Python Basic Assignment - 1

Question1.

In the below elements which of them are values or an expression?
eg:- values can be integer or string and expressions will be mathematical operators.

(a) *
(b) 'hello'
(c) -87.8
(d) -
(e) /
(f) +
(g) 6

Answer:

(a) expression
(b) value
(c) value
(d) expression
(e) expression
(f) expression
(g) value

Question2.

What is the difference between string and variable?

Answer:

Variable is nothing but a kind of an entity (reserved memory location) which helps us out to hold data and gives the data to the computer when it is required by the computer for processing. Whereas string, which is a type of data that can be stored in a variable, is a sequence of characters that can be enclosed by single, double or triple quotation mark.

Question3.

Describe three different data types.

Answer:

(1) Integer data type: It is a data type used to store positive or negative counting numbers, along with zero, which have having no fractional component.

(2) Float data type: It is a data type used to store integers having fractional component. Fractional component is separated by a decimal.

(3) Boolean data type: It is a data type that represents one of two possible values, True or False.

Question4.

What is an expression made up of? What do all expressions do?

Answer:

An expression is a combination of constants, values ,variables, operands and operators, where operators include arithmetic and boolean operators, the function call operator (), the subscription operator [] etc.

All expressions produces the result, which is always a single value, when reduced or evaluated down with the help of an interpreter.

Question5.

This assignment statements, spam = 10. What is the difference between an expression and a statement?

Answer:

An expression always evaluates to a single value, whereas a statement does not. The python executes a statement, but the interpreter does not display any results, whereas in case of an expresssion, the interpreter evaluates an expression and displays the result.

Question6.

After running the following code, what does the variable bacon contain?

bacon = 22
bacon + 1

Answer:

After running the given code, the variable bacon still contains 22 because the bacon variable is assigned to 22. The bacon + 1 expression does not reassign the value in bacon.

If we want to reassign, we would need an assignment statement:
bacon = bacon + 1

Crosscheck by running the program:

In [1]:

```
bacon = 22
bacon + 1
```

Out[1]:

```
23
```

In [2]:

```
bacon
```

Out[2]:

```
22
```

Question7.

What should the values of the following two terms be?

'spam' + 'spamspam'
'spam'*3

Answer:

Both of the terms would give 'spamspamspam' as output.

Crosscheck by running the program:

In [3]:

```
'spam' + 'spamspam'
```

Out[3]:

```
'spamspamspam'
```

In [4]:

```
'spam'*3
```

Out[4]:

```
'spamspamspam'
```

Question8.

Why is eggs a valid variable name while 100 is invalid?

Answer:

A variable name cannot start with a number.
In the given example, 100 is an invalid variable because it starts with a number 1

Question9.

What three functions can be used to get the integer, floating-point number, or string version of a value?

Answer:

The int(), float(), and str() functions will evaluate to the integer, floating-point number or string versions of the value we pass, respectively.

Question10.

Why does this expression cause an error? How can you fix it?

'I have eaten' + 99 + 'burritos.'

In [6]:

```
'I have eaten' + 99 + 'burritos.'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_12316/3116421743.py in <module>
----> 1 'I have eaten' + 99 + 'burritos.'

TypeError: can only concatenate str (not "int") to str
```

Answer:

The expression causes an error because 99 is an integer, and the + operator can only concatenate strings to other strings (not integers). The correct way is by concatenating the string version of the integer 99 in any one of the following ways:

In [7]:
```
'I have eaten ' + str(99)  + ' burritos.'
```

Out[7]:
```
'I have eaten 99 burritos.'
```

In [8]:
```
'I have eaten ' + '99'  + ' burritos.'
```

Out[8]:
```
'I have eaten 99 burritos.'
```

In [9]:
```
'I have eaten ' + "99"  + ' burritos.'
```

Out[9]:
```
'I have eaten 99 burritos.'
```

In [10]:
```
'I have eaten ' + '''99'''  + ' burritos.'
```

Out[10]:
```
'I have eaten 99 burritos.'
```

Python Basic Assignment - 17

Question1.

Assign the value 7 to the variable guess_me. Then, write the conditional tests (if, else, and elif) to print the string 'too low' if guess_me is less than 7, 'too high' if greater than 7, and 'just right' if equal to 7.

Answer:

In [1]:
```
guess_me = 7
if guess_me < 7:
    print("too low")
elif guess_me > 7:
    print("too high")
else:
    print("just right")
```

```
just right
```

Question2.

Assign the value 7 to the variable guess_me and the value 1 to the variable start. Write a while loop that compares start with guess_me. Print too low if start is less than guess me. If start equals guess_me, print 'found it!' and exit the loop. If start is greater than guess_me, print 'oops' and exit the loop. Increment start at the end of the loop.

Answer:

In [2]:

```
guess_me = 7
start = 1

while True:

    if start < guess_me:
        print("too low")

    elif start == guess_me:
        print("found it!")
        break

    else:
        print("oops")
        break

    start = start + 1
```

```
too low
too low
too low
too low
too low
too low
found it!
```

Question3.

Print the following values of the list [3, 2, 1, 0] using a for loop.

Answer:

In [3]:

```
list = [3,2,1,0]

for i in list:
    print(i)
```

```
3
2
1
0
```

Question4.

Use a list comprehension to make a list of the even numbers in range(10)

Answer:

In [4]:

```
Even_List = [i for i in range(10) if i%2 == 0]
Even_List
```

Out[4]:

```
[0, 2, 4, 6, 8]
```

Question5.

Use a dictionary comprehension to create the dictionary squares. Use range(10) to return the keys, and use the square of each key as its value.

Answer:

In [5]:

```
Squares_Dictonary = {i:i**2 for i in range(10)}
Squares_Dictonary
```

Out[5]:

```
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81}
```

Question6.

Construct the set odd from the odd numbers in the range (10) using a set comprehension.

Answer:

In [6]:

```
Odd_set = {i for i in range(10) if i%2 == 1}
Odd_set
```

Out[6]:

```
{1, 3, 5, 7, 9}
```

Question7.

Use a generator comprehension to return the string 'Got ' and a number for the numbers in range(10). Iterate through this by using a for loop.

Answer:

In [7]:

```
string_generator = ('Got ' + str(i) for i in range(10))
for i in string_generator:
    print(i)
```

```
Got 0
Got 1
Got 2
Got 3
Got 4
```

```
Got 5
Got 6
Got 7
Got 8
Got 9
```

Question8.

Define a function called good that returns the list ['Harry', 'Ron', 'Hermione'].

Answer:

In [8]:

```
def good():
    list = ['Harry', 'Ron', 'Hermione']
    return list

good()
```

Out[8]:

```
['Harry', 'Ron', 'Hermione']
```

Question9.

Define a generator function called get_odds that returns the odd numbers from range(10). Use a for loop to find and print the third value returned.

Answer:

In [9]:

```
def get_odds():
    for i in range(10):
        if i%2 == 1:
            yield i

count = 1
for i in get_odds():
        if count == 3:
            print("The third value returned from the function get_odds is ",i)
            break
        count = count + 1
```

```
The third value returned from the function get_odds is  5
```

Question10.

Define an exception called OopsException. Raise this exception to see what happens. Then write the code to catch this exception and print 'Caught an oops'.

Answer:

In [10]:

```
class OopsException(Exception):
    pass
```

```
def Exception_Raiser(number):
    if number < 0:
        raise OopsException(number)


try:
    Exception_Raiser(-32453445)
except OopsException as OE:
    print('Caught an oops')
```

```
Caught an oops
```

Question11.

Use zip() to make a dictionary called movies that pairs these lists: titles = ['Creature of Habit', 'Crewel Fate'] and plots = ['A nun turns into a monster','A haunted yarn shop'].

Answer:

In [11]:

```
titles = ['Creature of Habit', 'Crewel Fate']
plots = ['A nun turns into a monster','A haunted yarn shop']

movies = dict(zip(titles,plots))

movies
```

Out[11]:

```
{'Creature of Habit': 'A nun turns into a monster',
 'Crewel Fate': 'A haunted yarn shop'}
```

Python Basic Assignment - 19

Question 1.

Make a class called Thing with no contents and print it. Then, create an object called example from this class and also print it. Are the printed values the same or different?

Answer:

In [1]:

```
class Thing:
    pass

print(Thing)

example = Thing()
print(example)
```

```
<class '__main__.Thing'>
<__main__.Thing object at 0x000001722DE27760>
```

Printed values are not same, second one prints details of object along with the details of class.

Question 2.

Create a new class called Thing2 and add the value 'abc' to the letters class attribute. Letters should be printed.

Answer:

In [2]:

```
class Thing2:
    letters = 'abc'

print(Thing2.letters)
```

```
abc
```

Question 3.

Make yet another class called, of course, Thing3. This time, assign the value 'xyz' to an instance (object) attribute called letters. Print letters. Do you need to make an object from the class to do this?

Answer:

In [3]:

```
class Thing3:

    def __init__(self,letters):
        self.letter = letters

    def alternative(self):
        print(self.letter)

Thing3('xyz').alternative()
```

```
xyz
```

No, we don't need to make an object from the class to print letters.

Question 4.

Create an Element class with the instance attributes name, symbol, and number. Create a class object with the values 'Hydrogen', 'H', and 1.

Answer:

In [4]:

```
class Element:
    def __init__(self, name, symbol, number):
        self.name = name
        self.symbol = symbol
        self.number = number

Element_class_object = Element('Hydrogen','H',1)
```

In [5]:

```
Element_class_object.name
```

Out[5]:

```
'Hydrogen'
```

In [6]:

```
Element_class_object.symbol
```

Out[6]:

```
'H'
```

In [7]:

```
Element_class_object.number
```

Out[7]:

```
1
```

Question 5.

Make a dictionary with these keys and values: 'name': 'Hydrogen', 'symbol': 'H', 'number': 1. Then, create an object called hydrogen from class Element using this dictionary.

Answer:

In [8]:

```
dictionary = {'name':'Hydrogen','symbol':'H','number':1}
print(dictionary)
print()

# Method 1
hydrogen = Element(*dictionary.values())
print('Using Method #1 ---->', hydrogen.name, hydrogen.symbol, hydrogen.number, sep='\t'
)

# Method 2
hydrogen = Element(**dictionary)
print('Using Method #2 ---->', hydrogen.name, hydrogen.symbol, hydrogen.number, sep='\t'
)
```

```
{'name': 'Hydrogen', 'symbol': 'H', 'number': 1}

Using Method #1 ----> Hydrogen H 1
Using Method #2 ----> Hydrogen H 1
```

Question 6.

For the Element class, define a method called dump() that prints the values of the object's attributes (name, symbol, and number). Create the hydrogen object from this new definition and use dump() to print its attributes.

Answer:

In [9]:

```
class Element:

    def __init__(self, name, symbol, number):
        self.name = name
        self.symbol = symbol
        self.number = number
```

```
    def dump(self):
        return print(self.name, self.symbol, self.number)

hydrogen = Element('Hydrogen','H',1)
hydrogen.dump()
```

```
Hydrogen H 1
```

Question 7.

Call print(hydrogen). In the definition of Element, change the name of method dump to str, create a new hydrogen object, and call print(hydrogen) again.

Answer:

In [10]:

```
print(hydrogen)


class Element:

    def __init__(self, name, symbol, number):
        self.name = name
        self.symbol = symbol
        self.number = number

    def __str__(self):
        return f'{self.name} {self.symbol} {self.number}'

hydrogen = Element('Hydrogen','H',1)
print(hydrogen)
```

```
<__main__.Element object at 0x000001722DE6FDC0>
Hydrogen H 1
```

Conclusion:

By using print(hydrogen), we can directly access the return value of the str method.

Question 8.

Modify Element to make the attributes name, symbol, and number private. Define a getter property for each to return its value.

