 3
print(laugh) #HaHaHa

#string methods
#strings have many built-in methods for manipulating and transforming them
print(my_string.lower()) #hello, world!
print(my_string.upper()) #HELLO, WORLD!
print(my_string.replace("World", "Python")) #Hello, Python!
print(my_string.split(",")) #['Hello', ' World!']
print(my_string.strip()) #Hello, World! #here it removes any leading or trailing
whitespace
print(len(my_string)) #13 #length of the string
print(my_string.find("World")) #7 #index of the first occurrence of the substring
print(my_string.startswith("Hello")) #True
print(my_string.endswith("!")) #True
print(my_string.count("o")) #2 #number of occurrences of the substring
print(my_string.isalpha()) #False #checks if all characters are alphabetic
print(my_string.isdigit()) #False #checks if all characters are digits
print(my_string.isalnum()) #False #checks if all characters are alphanumeric
print(my_string.title()) #Hello, World! #capitalizes the first letter of each word
print(my_string.capitalize()) #Hello, world! #capitalizes the first letter of the string
print(my_string.center(20)) #   Hello, World!    #centers the string within a specified width
print(my_string.zfill(20)) #0000000000Hello, World! #pads

#string formatting
#you can format strings using f-strings (Python 3.6+), the format() method, or the % operator
name = "Bob"
age = 25
#using f-strings
formatted_string = f"My name is {name} and I am {age} years old."
print(formatted_string) #My name is Bob and I am 25 years old.

#using the format() method
formatted_string2 = "My name is {} and I am {} years old.".format(name, age)
print(formatted_string2) #My name is Bob and I am 25 years old.
```

```
#using the % operator

formatted_string3 = "My name is %s and I am %d years old." % (name, age)

print(formatted_string3) #My name is Bob and I am 25 years old.


#ITERATING THROUGH A STRING

for char in my_string:

    print(char) #prints each character in the string on a new line


#STRINGS ARE MOST IMPORTANT IN PYTHON AS THEY ARE USED TO STORE AND MANIPULATE TEXTUAL DATA.

#they are widely used in various applications such as web development, data analysis, and machine learning.

#they provide a convenient way to work with text and perform various operations on it.

#understanding strings and their methods is essential for any python programmer.


#escape sequences in strings

#escape sequences are special characters that are used to represent certain characters in a string

#they are preceded by a backslash (\) and are used to represent characters that are difficult

#to type directly into a string, such as newlines, tabs, and quotes.

#some common escape sequences in python are:

#\n- newline

#\t- tab  #space

#\\- backslash #represents a single backslash

#\'- single quote
```

```
#\"- double quote
#\r- carriage return #moves the cursor to the beginning of the line
#\b- backspace #deletes the character before the cursor
#\f- form feed #advances the cursor to the next page
#\v- vertical tab #moves the cursor down to the next vertical tab stop
#example of escape sequences in strings
escaped_string = "Hello,\nWorld!\tThis is a string with escape sequences.\nHe said, \"Python is fun!\""
print(escaped_string)


#backslash example
path = "C:\\Users\\Alice\\Documents\\file.txt"
print(path) #C:\Users\Alice\Documents\file.txt


#raw strings
#raw strings are strings that treat backslashes as literal characters
#they are created by prefixing the string with an 'r' or 'R'
raw_string = r"C:\Users\Alice\\Documents\\file.txt"
print(raw_string) #C:\Users\Alice\\Documents\\file.txt


#raw strings are useful when working with regular expressions or file paths, where backslashes are commonly used.
```

<u>Functions</u>

```
#function is a block of code which only runs when it is called
#we can pass data, known as parameters, into a function.
#A function can return data as a result.

#syntax
#def function_name(parameters):
   #code to be executed

#types of functions
#built-in functions- print(), len(), type()
#user-defined functions- created by u

def greet():
   print("Hello welcome to python")
greet()  #calling the function
```

```
#funtion with parameters
def add(a, b):
    sum = a + b
    print("sum is", sum)
add(3, 5)  #calling the function with arguments
```

<u>while loop</u>

in python, a while loop is used to repeatedly execute a block of code as long as

# a specified condition is true.