Answer:

In [11]:

```
class Element:
    def __init__(self,name,symbol,number):
        self.__name = name
        self.__symbol = symbol
        self.__number = number

    @property
    def get_name(self):
        return self.__name

    @property
    def get_symbol(self):
```

```
        return self.__symbol

    @property
    def get_number(self):
        return self.__number

hydrogen = Element('Hydrogen','H',1)
print(hydrogen.get_name)
print()
print(hydrogen.get_symbol)
print()
print(hydrogen.get_number)
```

```
Hydrogen

H

1
```

Question 9.

Define three classes: Bear, Rabbit, and Octothorpe. For each, define only one method: eats(). This should return 'berries' (Bear), 'clover' (Rabbit), or 'campers' (Octothorpe). Create one object from each and print what it eats.

Answer:

In [12]:

```
#Defining three classes

class Bear:
    def eats(self):
        print('berries')
class Rabbit:
    def eats(self):
        print('clover')
class Octothorpe:
    def eats(self):
        print('campers')


#Creating one object from each
bear_object = Bear()
rabbit_object = Rabbit()
octothrope_object = Octothorpe()


#printing what each eats
bear_object.eats()
rabbit_object.eats()
octothrope_object.eats()
```

```
berries
clover
campers
```

Question 10.

Define these classes: Laser, Claw, and SmartPhone. Each has only one method: does(). This returns 'disintegrate' (Laser), 'crush' (Claw), or 'ring' (SmartPhone). Then, define the class Robot that has one instance (object) of each of these. Define a does() method for the Robot that prints what its component objects do.

Answer:

In [13]:

```
class Laser:
    def does(self):
        return 'disintegrate'
class Claw:
    def does(self):
        return 'crush'
class Smartphone:
    def does(self):
        return 'ring'


class Robot:
    def __init__(self):
        self.laser = Laser()
        self.claw = Claw()
        self.smartphone = Smartphone()
    def does(self):
        print(self.laser.does(), self.claw.does(), self.smartphone.does())


object_for_Robot_class = Robot()
object_for_Robot_class.does()
```

```
disintegrate crush ring
```

Python Basic Assignment - 25

Question 1)

What is the difference between enclosing a list comprehension in square brackets and parentheses?

Answer:

Enclosing a list comprehension in square brackets returns a list, whereas enclosing a list comprehension in parentheses looks like it might create a tuple, but it returns a generator object.

In [4]:

```
List = [i for i in range(10)]
List
```

Out[4]:

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

In [5]:

```
Generator = (i for i in range(10))
Generator
```

Out[5]:

```
<generator object <genexpr> at 0x00000236E1BE9270>
```

Question 2)

What is the relationship between generators and iterators?

Answer:

Answer:

An `iterator` is an object which contains a countable number of values and it is used to iterate over iterable objects like list, tuples etc. `Iterators` are implemented using a class. It follows lazy evaluation where the evaluation of the expression will be on hold and stored in the memory until the item is called specifically which helps us to avoid repeated evaluation. As lazy evaluation is implemented, it requires only 1 memory location to process the value and when we are using a large dataset then, wastage of RAM space will be reduced the need to load the entire dataset at the same time will not be there. For an iterator: `iter()` keyword is used to create an `iterator` containing an iterable object. `next()` keyword is used to call the next element in the iterable object.

Similarly `Generators` are an another way of creating `iterators` in a simple way where it uses the keyword `yield` statement instead of `return` statement in a defined function. `Generators` are implemented using a function. Just as `iterators`, `generators` also follow lazy evaluation. Here, the `yield` function returns the data without affecting or exiting the function. It will return a sequence of data in an iterable format where we need to iterate over the sequence to use the data as they won't store the entire sequence in the memory.

In [6]:

```
# Example of iterartor
iter_str = iter(['iNeuron','Full','Stack','Data Science'])
print(type(iter_str))
print(next(iter_str))
print(next(iter_str))
print(next(iter_str))
print(next(iter_str))
print(iter_str) # After the iterable object is completed, to use them again we have reass
ign them to the same object.


print()
# Example of Generator
def cube_numbers(x):
    for i in range(x):
        yield i**3

result = cube_numbers(5)
print(next(result))
print(next(result))
print(next(result))
print(next(result))
print(next(result))
```

```
<class 'list_iterator'>
iNeuron
Full
Stack
Data Science
<list_iterator object at 0x00000236E1BED100>

0
1
8
27
64
```

Question 3)

What are the signs that a function is a generator function?

Answer:

A generator function uses a yield statement instead of a return statement. A generator function will always return a iterable object called generator, where as a normal function can return a string/list/tuple/dict/NoneType

return a iterable object called generator. Where as a normal function can return a string, list, tuple, dict, NoneType ... etc

Question 4)

What is the purpose of a yield statement?

Answer:

The yield statement suspends function's execution and sends a value back to the caller, but retains enough state to enable function to resume where it is left off. When resumed, the function continues execution immediately after the last yield run. This allows its code to produce a series of values over time, rather than computing them at once and sending them back like a list.

Question 5)

What is the relationship between map calls and list comprehensions? Make a comparison and contrast between the two.

Answer:

- **Map function: Suppose we have a function and we want to compute this function for different values in a single line of code . This is where map() function plays its role. map() function returns a map object(which is an iterator) of the results after applying the given function to each item of a given iterable (list, tuple etc.)**
- **If we already have a function defined, it is often good to use map For example, map(sum, myLists) is more neat than [sum(x) for x in myLists]. You gain the elegance of not having to make up a dummy variable (e.g. sum(x) for x... or sum() for ... or sum(readableName) for readableName...) which you have to type twice, just to iterate.**
- **List Comprehension: List Comprehension is a substitute for the lambda function, map(), filter() and reduce()**
- **Comparision :**

1. **List comprehension is more concise and easier to read as compared to map.**
2. **List comprehension allows filtering. In map, we have no such facility For example, to print all even numbers in range of 100, we can write [n for n in range(100) if n%2 == 0]. There is no alternate for it in map.**
3. **List comprehension are used when a list of results is required, where as map only returns a map object and does not return any list.**
4. **List comprehension is faster than map when we need to evaluate expressions that are too long or complicated to express.**
5. **Map is faster in case of calling an already defined function (as no lambda is required).**

Python Basic Assignment - 2

Question1.

What are the two values of the Boolean data type? How do you write them?

Answer:

The two values of the Boolean data type are True and False. We write True by using T in uppercase and the rest in lowercase. Similarly, we write False by using F in uppercase and the rest in lowercase.

Question2.

What are the three different types of Boolean operators?

Answer:

The three different types of Boolean operators are as follows:
a) and
b) or
c) not

In [ ]:

```
### Question 3.
Make a list of each Boolean operator's truth tables
(i.e. every possible combination of Boolean values for the operator and what it evaluate
).

Answer:

    (Boolean_values)        Operator              Evaluated_result
      x       y


    True    True              and                       True
    True    False             and                       False
    False   True              and                       False
    False   False             and                       False

    True    True              or                        True
    True    False             or                        True
    False   True              or                        True
    False   False             or                        False
```

For not Boolean operator:

not True is False
not False is true

Question4.

What are the values of the following expressions?
a.) (5 > 4) and (3 == 5)
b.) not (5 > 4)
c.) (5 > 4) or (3 == 5)
d.) not ((5 > 4) or (3 == 5))
e.) (True and True) and (True == False)
f.) (not False) or (not True)

Answer:

a.) False
b.) False
c.) True
d.) False
e.) False
f.) True

Explaination:

Since,
(5 > 4) is True
(3 == 5) is False
(True == False) is False

therefore,

a.) True and False is False
b.) not True is False
c.) True or False is True
d.) not (True or False) is not True i.e. False
e.) True and False is False
f.) True or False is True

Crosscheck by running the code:

In [1]:

```
(5 > 4) and (3 == 5)
```

Out[1]:

```
False
```

In [2]:

```
not (5 > 4)
```

Out[2]:

```
False
```

In [3]:

```
(5 > 4) or (3 == 5)
```

Out[3]:

```
True
```

In [4]:

```
not ((5 > 4) or (3 == 5))
```

Out[4]:

```
False
```

In [5]:

```
(True and True) and (True == False)
```

Out[5]:

```
False
```

In [6]:

```
(not False) or (not True)
```

Out[6]:

```
True
```

Question5.

What are the six comparison operators?

Answer:

a) < (less than)
b) <= (less than or equal to)
c) == (equal to)
d) > (greater than)
e) >= (greater than or equal to)
f) != (not equal to)

Question6.

How do you tell the difference between the equal to and assignment operators? Describe a condition and when you would use one.

Answer:

== is the equal to operator that compares two values and checks whether the two given values are equal or not and then evaluates to a Boolean value. If the compared values are equal, it returns True. Otherwise it returns False. On the other hand, = is the assignment operator that assigns a value to a variable.

The use of both, the equal to operator and the assignment operator, can be seen in control flow statements.

In [ ]:

```
### Question7.
Identify the three blocks in this code:

spam = 0
if spam == 10:
    print('eggs')
    if spam > 5:
        print('bacon')
    else:
        print('ham')
    print('spam')
print('spam')



Answer:

The three blocks are everything inside the if statement and the lines print('bacon') and
print('ham'), i.e.

print('eggs')
if spam > 5:
    print('bacon')
else:
    print('ham')
print('spam')



Explaination:

There are 3 rules for blocks of code:
1. Blocks begin when indentation starts.
2. Blocks can contain other blocks.
3. Blocks end when indentation decreases to zero or to a container block's indentation.
```

```
spam = 0
if spam == 10:
    print('eggs')                                   # indent increased, block-1
    if spam > 5:                                    # still block-1
        print('bacon')                              # still block-1, indent increased
, block-2 inside block-1
    else:                                           # still block-1, indent decreased
, block-2 ended in line above
        print('ham')                                # still block-1, indent increased
, block-3 inside block-1
    print('spam')                                   # still block-1, indent decreased
, block-3 ended in line above
print('spam')                                       # indent decreased, block-1 ended
in line above
```

Question8.

Write code that prints Hello if 1 is stored in spam, prints Howdy if 2 is stored in spam, and prints Greetings! if anything else is stored in spam.

Answer:

In [7]:

```
spam=int(input("Input any integer: "))
if spam == 1:
    print('Hello')
elif spam == 2:
     print('Howdy')
else:
     print('Greetings!')
```

```
Input any integer
2
Howdy
```

Question9.

If your programme is stuck in an endless loop, what keys you'll press?

Answer:

If our program is stuck in an endless loop, we press CTRL-C to stop or interrupt the program.

Question10.

How can you tell the difference between break and continue?

Answer:

The break statement moves the execution outside and ends the execution to the end of the loop, whereas the continue statement moves the execution to the start of the loop.

Question11.

In a for loop, what is the difference between range(10), range(0, 10), and range(0, 10, 1)?

Answer:

They all return the same output. They return an object that produces a sequence of integers from 0 (lowerbound/start (inclusive)) to 10 (upperbound/stop (exclusive)) with step/jump size of +1. The only differences between them are as follows:

range(10) - Upperbound is explicitly mentioned, but lowerbound=0 and step size = +1 by default.
range(0, 10) - Both lowerbound and upperbound are clearly mentioned, but step size = +1 by default.
range(0, 10, 1) - Lowerbound, upperbound, and step size are explicitly mentioned.

Question12.

Write a short program that prints the numbers 1 to 10 using a for loop. Then write an equivalent program that prints the numbers 1 to 10 using a while loop.

Answer:

Using for loop:

In [8]:

```
for i in range(1,11):
    print(i)
```

```
1
2
3
4
5
6
7
8
9
10
```

Using while loop:

In [9]:

```
i=1
while i<11:
    print(i)
    i=i+1
```

```
1
2
3
4
5
6
7
8
9
10
```

Question13.

If you had a function named bacon() inside a module named spam, how would you call it after importing spam?

Answer:

This function could be called with spam.bacon()

Python Basic Assignment - 20

Question 1. Set the variable test1 to the string 'This is a test of the emergency text system,' and save test1 to a file named test.txt.

Answer:

In [1]:

```
test1 = 'This is a test of the emergency text system,'
with open('test.txt','w') as file:
    file.write(test1)
```

Question 2. Read the contents of the file test.txt into the variable test2. Is there a difference between test 1 and test 2?

Answer:

In [2]:

```
file = open('test.txt', 'r')
test2 = file.readline()

file.close()

if test1 == test2:
    print("There is no difference between test 1 and test 2")
else:
    print("test 1 and test 2 are different from each other")
```

```
There is no difference between test 1 and test 2
```

Question 3. Create a CSV file called books.csv by using these lines:

```
title,author,year
The Weirdstone of Brisingamen,Alan Garner,1960
Perdido Street Station,China Miéville,2000
Thud!,Terry Pratchett,2005
The Spellman Files,Lisa Lutz,2007
Small Gods,Terry Pratchett,1992
```

Answer:

In [3]:

```
import csv
```

```
rows =[ ['title','author','year'],
    ['The Weirdstone of Brisingamen','Alan Garner',1960],
    ['Perdido Street Station','China Miéville',2000],
    ['Thud!','Terry Pratchett',2005],
    ['The Spellman Files','Lisa Lutz',2007],
    ['Small Gods','Terry Pratchett',1992]]

with open('books.csv','w',newline='') as file:
    writer = csv.writer(file)
    writer.writerows(rows)

with open('books.csv','r',newline='') as file:
    for line in file.readlines():
        print(line)
```

```
title,author,year

The Weirdstone of Brisingamen,Alan Garner,1960

Perdido Street Station,China Miéville,2000

Thud!,Terry Pratchett,2005

The Spellman Files,Lisa Lutz,2007

Small Gods,Terry Pratchett,1992
```

Question 4. Use the sqlite3 module to create a SQLite database called books.db, and a table called books with these fields: title (text), author (text), and year (integer).

Answer:

In [4]:

```
import sqlite3
db = sqlite3.connect('books.db')
cursor = db.cursor()
cursor.execute("CREATE TABLE books (title text, author text, year int)")
db.commit()
db.close()
```

Question 5. Read books.csv and insert its data into the book table.

Answer:

In [5]:

```
import sqlite3
import pandas as pd

db = sqlite3.connect('books.db')
df = pd.read_csv('books.csv',encoding='unicode_escape')
df.to_sql('books', db, if_exists='append', index = False)

db.commit()
db.close()
```

Question 6. Select and print the title column from the book table in alphabetical order.

Answer:

In [6]:

```
import sqlite3
conn = sqlite3.connect('books.db')
cursor = conn.cursor()
output = cursor.execute("SELECT title FROM books ORDER BY title ASC")
for ele in output:
    print(ele[0])
conn.commit()
conn.close()
```

```
Perdido Street Station
Small Gods
The Spellman Files
The Weirdstone of Brisingamen
Thud!
```

Question 7. From the book table, select and print all columns in the order of publication.

Answer:

In [7]:

```
import sqlite3
conn = sqlite3.connect('books.db')
cursor = conn.cursor()
output = cursor.execute("SELECT * FROM books ORDER BY year")
for ele in output:
    print(ele)
conn.commit()
conn.close()
```

```
('The Weirdstone of Brisingamen', 'Alan Garner', 1960)
('Small Gods', 'Terry Pratchett', 1992)
('Perdido Street Station', 'China Miéville', 2000)
('Thud!', 'Terry Pratchett', 2005)
('The Spellman Files', 'Lisa Lutz', 2007)
```

In [9]:

```
import sqlite3
import pandas as pd

conn = sqlite3.connect('books.db')
cursor = conn.cursor()
cursor.execute('select * from books ORDER by year')

df = pd.DataFrame(cursor.fetchall(), columns=['title','author','year'])
df
```

Out[9]:

	title	author	year
0	The Weirdstone of Brisingamen	Alan Garner	1960
1	Small Gods	Terry Pratchett	1992
2	Perdido Street Station	China Miéville	2000

3	Thud!	Terry Pratchett	2005
4	The Spellman Files	Lisa Lutz	2007

Question 8. Use the sqlalchemy module to connect to the sqlite3 database books.db that you just made in exercise 6.

Answer:

In [10]:

```
import sqlalchemy
engine = sqlalchemy.create_engine("sqlite:///books.db")
rows = engine.execute('select * from books')
for i in rows:
    print(i)
```

```
('The Weirdstone of Brisingamen', 'Alan Garner', 1960)
('Perdido Street Station', 'China Miéville', 2000)
('Thud!', 'Terry Pratchett', 2005)
('The Spellman Files', 'Lisa Lutz', 2007)
('Small Gods', 'Terry Pratchett', 1992)
```

Question 9. Install the Redis server and the Python redis library (pip install redis) on your computer. Create a Redis hash called test with the fields count (1) and name ('Fester Bestertester'). Print all the fields for test.

Answer:

In [11]:

```
! python -m pip install redis

# or try this: !pip install redis
```

```
Collecting redis
  Downloading redis-4.1.4-py3-none-any.whl (175 kB)
Requirement already satisfied: packaging>=20.4 in c:\users\utkarsh\anaconda3\lib\site-pac
kages (from redis) (21.0)
Collecting deprecated>=1.2.3
  Downloading Deprecated-1.2.13-py2.py3-none-any.whl (9.6 kB)
Requirement already satisfied: wrapt<2,>=1.10 in c:\users\utkarsh\anaconda3\lib\site-pack
ages (from deprecated>=1.2.3->redis) (1.12.1)
Requirement already satisfied: pyparsing>=2.0.2 in c:\users\utkarsh\anaconda3\lib\site-pa
ckages (from packaging>=20.4->redis) (3.0.4)
Installing collected packages: deprecated, redis
Successfully installed deprecated-1.2.13 redis-4.1.4
```

In [25]:

```
import redis
conn = redis.Redis()
conn.delete('test')
conn.hmset('test', {'count': 1, 'name': 'Fester Bestertester'})
conn.hgetall('test')
```

```
C:\Users\Utkarsh\AppData\Local\Temp/ipykernel_8704/19521559.py:4: DeprecationWarning: Red
is.hmset() is deprecated. Use Redis.hset() instead.
  conn.hmset('test', {'count': 1, 'name': 'Fester Bestertester'})
```

Out[25]:

```
{b'count': b'1', b'name': b'Fester Bestertester'}
```

Question 10. Increment the count field of test and print it.

Answer:

In [26]:

```
conn.hincrby('test','count', 3)
```

Out[26]:

```
4
```

Python Basic Assignment - 21

Question 1.

Add the current date to the text file today.txt as a string.

Answer:

In [1]:

```
from datetime import datetime

with open("today.txt", "w") as file:
    file.write(datetime.now().strftime("%d/%m/%Y"))
    file.close()

file = open("today.txt", "r")
print(file.read())
file.close()
```

```
27/01/2022
```

Question2.

Read the text file today.txt into the string today_string

Answer:

In [2]:

```
file = open("today.txt", "r")
today_string = file.read()
print(today_string)
file.close()
```

```
27/01/2022
```

Question3.

Parse the date from today_string.

Answer:

In [3]:

```
from datetime import datetime
format = '%d/%m/%Y'
parsed_data = datetime.strptime(today_string,format)

print(parsed_data)
```

```
2022-01-27 00:00:00
```

Question4.

List the files in your current directory

Answer:

In [4]:

```
import os


for folders, subfolders, files in os.walk(os.getcwd()):
    for file in files:
        print(file)
```

```
Python Basic Assignment - 1.ipynb
Python Basic Assignment - 10.ipynb
Python Basic Assignment - 11.ipynb
Python Basic Assignment - 15.ipynb
Python Basic Assignment - 16.ipynb
Python Basic Assignment - 17.ipynb
Python Basic Assignment - 18.ipynb
Python Basic Assignment - 19.ipynb
Python Basic Assignment - 2.ipynb
Python Basic Assignment - 21.ipynb
Python Basic Assignment - 21.py.bak
Python Basic Assignment - 22.ipynb
Python Basic Assignment - 23.ipynb
Python Basic Assignment - 24.ipynb
Python Basic Assignment - 25.ipynb
Python Basic Assignment - 3.ipynb
Python Basic Assignment - 4.ipynb
Python Basic Assignment - 5.ipynb
Python Basic Assignment - 6.ipynb
Python Basic Assignment - 9.ipynb
PythonBasicAssignmentTwentyOneMultiProcessing.py
PythonBasicAssignmentTwentyOneMultiProcessing.py.bak
today.txt
zoo.py
Python Basic Assignment - 16-checkpoint.ipynb
Python Basic Assignment - 17-checkpoint.ipynb
Python Basic Assignment - 18-checkpoint.ipynb
Python Basic Assignment - 19-checkpoint.ipynb
Python Basic Assignment - 21-checkpoint.ipynb
Untitled-checkpoint.ipynb
zoo.py-checkpoint.ipynb
PythonBasicAssignmentTwentyOneMultiProcessing.cpython-39.pyc
zoo.cpython-39.pyc
```

Question5.

Create a list of all of the files in your parent directory (minimum five files should be available).

Answer:

In [5]:

```
import os
os.listdir()
```

Out[5]:

```
['.ipynb_checkpoints',
 'Python Basic Assignment - 1.ipynb',
 'Python Basic Assignment - 10.ipynb',
 'Python Basic Assignment - 11.ipynb',
 'Python Basic Assignment - 15.ipynb',
 'Python Basic Assignment - 16.ipynb',
 'Python Basic Assignment - 17.ipynb',
 'Python Basic Assignment - 18.ipynb',
 'Python Basic Assignment - 19.ipynb',
 'Python Basic Assignment - 2.ipynb',
 'Python Basic Assignment - 21.ipynb',
 'Python Basic Assignment - 21.py.bak',
 'Python Basic Assignment - 22.ipynb',
 'Python Basic Assignment - 23.ipynb',
 'Python Basic Assignment - 24.ipynb',
 'Python Basic Assignment - 25.ipynb',
 'Python Basic Assignment - 3.ipynb',
 'Python Basic Assignment - 4.ipynb',
 'Python Basic Assignment - 5.ipynb',
 'Python Basic Assignment - 6.ipynb',
 'Python Basic Assignment - 9.ipynb',
 'PythonBasicAssignmentTwentyOneMultiProcessing.py',
 'PythonBasicAssignmentTwentyOneMultiProcessing.py.bak',
 'today.txt',
 'zoo.py',
 '__pycache__']
```

Question6.

Use multiprocessing to create three separate processes. Make each one wait a random number of seconds between one and five, print the current time, and then exit.

Answer:

In [6]:

```
import multiprocessing
import time
import random
from datetime import datetime

def First_Process():
    print(f'Start_Time_For_First_Process ----> {datetime.now()}')
    time.sleep(random.randrange(1,5))
    print(f'End_Time_For_First_Process ----> {datetime.now()}')

print()

def Second_Process():
    print(f'Start_Time_For_First_Process ----> {datetime.now()}')
    time.sleep(random.randrange(1,5))
    print(f'End_Time_For_First_Process ----> {datetime.now()}')

print()
```

```
def Third_Process():
    print(f'Start_Time_For_Third_Process ----> {datetime.now()}')
    time.sleep(random.randrange(1,5))
    print(f'End_Time_For_Third_Process ----> {datetime.now()}')

if __name__ == "__main__":
    p1 = multiprocessing.Process(target = First_Process)
    p2 = multiprocessing.Process(target = First_Process)
    p3 = multiprocessing.Process(target = Third_Process)

    p1.start()
    p2.start()
    p3.start()

    p1.join()
    p2.join()
    p3.join()
```

```
!python PythonBasicAssignmentTwentyOneMultiProcessing.py
```

```
Start_Time_For_Third_Process ----> 2022-01-27 08:28:42.129917
End_Time_For_Third_Process ----> 2022-01-27 08:28:44.138796


Start_Time_For_First_Process ----> 2022-01-27 08:28:42.264602
End_Time_For_First_Process ----> 2022-01-27 08:28:45.273739


Start_Time_For_First_Process ----> 2022-01-27 08:28:42.264602
End_Time_For_First_Process ----> 2022-01-27 08:28:45.289251
```

Question7.