```
#syntax of while loop
'''while condition:
    # code to be executed
    # increment/decrement'''

#example of while loop
i = 1
while i <= 5:
    print(i)
    i += 1  #i = i + 1
print("loop ended")
#in the above example, the loop will continue to execute as long as i is less than or equal to 5
```

<u>Input Function</u>

this function is used to take input from the user

```
#input()


'''name = input("Enter the name = ")

print("Welcome " + name )'''


#python function for addition

x = input("Enter first number = ")

y = input("Enter second number = ")

x = int(x)

y = int(y)

res = x + y

print("sum is", res)


#input() always gives us a string so wherever we need mathematical exps,

#we have to convert it from strings
```

write a program that accepts a sentence and

calculATE the number of letters and digits.

suppose the following input is supplied to the program:

```
#hello world! 123

#then, the output should be:

#LETTERS 10

#DIGITS 3
```

```
sentence = input("Enter a sentence: ")
letters = 0
digits = 0

for char in sentence:
    if char.isalpha():
        letters += 1
    elif char.isdigit():
        digits += 1

        print("LETTERS", letters)
        print("DIGITS", digits)
```

write a python program to find largest element in an array.

```
def find_largest_element(arr):
    if not arr:
        return "Array is empty"  # Handle empty array case
    largest_element = arr[0]
    for element in arr:
        if element > largest_element:
         largest_element = element
    return largest_element
my_array=(10,20,30,40)
result= find_largest_element(my_array)
print(f"largest element in the array is:{result}")
```

data structures in python

```python
#in python we have several built-in data structures to store and organize data.
#Some of the most commonly used data structures are:

#1. List:
#A list is an ordered collection of items that can be of different types. Lists are mutable, meaning you can change their content.
my_list = [1, 2, 3, 4, 5]
print("List:", my_list)

#2. Tuple: A tuple is similar to a list, but it is immutable, meaning once created, its content cannot be changed.
my_tuple = (1, 2, 3, 4, 5)
print( my_tuple[2])

#3. Set: A set is an unordered collection of unique items.
#Sets are mutable and can be used to perform mathematical set operations like union, intersection, and difference.
my_set = {1, 2, 3, 4, 5}
print("Set:", my_set)

#4. Dictionary: A dictionary is an unordered collection of key-value pairs. Each key must
# be unique, and values can be of any type. Dictionaries are mutable.
my_dict = {'a': 1, 'b': 2, 'c': 3}
print("Dictionary:", my_dict)
```

For Loop

in python, a for loop is used to iterate over a sequence (like a list, tuple, dictionary, set, or string).

```
#syntax of for loop
'''for item in sequence:
    # do something with item'''

#example of for loop
fruits = ["apple", "banana", "cherry"]
for fruit in fruits:
    print(fruit)

#example of for loop with range function
for i in range(5):  # This will iterate from 0 to 4
    print(i)
```

Parameters

parameters are the values that we pass to a function when we call it.

#they act as placeholders for the actual values that will be used when the function is executed.

#parameters are defined within the parentheses in the function definition.

#when we call the function, we provide the actual values, known as arguments, that correspond to the parameters.

#this allows us to create functions that can work with different inputs and produce different outputs based on those inputs.

```
#example of function with parameters
'''def multiply(x, y):
    result = x * y
    return result
#calling the function with arguments
product = multiply(4, 5)
print("product is", product)'''

#in the above example, x and y are parameters of the multiply function.
#when we call the function with arguments 4 and 5, those values are passed to
#the parameters x and y, respectively
#the function then calculates the product and returns it.

#we can also use default parameters in a function
def power(base, exponent=3):    return base ** exponent
#calling the function with one argument
squared = power(3)
print("squared is", squared)

#in the above example, exponent has a default value of 2.
#when we call the function with only one argument, the default value is used for
exponent.
```

If-Else Statements

#there comes situations in real life where we need to do some specific tasks

# and ased on some specfic conditions and,

# we decide what should we do next.

#similarly, there comes a asituation in programming where a

#specific task is to be performed if a specific condition is true.

#in such cases, conditional statements can be used and the conditional statements are

#if

#if.. else

#nested if

#if-elif statements

'''if statements- in this statement, if the condition is true, the block of code inside the if statement is executed.

if the condition is false, the block of code inside the if statement is skipped.'''

```
if 5 > 2:
    print("5 is greater than 2") #this will be printed as the condition is true
    print("program ended")
```

#if else statement

#in conditional if statements the additional block of codes is merged as ese statements

```
#which is performed when if condition is false

'''x = 3
if x == 4:
   print("yes")
else:
   print("no")'''

#you can also chain if..else statement with more than one condition.

letter = "A"
if letter == "B":
   print("letter is B")
else:
   if letter == "c":
      print("letter is c")
```

write a python function to generate a random number

between 1 and 100. Import the "Random" module to do this