Create a date object of your day of birth.

Answer:

In [7]:

```
from datetime import datetime

Date_Of_Birth = input("Enter the date in dd/mm/yyyy format : ")

Proper_Format_For_Date_Of_Birth = datetime.strptime(Date_Of_Birth, '%d/%m/%Y')

Date_Object_For_Day_Of_Birth = Proper_Format_For_Date_Of_Birth.day

Date_Object_For_Day_Of_Birth
```

```
Enter the date in dd/mm/yyyy format : 13/02/1998
```

Out[7]:

```
13
```

Question8.

What day of the week was your day of birth?

Answer:

In [8]:

```
from datetime import datetime
import calendar

Date_Of_Birth = input("Enter the date in dd/mm/yyyy format : ")

Proper_Format_For_Date_Of_Birth = datetime.strptime(Date_Of_Birth , "%d/%m/%Y")

weekday_index = Proper_Format_For_Date_Of_Birth.weekday()

Day_Of_Birth = calendar.day_name[weekday_index]

print(f"The weekday on {Date_Of_Birth} is: {Day_Of_Birth}")
```

```
Enter the date in dd/mm/yyyy format : 13/02/1998
The weekday on 13/02/1998 is: Friday
```

Question9.

When will you be (or when were you) 10,000 days old?

Answer:

In [9]:

```
from datetime import timedelta

Date_Of_Birth = input("Enter the date in dd/mm/yyyy format : ")
Proper_Format_For_Date_Of_Birth = datetime.strptime(Date_Of_Birth , "%d/%m/%Y")
Date_after_10000_days_of_my_Date_Of_Birth = Proper_Format_For_Date_Of_Birth + timedelta(days=10000)

Date_after_10000_days_of_my_Date_Of_Birth
```

```
Enter the date in dd/mm/yyyy format : 13/02/1998
```

Out[9]:

```
datetime.datetime(2025, 7, 1, 0, 0)
```

Python Basic Assignment - 22

Question 1.

What is the result of the code, and explain?

> > **X = 'iNeuron'**
> > **def func():**
> > **print(X)**
> >
> > **func()**

Answer:

In [1]:

```
>>> X = 'iNeuron'

>>> def func():
    print(X)

>>> func()
```

```
iNeuron
```

Explaination:

The result of the given code is iNeuron. It is because the func() method intially calls print() function and looks for the variable X in its local scope, but since there is no local variable X, the func() method returns the value of global variable X i.e. the string, 'iNeuron'.

Question 2.

What is the result of the code, and explain?

```
X = 'iNeuron'
def func():
X = 'NI!'

func()
print(X)
```

Answer:

In [2]:

```
X = 'iNeuron'

def func():
    X = 'NI!'

func()
print(X)
```

```
iNeuron
```

Explaination:

The result of the given code is iNeuron. It is because the func() method just stores the value of the variable X in its local scope, but print(X) function has been called outside the func() method, therefore, it will return the value of the variable X stored in the global scope.

Question 3.

What does this code print, and why?

```
X = 'iNeuron'
def func():
X = 'NI'
print(X)

func()
print(X)
```

Answer:

In [3]:

```
X = 'iNeuron'
def func():
    X = 'NI'
    print(X)

func()
print(X)
```

```
NI
iNeuron
```

Explaination:

The given code has two results, NI and iNeuron. It is because the func() method first stores the value of the variable X in its local scope, and then calls print(X) function, which returns the value of the variable X in its local scope i.e. the string 'NI'. print(X) function has also been called once outside the func() method, therefore, it will return the value of the variable X stored in the global scope, i.e. the string, 'iNeuron'.

Question 4.

What output does this code produce? Why?

```
X = 'iNeuron'
def func():
global X
X = 'NI'

func()
print(X)
```

Answer:

In [4]:

```
X = 'iNeuron'
def func():
    global X
    X = 'NI'
```

```
func()
print(X)
```

```
NI
```

Explaination:

The result of the given code is NI. First, the method func() has been called, and since the global keyword allows the variable, X to be accessible in the current scope, and since we are using global keyword inside the method func(), it directly access the value of the variable in X, present in global scope and then assign the value, NI to the variable X. Therefore, the result of the code is NI after the function print(x) is called outside the method func().

Question 5.

What about this code—what's the output, and why?

> > > **X = 'iNeuron'**
> > > **def func():**
> > > **X = 'NI'**
> > > **def nested():**
> > > **print(X)**
> > > **nested()**
> > >
> > > **func()**
> > > **X**

Answer:

In [5]:

```
X = 'iNeuron'

def func():
    X = 'NI'
    def nested():
        print(X)
    nested()

func()
X
```

```
NI
```

Out[5]:

```
'iNeuron'
```

Explaination:

The given code has two outputs, NI and 'iNeuron'.

Reason for the output NI:
If a function(or method) wants to access a variable and if it is not available in its localscope, then it looks for the variable in its global scope. Similarly here also the nested() method looks for the variable X in its global scope, which is nothing but the local scope of the method func(). Hence the output is NI when func() is called.

Reason for the output 'iNeuron':

Reason for the output 'iNeuron':
It will simply look for the variable X in its global scope, which has been assigned the string value, 'iNeuron'. Hence, the result is 'iNeuron'.

Question 6.

How about this code: what is its output in Python 3, and explain?

> > > def func():
> > > X = 'NI'
> > > def nested():
> > > nonlocal X
> > > X = 'Spam'
> > > nested()
> > > print(X)
> > >
> > > func()

Answer:

In [6]:

```
def func():
    X = 'NI'
    def nested():
        nonlocal X
        X = 'Spam'
    nested()
    print(X)

func()
```

```
Spam
```

Explaination:

The output of the code is Spam.The keyword, nonlocal, in python is used to declare a variable as not local. Hence the statement X = "Spam" is modified in the global scope. Hence the output of print(X) statement is Spam, and therefore, the result of the method func() is Spam.

Python Basic Assignment - 23

Question 1.

What is the result of the code, and why?

> > > def func(a, b=6, c=8):
> > > print(a, b, c)
> > > func(1, 2)

Answer:

In [1]:

```
def func(a, b= 6,c = 8):
    print(a,b,c)

func(1,2)
```

```
1 2 8
```

Explanation:

The result of the above code is 1 2 8 because the function uses the default value of c ie 8 which is provided at the time of declaration and the default value for b has been overwrittten while assigning the value of b during the function call. The value of a has been assigned during the function call.

Question 2.

What is the result of this code, and why?

> **def func(a, b, c=5):**
> **print(a, b, c)**
> **func(1, c=3, b=2)**

Answer:

In [2]:

```
def func(a,b,c=5):
    print(a,b,c)

func(1,c=3,b=2)
```

```
1 2 3
```

Explanation:

The result of the above code is 1 2 3 because the function will use default values only when value for any argument is not provided during function call.

If argument name is mentioned while doing a function call, the order of arguments is also ignored by the python interpreter.

Question 3.

How about this code: what is its result, and why?

> > > > **def func(a, *pargs):**
> > > > **print(a, pargs)**
> > > > **func(1, 2, 3)**

Answer:

In [3]:

```
def func(a, *pargs):
    print(a,pargs)

func(1,2,3)
```

```
1 (2, 3)
```

Explaination:

The result of the code is 1 (2,3). ***pargs stands for variable length arguments. This format is used when we are not sure about the number of arguments to be passed to a function. All the values under this argument will be stored in a tuple. Since argument a is not a part of the*** **parg argument, hence, it is not stored in a tuple. And the rest arguments will automatically be considered as the values for the argument *parg, therefore, they have been stored separately in a tuple.**

Question 4.

What does this code print, and why?

> > > > **def func(a, **kargs):**
> > > > **print(a, kargs)**
> > > > **func(a=1, c=3, b=2)**

Answer:

In [4]:

```
def func(a, **kargs):
    print(a, kargs)

func(a=1, c=3, b=2)
```

```
1 {'c': 3, 'b': 2}
```

Explaination:

The result of the above code is 1 {'c': 3, 'b': 2}. **args stands for variable length keyword arguments. This format is used when we want pass key value pairs as input to a function. All these key value pairs will be stored in a dictionary. Similarly, the values for keys 'b' and 'c' have been stored in the dictionary separately.

Question 5.

What gets printed by this, and explain?

> > > **def func(a, b, c=8, d=5): print(a, b, c, d)**
> > > **func(1, *(5, 6))**

Answer:

In [5]:

```
def func(a, b, c=8, d=5): print(a, b, c, d)

func(1, *(5, 6))
```

```
1 5 6 5
```

Explaination:

The output of the above is 1 5 6 5. This is because the value for a is provided explicitly whereas for the values for arguments b and c, the function will expand the *(5,6) and consider the value of b as 5 and value of c as 6. Since the default value of d is provided explicitly while function declaration, therefore, value of d will be 5. However, it is not recommended to use the feature of positional arguments in between. It is recommended to use the feature of positional arguments at the end.

Question 6.

what is the result of this, and explain?

> > > **def func(a, b, c): a = 2; b[0] = 'x'; c['a'] = 'y'**
> > > **l = 1; m = [1]; n = {'a':0}**
> > > **func(l, m, n)**
> > >
> > > **l, m, n**

Answer:

In [6]:

```
def func(a, b, c): a = 2; b[0] = 'x'; c['a'] = 'y'
l = 1; m = [1]; n = {'a':0}

func(l, m, n)
l, m, n
```

Out[6]:

```
(1, ['x'], {'a': 'y'})
```

Explaination:

The output of above code is 1, ['x'], {'a': 'y'}

The output of above code is 1, ['x'], {'a': 'y'}

Eventhough Python gives importance to indentation. its provides a facility to declare an entire function in one single line. where statements in a function body are sepereated by ;

Here in the code, an integer, a list and a dictionary are passed as argument, and list and dictionary are mutable. Here the list m and parameter b point to the same list in the memory location where as dict n and c point to the same memory location. Therefore, any updates to these list or dictionary will not be updated in the memory location. So, we will be able to see the value which has been declared while defining the function.

l = 1 is an integer variable and immutable.
m is list and mutable.
n is dictionary and mutable.

Python Basic Assignment - 24

Question 1.

What is the relationship between def statements and lambda expressions ?

Answer:

def statement is used to create a normal function, where as lambda expressions are used to create anonymous functions. which can be assigned to a variable and can be called using the variable later in function.

Lambda's body is a single expression and not a block of statements like def statement. The lambda expression's body is similar to what we put in a def body's return statement. We simply type the result as an expression instead of explicitly returning it. Because it is limited to an expression, a lambda is less general than a def statement.

Question2.

What is the benefit of lambda?

Answer:

The following are some of the benefits of lambda expressions:

1.) Can be used to create nameless/anonymous functions inside some complex functions if we are planning to use it only once.
2.) Moderate to small functions can be created in a single line.
3.) Fuctions created using lambda expressions can be assigned to a variable and can be used by simply calling the variable.

Question3.

Compare and contrast map, filter, and reduce.

Answer:

The differences between map, filter and reduce are:

1.) map(): The map() function is a type of higher-order. This function takes another function as a parameter along

with a sequence of iterables and returns an output after applying the function to each iterable present in the sequence.
2.) filter(): The filter() function is used to create an output list consisting of values for which the function returns true.
3.) reduce(): The reduce() function, as the name describes, applies a given function to the iterables and returns a single value

In [1]:

```
from functools import reduce

# map function
print('Map ---->',list(map(lambda x:x+x, [1,2,3,4])))

# fitler function
print('Filter ---->',list(filter(lambda x:x%2 !=0, [1,2,3,4])))

# reduce function
print('Reduce ---->',reduce(lambda x,y:x+y, [1,2,3,4,5,6]))
```

```
Map ----> [2, 4, 6, 8]
Filter ----> [1, 3]
Reduce ----> 21
```

Question4.