```
import random
print(random.randint(1,100))
```

scope

```
#scope is a lightweight library for managing variable scopes in Python.
#It allows you to create nested scopes, set and get variables in those scopes,
#and manage the lifetime of variables.
#This can be particularly useful in scenarios where you want to avoid
#global variables or when you want to encapsulate variables within a specific context.
#local scope is a scope that is defined within a function or a block of code.
#variables defined in a local scope are only accessible within that scope.


glob=  "HI, i am GLOBAL variable present throughout the program"  #global scope
def f1():
    local= "Hello, i am LOCAL variable present only within this function"  #local scope
    print(local)  #accessing local variable within local scope
    print(glob)   #accessing global variable within local scope
f1()
print (glob)
print()
```

file handling in python

#file handling is an important aspect of programming that allows us to read, write, and manipulate files on our computer.

#In python, we can handle files using built-in functions and methods.

#we can open a file using the open() function, which takes two arguments: the name of the file and the mode in which we want to open it.

#the mode can be 'r' for reading, 'w' for writing, 'a' for appending, and 'b' for binary mode.

#once we have opened a file, we can read its contents using the read() method, readline() method, or readlines() method.

#we can also write to a file using the write() method or writelines() method.

#after we are done with a file, we should always close it using the close() method to free up system resources.

#we can also use the with statement to handle files, which automatically closes the file after we are done with it.

```
#example of file handling in python

#writing to a file

with open('example.txt', 'w') as file:

    file.write("Hello, World!\n")

    file.write("This is a file handling example in Python.\n")

#hre we have created a file named example.txt and written two lines to it.


#reading from a file

with open('example.txt', 'r') as file:

    content = file.read()

    print(content)
```

#here we have opened the example.txt file in read mode and printed its contents to the console.

```
#appending to a file
with open('example.txt', 'a') as file:
    file.write("This line is appended to the file.\n")
```

#here we have opened the example.txt file in append mode and added a new line to it.

```
#reading the updated file
with open('example.txt', 'r') as file:
    content = file.read()
    print(content)
```

#here we have opened the example.txt file in read mode again and printed its updated contents to the console.

Regex Functions

'REGEX- REGular EXpressions'

#Regular Expressions (Regex) are a patterns matching language used for searching and manipulating strings.

#they provide a powerful way to perform complex string operations such as searching, replacing, and validating text.

#Regex patterns are made up of special characters and symbols that define the search criteria.

#in python, the re module provides support for working with regular expressions.

```
import re
```

#basic regex functions

#1. re.match(): This function checks if the beginning of a string matches a specified pattern.

```
pattern = r'Hello' #here r denotes a raw string, which treats backslashes as literal characters

string = "Hello, World!"

match = re.match(pattern, string)

if match:

   print("Match found:", match.group())

else:

   print("No match found.")
```

#2. re.search(): This function searches the entire string for a match to the specified pattern.

```
pattern = r'World'

search = re.search(pattern, string)
```

```
if search:
    print("Search found:", search.group())
else:
    print("No match found.")

#3. re.findall(): This function returns a list of all non-overlapping matches of the
pattern in the string.
pattern = r'\d+' #\d+ matches one or more digits
string = "There are 3 apples and 5 oranges."
findall = re.findall(pattern, string)
print("Findall found:", findall) #['3', '5']

#another example of re.findall() with non-overlapping matches
text = 'aaaa' #-indexed 0123
pattern = r'aa' #-indexed 01, 23
findall = re.findall(pattern, text)
print("Findall found:", findall) #['aa', 'aa']

#non-overlapping matches means that once a part of the string is matched,
#it cannot be used again for another match.

#first 'aa' is matched at index 0 and 1,
#then the next 'aa' is matched at index 2 and 3.
#the "aa" at index 1 is skipped because it overlaps with the first match.

#4. re.sub(): This function replaces all occurrences of the pattern in the string
with a specified replacement string.
```

```
pattern = r'apples'

replacement = 'bananas'

string = "I like apples. Apples are my favorite fruit."

sub = re.sub(pattern, replacement, string, flags=re.IGNORECASE) #here flags=re.IGNORECASE makes the search case-insensitive

print("Sub result:", sub) #I like bananas. bananas are my favorite fruit.


#Regex is widely used in machine learning and data science for data preprocessing and cleaning.

#it is used to extract relevant information from unstructured text data, such as emails, social media posts, and web pages.

#Regex can be used to remove unwanted characters, such as punctuation and special characters, from text data.

#it can also be used to extract specific patterns, such as email addresses, phone numbers, and URLs, from text data.

#Regex is also used in feature engineering, where it can be used to create new features from text data.

#for example, regex can be used to extract the domain name from email addresses, which can be used as a feature in a machine learning model.

#Overall, regex is a powerful tool for working with text data in machine learning and data science
```