What are function annotations, and how are they used?

Answer:

Function annotation is the standard way to access the metadata with the arguments and the return value of the function. These are nothing but some random and optional Python expressions that get allied to different parts of the function. They get evaluated only during the compile-time and have no significance during the run-time of the code. They do not have any significance or meaning associated with them until accessed by some third-party libraries. They are used to type check the functions by declaring the type of the parameters and the return value for the functions. The string-based annotations help us to improve the help messages.

- **Syntax :**

def func(a: 'int') -> 'int': pass

- **Annotations for simple parameters:**

def func(x: 'float'=10.8, y: 'argument2'):

In the above code the argument, 'x' of the function func, has been annotated to float data type and the argument 'y' has a string-based annotation. The argument can also be assigned to a default value using a '=' symbol followed by the default value. These default values are optional to the code.

- **Annotations for return values:**

def func(a: expression) -> 'int':

The annotations for the return value is written after the '->' symbol.

In [2]:

```
def fib(n:'float', b:'int')-> 'result':
    pass

print(fib.__annotations__)
```

```
{'n': 'float', 'b': 'int', 'return': 'result'}
```

Question5.

What are recursive functions, and how are they used?

Answer:

A recursive function is a function that calls itself during its execution. This means that the function will continue to call itself and repeat its behavior until some condition is met to return a result.

Question6.

What are some general design guidelines for coding functions?

Answer:

Some of the general design guidelines for coding functions are:

1. **Use 4-space indentation and no tabs to increase the code readability**
2. **Use docstrings to explain the functionality of the function**
3. **Wrap linethat they don't exceed 79 characters**
4. **Use of regular and updated comments are valuable to both the coders and users**
5. **Use of trailing commas : in case of tuple -> ('good',)**
6. **Use Python's default UTF-8 or ASCII encodings and not any fancy encodings**
7. **Naming Conventions 8.Characters that should not be used for identifiers : 'l' (lowercase letter el), 'O' (uppercase letter oh), 'I' (uppercase letter eye) as single character variable names as these are similar to the numerals one and zero.**
8. **Don't use non-ASCII characters in identifiers**
9. **Name your classes and functions consistently**
10. **While naming of function of methods always use self for the first argument**
11. **Avoid using or limited use of global variables**
12. **Avoid using digits while choosing a variable name**
13. **Try to use a name for the function which conveys the purpose of the function**
14. **Local variables should be named using camelCase format** `(ex: localVariable)` **whereas Global variables names should be using PascalCase** `(ex:GlobalVariable)`**.**
15. **Constant should be represented in allcaps** `(ex:CONSTANT)`**.**

Question7.

Name three or more ways that functions can communicate results to a caller.

Answer:

Some of the ways in which a function can communicate with the calling function is:

1. **print**
2. **return**
3. **yield**

Python Basic Assignment - 3

Question1.

Why are functions advantageous to have in your programs?

Answer:

Functions are advantageous to have in our programs primarily because they reduce the need for duplicate code. Thus, they make programs shorter, easier to read, and easier to update. Debugging of the programs also becomes faster.

Question2.

When does the code in a function run: when it's specified or when it's called?

Answer:

The code in a function runs when the function is called, not when the function is defined.

Question3.

What statement creates a function?

Answer:

The def statement creates a function.

Question4.

What is the difference between a function and a function call?

Answer:

A function consists of the def statement and the code in its def clause, whereas, a function call is somethong that moves the program execution into the function, and the function call evaluates to the return value of the function.

Question5.

How many global scopes are there in a Python program? How many local scopes?

Answer:

There is one global scope, and a local scope that is created whenever a function is called.

Question6

Question6.

What happens to variables in a local scope when the function call returns?

Answer:

When a function call returns, the local scope is destroyed, and all the variables in it are forgotten.

Question7.

What is the concept of a return value? Is it possible to have a return value in an expression?

Answer:

A return value is the value that a function call evaluates to. Like any value, a return value can be used as part of an expression.

Question8.

If a function does not have a return statement, what is the return value of a call to that function?

Answer:

If there is no return statement for a function, its return value is None.

Question9.

How do you make a function variable refer to the global variable?

Answer:

A global statement will force a variable in a function to refer to the global variable.

Question10.

What is the data type of None?

Answer:

The data type of None is NoneType.

Question11.

What does the sentence import areallyourpetsnamederic do?

Answer:

That import statement imports a module named areallyourpetsnamederic, although it isn't a real Python module.

Question12.

If you had a bacon() feature in a spam module, what would you call it after importing spam?

Answer:

This function could be called by writing the name of the module first and then the name of the feature separated from the module name by a dot as follows:

spam.bacon()

Question13.

What can you do to save a programme from crashing if it encounters an error?

Answer:

We can place the line of code that might cause an error in a try clause.

Question14.

What is the purpose of the try clause? What is the purpose of the except clause?

Answer:

The code that could potentially cause an error goes in the try clause and thus, the try block allows us to test a block of code for errors i.e the code inside the try block will execute when there is no error in the program.

Whereas The purpose of the except clause is to execute code inside the except block whenever the program encounters some error in the preceding try block.

Python Basic Assignment - 9

Question1.

To what does a relative path refer?

Answer:

A relative path refer to the path to some file or folder with respect to the current working directory.

For example: If absolute path is C:/users/admin/INEURON/FSDS.txt, and the present working directory is C:/users/admin/ , then the relative path to FSDS.txt would be: INEURON/FSDS.txt.

NOTE:
Present Working Directory + relative path = absolute path

Question2.

What does an absolute path start with your operating system?

Answer:

Absolute paths start with the root folder, such as / or C:

Question3.

What do the functions os.getcwd() and os.chdir() do?

Answer:

os.getcwd() helps us to know the current working directory. os.chdir() helps us to change the current working directory.

Question4.

What are the . and .. folders?

Answer:

The . folder is the current folder. The .. is the parent folder.

Question5.

In C:\bacon\eggs\spam.txt, which part is the dir name, and which part is the base name?

Answer:

C:\bacon\eggs is the dir name, while spam.txt is the base name.

Question6.

What are the three “mode” arguments that can be passed to the open() function?

Answer:

The string 'r' for read mode, 'w' for write mode, and 'a' for append mode are the three “mode” arguments that can be passed to the open() function

Question7.

What happens if an existing file is opened in write mode?

Answer:

If an existing file is opened in write mode, then the existing file is erased and completely overwritten.

Question8.

How do you tell the difference between read() and readlines()?

Answer:

The read() method returns the file's entire contents as a single string value, while the readlines() method returns a list of strings, where each string is a line from the file's contents.

Question9.

What data structure does a shelf value resemble?

Answer:

A shelf value resembles a dictionary value; it has keys and values, along with keys() and values() methods that work similarly to the dictionary methods of the same names.

Python Basic Assignment - 10

Question1.

How do you distinguish between shutil.copy() and shutil.copytree()?

Answer:

The shutil.copy() function copies a single file, while shutil.copytree() copies an entire folder, along with all of its contents.

Question2.

What function is used to rename files??

Answer:

The os.rename() function can be used to rename a file. It takes two arguments, the first one is the old name and the second one is the new name. The shutil.move() function can also be used for renaming files as well as moving them.

Question3.

What is the difference between the delete functions in the send2trash and shutil modules?

Answer:

The send2trash function moves a file or folder to the recycle bin, while shutil function permanently deletes files and folders.

Question4.

ZipFile objects have a close() method just like File objects' close() method. What ZipFile method is equivalent to File objects' open() method?

Answer:

The zipfile.ZipFile() function is equivalent to the open() function; the first argument is the filename, and the second argument is the mode to open the ZIP file in read mode, write mode, or append mode.

Question5.

Create a programme that searches a folder tree for files with a certain file extension (such as .pdf or .jpg). Copy these files from whatever location they are in to a new folder.

Answer:

In [1]:

```
def SEARCH_AND_COPY():

    """
    It is a function that searches a folder tree for files with a certain file extension
    (such as .pdf or .jpg) and Copy these files from whatever location they are in to a n
ew folder.

    It takes following three inputs:

    It takes Source_Path as input (Follow double forward slash to give input).
    It takes Type_Of_File_Extension as input.
    It takes Destination_Path as input (Follow double forward slash to give input).

    """

    import os, shutil

    Source_Path = input("Enter the absolute path of the source directory that you want to
be searched:   ")
    Type_Of_File_Extension = input("Enter the type of file to copy (such as .pdf or .jpg)
:    ").lower()
    Destination_Path = input("Enter the absolute path of the destination directory where
you want the searched files to be copied:   ")

    for FolderName, SubFolders, FileNames in os.walk(Source_Path):

        for FileName in FileNames:

            if FileName.lower().endswith(Type_Of_File_Extension):

                Source_Path_Files_To_Be_Copied = os.path.join(FolderName, FileName)
                shutil.copy(Source_Path_Files_To_Be_Copied, Destination_Path)

            else:
                continue


SEARCH_AND_COPY()
```

```
Enter the absolute path of the source directory that you want to be searched:   C:\\Users
\\Utkarsh\\Desktop\\Sleepless in Seattle (1993) (1080p BluRay x265 HEVC 10bit AAC 5.1 Tig
ole)
Enter the type of file to copy (such as .pdf or .jpg):    .jpg
Enter the absolute path of the destination directory where you want the searched files to
be copied:   C:\\Users\\Utkarsh\\Desktop\\New folder (3)
```

Python Basic Assignment - 4

Question1.

What exactly is []?

Answer:

[] is an empty list value that contains no elements in it.

In [ ]:

```
### Question2.
In a list of values stored in a variable called spam, how would you assign the value "he
llo" as the
third value? (Assume [2, 4, 6, 8, 10] are in spam.)

Answer:
Since, spam = [2, 4, 6, 8, 10] and we want to assign the value "hello" as the third valu
e.
Therefore, "hello" must have an index of 2 if we are opting forward indexing method or
an index of -3 if we are opting for backward indexing method, and assignment can be done
as follows:

spam[2] = "hello"

or

spam[-3] = "hello"

Crosscheck by running program:
```

In [1]:

```
spam = [2, 4, 6, 8, 10]
```

In [2]:

```
spam[2] = "hello"
```

In [3]:

```
spam
```

Out[3]:

```
[2, 4, 'hello', 8, 10]
```

In [4]:

```
spam = [2, 4, 6, 8, 10]
```

In [5]:

```
spam
```

Out[5]:

```
[2, 4, 6, 8, 10]
```

In [6]:

```
spam[-3] = "hello"
```

In [7]:

```
spam
```

Out[7]:

```
[2, 4, 'hello', 8, 10]
```

In [ ]:

```
Let's pretend the spam includes the list ['a','b','c','d'] for the next three queries.

### Question3.
What is the value of spam[int(int('3' * 2) / 11)]?

### Question4.
What is the value of spam[-1]?

### Question5.
What is the value of spam[:2]?




spam = ['a','b','c','d']



Answer3:

spam[int(int('3' * 2) / 11)] = spam[int(33 / 11)] = spam [3] = 'd'



Answer4:

spam[-1] = 'd'



Answer5:

spam[:2] = spam[0:2:1] = ['a', 'b']



Crosscheck by running the program:
```

In [8]:

```
spam = ['a','b','c','d']
```

In [9]:

```
spam[int(int('3' * 2) / 11)]
```

Out[9]:

```
'd'
```

In [10]:

```
spam[-1]
```

Out[10]:

```
'd'
```

In [11]:

```
spam[:2]
```

Out[11]:

```
['a', 'b']
```

In [ ]:

```
Let's pretend bacon has the list [3.14, 'cat', 11, 'cat', True] for the next three questi
ons.

### Question6.
What is the value of bacon.index('cat')?

### Question7.
How does bacon.append(99) change the look of the list value in bacon?

### Question8.
How does bacon.remove('cat') change the look of the list in bacon?









bacon = [3.14, 'cat', 11, 'cat', True]


Answer6:

bacon.index('cat') = 1, however there is an another string in the list,'cat' with an ind
ex of 3,
but index() function returns the first occurrence of an item.



Answer7:

bacon.append(99) will append 99 to the end of the list and thus, the list becomes as fol
lows:
[3.14, 'cat', 11, 'cat', True, 99]



Answer8:

bacon.remove('cat') will remove the first occurence of the string 'cat' from the list and
thus, the list becomes as follows:
[3.14, 11, 'cat', True]



Crosscheck by running the program:
```

In [12]:

```
bacon = [3.14, 'cat', 11, 'cat', True]
```

In [13]:

```
bacon.index('cat')
```

Out[13]:

```
1
```

In [14]:

```
bacon.append(99)
```

In [15]:

```
bacon
```

Out[15]:

```
[3.14, 'cat', 11, 'cat', True, 99]
```

In [16]:

```
bacon = [3.14, 'cat', 11, 'cat', True]
```

In [17]:

```
bacon
```

Out[17]:

```
[3.14, 'cat', 11, 'cat', True]
```

In [18]:

```
bacon.remove('cat')
```

In [19]:

```
bacon
```

Out[19]:

```
[3.14, 11, 'cat', True]
```

Question9.

What are the list concatenation and list replication operators?

Answer:

The operator for list concatenation is + , while operator for list replication is *

Question10.

What is difference between the list methods append() and insert()?

Answer:

If we use append(), the default location of the value to be appended is at the last of the list. On the other hand, if we use insert(), we can add values anywhere in the list, because it takes both, a value and a specific location (index) of the value, as input.

Question11.

What are the two methods for removing items from a list?

Answer:

Following are the two methods for removing items from a list:

a) The del statement can be used to delete an item in the list by providing index as input.
b) The remove() list method can be used to remove an item from a list by providing the value of the item as input.

Question12.

Describe how list values and string values are identical.

Answer:

Both lists and strings can:

a.) be passed to len(),
b.) have indexes and slices,
c.) be used in for loops,
d.) be concatenated or replicated, and
e.) be used with the in and not in operators.

Question13.

What's the difference between tuples and lists?

Answer:

Lists are mutable; they can have values added, removed, or changed. On the other hand, tuples are immutable; they cannot be changed at all after their creation.
Also, lists use the square brackets, [], whereas tuples are written using parentheses, ().

Question14.

How do you type a tuple value that only contains the integer 42?

Answer:

To type a tuple value that only contains the integer 42, the trailing comma is compulsory and can be written as follows:
(42,)

Question15.

How do you get a list value's tuple form? How do you get a tuple value's list form?

Answer:

We can get a list value's tuple form by using the tuple() functions, and we can get a tuple value's list form by using the list() function.

using the list() function.

Question16.

Variables that "contain" list values are not necessarily lists themselves. Instead, what do they contain?

Answer:

They contain references (reserved memory location) to list values.

Question17.

How do you distinguish between copy.copy() and copy.deepcopy()?

Answer:

The copy.copy() function will do a shallow copy of a list, while the copy.deepcopy() function will do a deep copy of a list. That is, only copy.deepcopy() will duplicate any lists inside the list.

Python Basic Assignment - 5

Question1.

What does an empty dictionary's code look like?

Answer:

An empty dictionary's code look like {} i.e. two curly brackets.

Question2.

What is the value of a dictionary value with the key 'foo' and the value 42?

Answer:

If {'foo': 42} is it itself a dictionary value, then its value would be 42.

Question3.

What is the most significant distinction between a dictionary and a list?

Answer:

The items stored in a dictionary are unordered, whereas the items in a list are ordered.

In [ ]:

```
### Question4.
What happens if you try to access spam['foo'] if spam is {'bar': 100}?

Answer:

If spam is {'bar': 100}, and if we try to access spam['foo'], then we get a KeyError,
because there is no key named as 'foo' in the given dictionary.

Crosscheck by running the program:
```

In [1]:

```
spam = {'bar': 100}
```

In [2]:

```
spam
```

Out[2]:

```
{'bar': 100}
```

In [3]:

```
spam['foo']
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_2680/3999281786.py in <module>
----> 1 spam['foo']

KeyError: 'foo'
```

Question5.

If a dictionary is stored in spam, what is the difference between the expressions 'cat' in spam and 'cat' in spam.keys()?

Answer:

There is no difference. The in operator in both the cases will check whether 'cat' exist as a key in the dictionary.

Question6.

If a dictionary is stored in spam, what is the difference between the expressions 'cat' in spam and 'cat' in spam.values()?

Answer:

The expression 'cat' in spam will check whether a 'cat' key exist in the dictionary, whereas the expression 'cat' in spam.values() will check whether a value 'cat' exist for one of the keys in spam.

In [ ]:

```
### Question7.
What is a shortcut for the following code?

if 'color' not in spam:
spam['color'] = 'black'
```

```
Answer:

spam.setdefault('color', 'black')
```

Question8.

How do you "pretty print" dictionary values using which module and function?

Answer:

We "pretty print" dictionary values using pprint module and pprint() function as follows:
pprint.pprint()

Python Basic Assignment - 6

Question1.

What are escape characters, and how do you use them?

Answer:

Escape characters represent characters in string values that would otherwise be difficult or impossible to type into code. In Python strings, the backslash “\” is a special character, also called the “escape” character can be used as a prefix just before the character we want to escape.

Question2.

What do the escape characters n and t stand for?

Answer:

\n is a newline and \t is a tab.

Question3.

What is the way to include backslash characters in a string?

Answer:

We need to include backslash characters in a string by using the \ escape character.

Question4.

The string "Howl's Moving Castle" is a correct value. Why isn't the single quote character in the word Howl's not escaped a problem?

Answer:

The string "Howl's Moving Castle" is a correct value. The single quote character in the word Howl's not escaped is not a problem because double quotes have been used to mark the beginning and end of the given string.

Question5.

How do you write a string of newlines if you don't want to use the n character?

Answer:

Multiline strings allow us to write a string of newlines if we don't want to use the \n escape character.

Question6.

What are the values of the given expressions?
a.) 'Hello, world!'[1]
b.) 'Hello, world!'[0:5]
c.) 'Hello, world!'[:5]
d.) 'Hello, world!'[3:]

Answer:

The values of the given expressions are as follows:
a.) 'e'
b.) 'Hello'
c.) 'Hello'
d.) 'lo, world!'

Crosscheck by running the program:

```
In [1]:
'Hello, world!'[1]
Out[1]:
'e'

In [2]:
'Hello, world!'[0:5]
Out[2]:
'Hello'

In [3]:
'Hello, world!'[:5]
Out[3]:
'Hello'

In [4]:
'Hello, world!'[3:]
Out[4]:
'lo, world!'
```

Question7.

What are the values of the following expressions?

a.) 'Hello'.upper()
b.) 'Hello'.upper().isupper()
c.) 'Hello'.upper().lower()

Answer:

The values of the given expressions are as follows:
a.) 'HELLO' because .upper() function changes
b.) True
c.) 'hello'

Crosscheck by running the program:

In [5]:

```
'Hello'.upper()
```

Out[5]:

```
'HELLO'
```

In [6]:

```
'Hello'.upper().isupper()
```

Out[6]:

```
True
```

In [7]:

```
'Hello'.upper().lower()
```

Out[7]:

```
'hello'
```

In [ ]:

```
### Question8.
What are the values of the following expressions?
a.) 'Remember, remember, the fifth of July.'.split()
b.) '-'.join('There can only one.'.split())

Answer:

a.) ['Remember,', 'remember,', 'the', 'fifth', 'of', 'July.']
b.) 'There-can-only-one.'

Crosscheck by running the program:
```

In [8]:

```
'Remember, remember, the fifth of July.'.split()
```

Out[8]:

```
['Remember,', 'remember,', 'the', 'fifth', 'of', 'July.']
```

In [9]:

```
'-'.join('There can only one.'.split())
```

Out[9]:

```
'There-can-only-one.'
```

Question9.

What are the methods for right-justifying, left-justifying, and centering a string?

Answer:

For right justifying a string, we can use rjust() string method.
For left justifying a string, we can use ljust() string method.
For centering a string, we can use center() string method.

Question10.

What is the best way to remove whitespace characters from the start or end?

Answer:

The lstrip() and rstrip() methods can be used to remove whitespaces from the left and right ends of a string, respectively.

Python Basic Assignment - 7

Question1. What is the name of the feature responsible for generating Regex objects?

Answer:

re.compile() is the name feature responsible for generating of Regex objects.

In [1]:

```
import re
variable = re.compile("I_love_milkybar_chocolates")
print(type(variable))
print(variable)
```

```
<class 're.Pattern'>
re.compile('I_love_milkybar_chocolates')
```

Question2. Why do raw strings often appear in Regex objects?

Answer:

Regular expressions use the backslash character `('\')` to indicate special forms `(Metacharacters)` or to allow special characters `(speical sequences)` to be used without invoking their special meaning. This collides with Python's usage of the same character for the same purpose in string literals. Hence, Raw strings

collides with Python's usage of the same character for the same purpose in string literals. Hence, Raw strings are used (e.g. r"\n") so that backslashes do not have to be escaped.

Question3. What is the return value of the search() method?

Answer:

The return value of re.search(pattern,string) method is a `match` object if the pattern is observed in the string otherwise, it returns a `None`.

In [2]:

```
import re

string = 'Ineuron Full Stack Data Science Program is truely awesome'
pattern_1 = 's'
pattern_2 = 'Z'

match = re.search(pattern_1, string, flags=re.IGNORECASE)
print('Output:',match)

match = re.search(pattern_2, string, flags=re.IGNORECASE)
print('Output:',match)
```

```
Output: <re.Match object; span=(13, 14), match='S'>
Output: None
```

Question4. From a Match item, how do you get the actual strings that match the pattern?

Answer:

From a match item, `group()` method helps to get the actual strings that match the pattern.

In [3]:

```
import re

pattern = "Ineuron is helping"
string = "The full stack data science program of Ineuron is helping me to build self-conf
idence for a successful career transition"
match = re.search(pattern, string)
print('Output:',match.group())
```

```
Output: Ineuron is helping
```

Question5. In the regex which created from the r'(\d\d\d)-(\d\d\d-\d\d\d\d)', what does group zero cover Group 2? Group 1?

Answer:

In the Regex, `r'(\d\d\d)-(\d\d\d-\d\d\d\d)'`, the `zero` group covers the `entire pattern match` where as the `first group` cover `(\d\d\d)` and the `second group` cover `(\d\d\d-\d\d\d\d)`.

In [4]:

```
# Example
import re
phoneNumRegex = re.compile(r'(\d\d\d)-(\d\d\d-\d\d\d\d)')
mo = phoneNumRegex.search('My contact number is 343-678-9674.')
print(mo.groups()) # Prints all groups in a tuple format
print(mo.group()) # Always returns the fully matched string
print(mo.group(1)) # Returns the first group
print(mo.group(2)) # Returns the second group
```

```
('343', '678-9674')
343-678-9674
343
678-9674
```

Question6. In standard expression syntax, parentheses and intervals have distinct meanings. How can you tell a regex that you want it to fit real parentheses and periods?

Answer:

Periods and parentheses can be escaped with a backslash: `\.`, `\(`, and `\)`.

Question7. The findall() method returns a string list or a list of string tuples. What causes it to return one of the two options?

Answer:

If the regex has no groups, a list of strings is returned. If the regex has groups, a list of tuples of strings is returned.

In [5]:

```
# Example Program
import re
phoneNumRegex = re.compile(r'(\(\d\d\d\))-(\d\d\d-\d\d\d\d)')
mo = phoneNumRegex.findall('My phone number is (343)-678-9674.')
print(mo)

# Example Program
import re
phoneNumRegex = re.compile(r'\d{3}-\d{3}-\d{4}')
mo = phoneNumRegex.findall('My number is 343-678-9674.')
print(mo) # Prints all groups in a tuple format
```

```
[('(343)', '678-9674')]
['343-678-9674']
```

Question8. In standard expressions, what does the | character mean?

Answer:

In standard expressions, the character `|` means `OR` operator.

The `|` character is called a pipe. One can use it anywhere you want to match one of many expressions.

For example, the regular expression `r'Banana|Apple Fruit'` will match either `'Banana'` or `'Apple Fruit'`.

When both Banana and Apple Fruit occur in the searched string, the first occurrence of matching text will be returned as the Match object.

The `|` character signifies matching "either, or" between two groups.

In [6]:

```
fruitRegex = re.compile (r'Banana|Apple Fruit')
mo1 = fruitRegex.search('Banana and Apple Fruit')
mo1.group()
```

Out[6]:

```
'Banana'
```

In [7]:

```
mo2 = fruitRegex.search('Apple Fruit and Banana')
mo2.group()
```

Out[7]:

```
'Apple Fruit'
```

Question9. In regular expressions, what does the character ? stand for?

Answer:

In regular expressions, the `?` character can either mean "match zero or one of the preceding group".

Sometimes there is a pattern that you want to match only optionally. That is, the regex should find a match regardless of whether that bit of text is there. The ? character flags the group that precedes it as an optional part of the pattern.

In [8]:

```
batRegex = re.compile(r'Bat(wo)?man')
mo1 = batRegex.search('The Adventures of Batman')
mo1.group()
```

Out[8]:

```
'Batman'
```

In [9]:

```
mo2 = batRegex.search('The Adventures of Batwoman')
mo2.group()
```

Out[9]:

```
'Batwoman'
```

Question10.In regular expressions, what is the difference between the + and * characters?

Answer:

In Regular Expressions, `*` Represents Zero ore more occurences of the preceeding group, whereas `+` represents one or more occurences of the preceeding group.

In [10]:

```
import re
match_1 = re.search("Bat(wo)*man","Batman returns")
print(match_1)
match_2 = re.search("Bat(wo)+man","Batman returns")
print(match_2)
```

```
<re.Match object; span=(0, 6), match='Batman'>
None
```

Question11. What is the difference between {4} and {4,5} in regular expression?

Answer:

The {4 matches exactly three instances of the preceding group.

The {4,5} matches between three and five instances.

{4} means that its preceeding group should repeat 4 times. where as {4,5} means that its preceeding group should repeat mininum 4 times and maximum 5 times inclusively

In [11]:

```
import re

haRegex1 = re.compile(r'(Ha){4}')
haRegex2 = re.compile(r'(Ha){4,5}')

mo1 = haRegex1.search('HaHaHaHa')
mo2 = haRegex1.search('Ha')
mo3 = haRegex1.search('HaHaHaHaHaHaHaHa')

print(mo1.group())
print(mo2)
print(mo3.group())

mo4 = haRegex2.search('HaHaHaHa')
mo5 = haRegex2.search('Ha')
mo6 = haRegex2.search('HaHaHaHaHaHaHaHa')

print(mo4.group())
print(mo5)
print(mo6.group())
```

```
HaHaHaHa
None
HaHaHaHa
HaHaHaHa
None
HaHaHaHaHa
```

Question12. What do you mean by the \d, \w, and \s shorthand character classes signify in regular expressions?

Answer:

`\d`, `\w` and `\s` are special sequences in regular expresssions in python:

1. \w – Matches a word character equivalent to [a-zA-Z0-9_]
2. \d – Matches digit character equivalent to [0-9]
3. \s – Matches whitespace character (space, tab, newline, etc.)

Question13. What do means by \D, \W, and \S shorthand character classes signify in regular expressions?

Answer:

`\D`, `\W` and `\S` are special sequences in regular expresssions in python:

1. \W – Matches any non-alphanumeric character equivalent to [^a-zA-Z0-9_]
2. \D – Matches any non-digit character, this is equivalent to the set class [^0-9]
3. \S – Matches any non-whitespace character

Question14. What is the difference between `.*?` and `.*`?

Answer:

`.*` is a Greedy mode, which returns the longest string that meets the condition. Whereas `.*?` is a non greedy mode which returns the shortest string that meets the condition.

`.*` - > The dot-star uses greedy mode: It will always try to match as much text as possible.

`.*?` - > To match any and all text in a non-greedy fashion, use the dot, star, and question mark (`.*?`). Like with braces, the question mark tells Python to match in a non-greedy way.

In [12]:

```
greedyRegex = re.compile(r'<.*>')
mo = greedyRegex.search('<To serve man> for dinner.>')
mo.group()
```

Out[12]:

```
'<To serve man> for dinner.>'
```

In [13]:

```
nongreedyRegex = re.compile(r'<.*?>')
mo = nongreedyRegex.search('<To serve man> for dinner.>')
mo.group()
```

Out[13]:

```
'<To serve man>'
```

Question15. What is the syntax for matching both numbers and lowercase letters with a character class?

Answer:

The syntax is either [a-z0-9] or [0-9a-z]

In [14]:

```
reg1 = re.compile(r'[0-9a-z]')
reg2 = re.compile(r'[a-z0-9]')

mo1 = reg1.search('100 times I am Reading  this for 100 th time')
mo1.group()
```

Out[14]:

```
'1'
```

In [15]:

```
reg2 = re.compile(r'[a-z0-9]')

mo1 = reg2.search('times I am Reading  this for 100 th time')
mo1.group()
```

Out[15]:

```
't'
```

Question16. What is the procedure for making a normal expression in regax case insensitive?

Answer:

The procedure for making a normal expression in regax case insensitive is by passing `re.IGNORECASE` as a flag.

Passing `re.I` the argument to re.compile() can also make the matching case insensitive.

Question17. What does the . character normally match? What does it match if re.DOTALL is passed as 2nd argument in re.compile()?

Answer:

Dot `.` character matches everything in input except newline character `.`. But, by passing `re.DOTALL` as a flag to `re.compile()`, one can make the dot character match all characters, including the newline character.

In [16]:

```
casesen = re.compile(r'machine', re.I)
casesen.search('Machine learning is part of data science').group()
```

Out[16]:

```
'Machine'
```

In [17]:

```
casesen.search('MACHINE is learning.').group()
```

Out[17]:

```
'MACHINE'
```

Question18. If numReg = re.compile(r'\d+'), what will numRegex.sub('X', '11 drummers, 10 pipers, five rings, 4 hen') return?

Answer:

The Ouput will be `'X drummers, X pipers, five rings, X hen'`

In [18]:

```
import re
numReg = re.compile(r'\d+')
numReg.sub('X', '11 drummers, 10 pipers, five rings, 4 hen')
```

Out[18]:

```
'X drummers, X pipers, five rings, X hen'
```

Question19. What does passing re.VERBOSE as the 2nd argument to re.compile() allow to do?

Answer:

`re.VERBOSE` **will allow to add whitespace and comments to string passed to** `re.compile()` **.**

In [19]:

```
# Without Using VERBOSE
regex_email = re.compile(r'^([a-z0-9_\.-]+)@([0-9a-z\.-]+)\.([a-z\.]{2, 6})$', re.IGNOREC
ASE)

# Using VERBOSE
regex_email = re.compile(r"""
                            ^([a-z0-9_\.-]+)              # local Part like username
                            @                             # single @ sign
                            ([0-9a-z\.-]+)                # Domain name like google
                            \.                            # single Dot .
                            ([a-z]{2,6})$                 # Top level Domain  like com/
in/org
                        """,re.VERBOSE | re.IGNORECASE)
```

Question20. How would you write a regex that match a number with comma for every three digits? It must match the given following:

`'42'`
`'1,234'`
`'6,368,745'`
`but not the following:`
`'12,34,567' (which has only two digits between the commas)`
`'1234' (which lacks commas)`

Answer:

In [20]:

```
import re
pattern = r'^\d{1,3}(,\d{3})*$'
pagex = re.compile(pattern)
for ele in ['42','1,234', '6,368,745','12,34,567','1234']:
    print('Output:',ele, '->', pagex.search(ele))
```

```
Output: 42 -> <re.Match object; span=(0, 2), match='42'>
Output: 1,234 -> <re.Match object; span=(0, 5), match='1,234'>
Output: 6,368,745 -> <re.Match object; span=(0, 9), match='6,368,745'>
Output: 12,34,567 -> None
Output: 1234 -> None
```

Question21. How would you write a regex that matches the full name of someone whose last name is Watanabe? You can assume that the first name that comes before it will always be one word that begins with a capital letter. The regex must match the following:

`'Haruto Watanabe'`

`'Alice Watanabe'`

`'RoboCop Watanabe'`

`but not the following:`

`'haruto Watanabe' (where the first name is not capitalized)`

`'Mr. Watanabe' (where the preceding word has a nonletter character)`

`'Watanabe' (which has no first name)`

`'Haruto watanabe' (where Watanabe is not capitalized)`

Answer:

In [21]:

```
import re
pattern = r'[A-Z]{1}[a-z]*\sWatanabe'
namex = re.compile(pattern)
for name in ['Haruto Watanabe','Alice Watanabe','RoboCop Watanabe','haruto Watanabe','Mr.
Watanabe','Watanabe','Haruto watanabe']:
    print('Output: ',name,'->',namex.search(name))
```

```
Output:  Haruto Watanabe -> <re.Match object; span=(0, 15), match='Haruto Watanabe'>
Output:  Alice Watanabe -> <re.Match object; span=(0, 14), match='Alice Watanabe'>
Output:  RoboCop Watanabe -> <re.Match object; span=(4, 16), match='Cop Watanabe'>
Output:  haruto Watanabe -> None
Output:  Mr. Watanabe -> None
Output:  Watanabe -> None
Output:  Haruto watanabe -> None
```

Question22. How would you write a regex that matches a sentence where the first word is either Alice, Bob, or Carol; the second word is either eats, pets, or throws; the third word is apples, cats, or baseballs; and the sentence ends with a period? This regex should be case-insensitive. It must match the following:

`'Alice eats apples.'`

`'Bob pets cats.'`

`'Carol throws baseballs.'`

`'Alice throws Apples.'`

`'BOB EATS CATS.'`

but not the following:
'RoboCop eats apples.'
'ALICE THROWS FOOTBALLS.'
'Carol eats 7 cats.'

Answer:

In [22]:

```
import re
pattern = r'(Alice|Bob|Carol)\s(eats|pets|throws)\s(apples|cats|baseballs)\.'
casex = re.compile(pattern,re.IGNORECASE)
for ele in ['Alice eats apples.','Bob pets cats.','Carol throws baseballs.','Alice throw
s Apples.','BOB EATS CATS.','RoboCop eats apples.'
,'ALICE THROWS FOOTBALLS.','Carol eats 7 cats.']:
    print('Output: ',ele,'->',casex.search(ele))
```

```
Output:  Alice eats apples. -> <re.Match object; span=(0, 18), match='Alice eats apples.'
>
Output:  Bob pets cats. -> <re.Match object; span=(0, 14), match='Bob pets cats.'>
Output:  Carol throws baseballs. -> <re.Match object; span=(0, 23), match='Carol throws b
aseballs.'>
Output:  Alice throws Apples. -> <re.Match object; span=(0, 20), match='Alice throws Appl
es.'>
Output:  BOB EATS CATS. -> <re.Match object; span=(0, 14), match='BOB EATS CATS.'>
Output:  RoboCop eats apples. -> None
Output:  ALICE THROWS FOOTBALLS. -> None
Output:  Carol eats 7 cats. -> None
```

Python Basic Assignment - 8

Question 1. Is the Python Standard Library included with PyInputPlus?

Answer:

No, `PyInputPlus` is not a part of Python Standard Library. It needs to be installed explicitly using the command `!pip install PyInputPlus`

Question 2. Why is PyInputPlus commonly imported with import pyinputplus as pyip?

Answer:

`pyip` is an alias of PyInputPlus.

The `as pyip` code in the import statement saves us from typing yInputPlus each time we want to call a PyInputPlus function. Instead we can use the shorter `pyip` name.

Question 3. How do you distinguish between inputInt() and inputFloat()?

Answer:

`inputInt()` : Accepts an integer value, and returns int value `inputFloat()` : Accepts integer/floating point value and returns float value

Both takes additional parameters ' `min` ', ' `max` ', ' `greaterThan` ' and ' `lessThan` ' for bounds.

In [1]:

```
import pyinputplus as pyip

pyip.inputInt()
```

```
1.2345
'1.2345' is not an integer.
2
```

Out[1]:

```
2
```

In [2]:

```
pyip.inputFloat()
```

```
3
```

Out[2]:

```
3.0
```

Question 4. Using PyInputPlus, how do you ensure that the user enters a whole number between 0 and 99?

Answer:

`PyInputPlus` module provides a function called as `inputInt()` which only returns only integer values. In order to restrict the input between 0 and 99, we can use parameters like `min` and `max` to ensure that user enters the values between the defined range only.

In [3]:

```
import pyinputplus as pyip

wholenumber = pyip.inputInt(prompt='Enter a number: ', min=0, max=99)
print(wholenumber)
```

```
Enter a number: 234
Number must be at maximum 99.
Enter a number: -1
Number must be at minimum 0.
Enter a number: 55
55
```

Question 5. What is transferred to the keyword arguments allowRegexes and blockRegexes?

Answer:

We can use regular expressions to specify whether an input is allowed or not. The `allowRegexes` and `blockRegexes` keyword arguments take a list of regular expression strings to determine what the

`PyInputPlus` **function will accept or reject as valid input.**

In [4]:

```
response = pyip.inputNum(allowRegexes=[r'(I|V|X|L|C|D|M)+', r'zero']) # It allows roman letters too as numbers
```

```
V
```

In [5]:

```
response = pyip.inputNum(blockRegexes=[r'[02468]$'])# It blocks the even numbers
```

```
3456
This response is invalid.
3245
```

Question 6. If a blank input is entered three times, what does inputStr(limit=3) do?

Answer:

The statement `inputStr(limit=3)` **will throw two exceptions** `ValidationException` **and** `RetryLimitException`**. The first exception is thrown because blank values are not allowed by** `inputStr()` **function by default. If we want to consider blank values as valid input, we must set** `blank=True`**. The second exception is thrown because we are provided only 3 chances to provide valid input.**

In [6]:

```
response = pyip.inputStr(limit=3)
```

```
Blank values are not allowed.

Blank values are not allowed.

Blank values are not allowed.

---------------------------------------------------------------------------
ValidationException                       Traceback (most recent call last)
~\anaconda3\lib\site-packages\pyinputplus\__init__.py in _genericInput(prompt, default, t
imeout, limit, applyFunc, validationFunc, postValidateApplyFunc, passwordMask)
    166         try:
--> 167             possibleNewUserInput = validationFunc(
    168                 userInput

~\anaconda3\lib\site-packages\pyinputplus\__init__.py in <lambda>(value)
    242
--> 243     validationFunc = lambda value: pysv._prevalidationCheck(
    244         value, blank=blank, strip=strip, allowRegexes=allowRegexes, blockRegexes=
blockRegexes, excMsg=None,

~\anaconda3\lib\site-packages\pysimplevalidate\__init__.py in _prevalidationCheck(value,
blank, strip, allowRegexes, blockRegexes, excMsg)
    249         # value is blank but blanks aren't allowed.
--> 250         _raiseValidationException(_("Blank values are not allowed."), excMsg)
    251     elif blank and value == "":

~\anaconda3\lib\site-packages\pysimplevalidate\__init__.py in _raiseValidationException(s
tandardExcMsg, customExcMsg)
    221     if customExcMsg is None:
--> 222         raise ValidationException(str(standardExcMsg))
    223     else:

ValidationException: Blank values are not allowed.

During handling of the above exception, another exception occurred:
```

```
RetryLimitException                       Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_9964/1250714940.py in <module>
----> 1 response = pyip.inputStr(limit=3)

~\anaconda3\lib\site-packages\pyinputplus\__init__.py in inputStr(prompt, default, blank,
timeout, limit, strip, allowRegexes, blockRegexes, applyFunc, postValidateApplyFunc)
    245     )[1]
    246
--> 247     return _genericInput(
    248         prompt=prompt,
    249         default=default,

~\anaconda3\lib\site-packages\pyinputplus\__init__.py in _genericInput(prompt, default, t
imeout, limit, applyFunc, validationFunc, postValidateApplyFunc, passwordMask)
    186                 else:
    187                     # If there is no default, then raise the timeout/limit except
ion.
--> 188                     raise limitOrTimeoutException
    189             else:
    190                 # If there was no timeout/limit exceeded, let the user enter inpu
t again.

RetryLimitException:
```

Question 7. If blank input is entered three times, what does inputStr(limit=3, default='hello') do?

Answer:

Since the default parameter is set to `hello`, if blank input is entered three times instead of raising `RetryLimitException exception`, the function will return `hello` as response to the calling function.

In [7]:

```
response = pyip.inputStr(limit=3,default='hello')
response
```

```
Blank values are not allowed.

Blank values are not allowed.

Blank values are not allowed.
```

Out[7]:

```
'hello'
```

Python Basic Assignment - 11

Question1.

Create an assert statement that throws an AssertionError if the variable spam is a negative integer.

Answer:

In [1]:

```
spam = int(input("Enter a positive integer in order to not to get any AssertionError:  "))
```

```
)
assert spam >= 0, "The spam variable is a negative number, which is unacceptable."
```

```
Enter a positive integer in order to not to get any AssertionError:  -876996

---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_8868/1176719640.py in <module>
      1 spam = int(input("Enter a positive integer in order to not to get any AssertionEr
ror:  "))
----> 2 assert spam >= 0, "The spam variable is a negative number, which is unacceptable.
"

AssertionError: The spam variable is a negative number, which is unacceptable.
```

Question2.

Write an assert statement that triggers an AssertionError if the variables eggs and bacon contain strings that are the same as each other, even if their cases are different (that is, 'hello' and 'hello' are considered the same, and 'goodbye' and 'GOODbye' are also considered the same).

Answer:

In [2]:

```
eggs = input("Enter any string:     ")
bacon = input("Enter any string:    ")

assert eggs.lower() != bacon.lower(), "The eggs and bacon variables are the same!"
assert eggs.upper() != bacon.upper(), "The eggs and bacon variables are the same!"
```

```
Enter any string:     qwERTY
Enter any string:    qwerty

---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_8868/2971924175.py in <module>
      2 bacon = input("Enter any string:    ")
      3
----> 4 assert eggs.lower() != bacon.lower(), "The eggs and bacon variables are the same!
"
      5 assert eggs.upper() != bacon.upper(), "The eggs and bacon variables are the same!
"

AssertionError: The eggs and bacon variables are the same!
```

Question3.

Create an assert statement that throws an AssertionError every time.

Answer:

In [3]:

```
assert False, 'This assertion statement always throws an AssertionError.'
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_8868/2930411765.py in <module>
----> 1 assert False, 'This assertion statement always throws an AssertionError.'

AssertionError: This assertion statement always throws an AssertionError.
```

Question4.

What are the two lines that must be present in your software in order to call logging.debug()?

Answer:

In order to call logging.debug(), we must have following two lines at the start of our program:

```
import logging
logging.basicConfig(level = logging.DEBUG, format = ' %(asctime)s %(levelname)s %(message)s')
```

Question5.

What are the two lines that your program must have in order to have logging.debug() send a logging message to a file named programLog.txt?

Answer:

To be able to send logging messages to a file named programLog.txt with logging.debug(), we must have following two lines at the start of our program:

```
import logging
logging.basicConfig(filename = 'programLog.txt', level = logging.DEBUG, format = ' %(asctime)s %(levelname)s %(message)s')
```

Question6.

What are the five levels of logging?

Answer:

DEBUG, INFO, WARNING, ERROR, and CRITICAL are the five levels of logging in ascending order.

Question7.

What line of code would you add to your software to disable all logging messages?

Answer:

logging.disable(logging.CRITICAL) can be added to our program in order to disable all logging messages.

Question8.

Why is using logging messages better than using print() to display the same message?

Answer:

Using logging messages better than using print() to display the same message because:

1.) We can disable logging messages without removing the logging function calls.

2.) We can selectively disable lower-level logging messages.
3.) We can create logging messages.
4.) Logging messages provides a timestamp

Question9.

What are the differences between the Step Over, Step In, and Step Out buttons in the debugger?

Answer:

The Step In button moves the debugger into a function call. The Step Over button quickly executes the function call without stepping into it. The Step Out button quickly executes the rest of the code until it steps out of the function it currently is in.

Question10.

After you click Continue, when will the debugger stop ?

Answer:

After we click Continue, the debugger will stop when it has reached the end of the program or a line with a breakpoint.

Question11.

What is the concept of a breakpoint?

Answer:

A breakpoint is a setting on a line of code that causes the debugger to pause when the program execution reaches the line.

Python Basic Assignment - 18

Question1.

Create a zoo.py file first. Define the hours() function, which prints the string 'Open 9-5 daily'. Then, use the interactive interpreter to import the zoo module and call its hours() function.

Answer:

In [1]:

```
import zoo
zoo.hours()
```

```
Open 9-5 daily
```

Question2.

In the interactive interpreter, import the zoo module as menagerie and call its hours() function.

Answer:

In [2]:

```
import zoo as menagerie
menagerie.hours()
```

```
Open 9-5 daily
```

Question3.

Using the interpreter, explicitly import and call the hours() function from zoo.

Answer:

In [3]:

```
from zoo import hours
hours()
```

```
Open 9-5 daily
```

Question4.

Import the hours() function as info and call it.

Answer:

In [4]:

```
from zoo import hours as  info
info()
```

```
Open 9-5 daily
```

Question5.

Create a plain dictionary with the key-value pairs 'a': 1, 'b': 2, and 'c': 3, and print it out.

Answer:

In [5]:

```
plain_dictionary = {'a':1,'b':2,'c':3}
print(plain_dictionary)
```

```
{'a': 1, 'b': 2, 'c': 3}
```

Question6.

Make an OrderedDict called fancy from the same pairs listed in 5 and print it. Did it print in the same order as

plain?

Answer:

In [6]:

```
from collections import OrderedDict
fancy = OrderedDict(plain_dict)

print(fancy)
```

```
OrderedDict([('a', 1), ('b', 2), ('c', 3)])
```

Yes, the fancy dictionary got printed in the same order as the plain_dictionary.

Question7.

Make a default dictionary called dict_of_lists and pass it the argument list. Make the list dict_of_lists['a'] and append the value 'something for a' to it in one assignment. Print dict_of_lists['a'].

Answer:

In [7]:

```
from collections import defaultdict
dict_of_lists = defaultdict(list)
dict_of_lists['a'].append('something for a')
print(dict_of_lists['a'])
```

```
['something for a']
```

Python Basic Assignment - 12

Question 1. In what modes should the PdfFileReader() and PdfFileWriter() File objects will be opened?

Answer:

For `PdfFileReader()`, file objects should be opened in `rb` i.e. read binary mode; whereas for `PdfFileWriter()`, file objects should be opened in `wb` i.e. write binary mode.

Question 2. From a PdfFileReader object, how do you get a Page object for page 5?

Answer:

`PdfFileReader` class provides a method called `getPage(page_no)` to get a page object.

In [ ]:

```
# Example Code:
from PyPDF2 import PdfFileReader
pdf_reader = PdfFileReader(file_path)
```

```
for page in pdf_reader.getNumPages():
    pdf_reader.getPage(page_no) # put page_no = 5 to get a Page object for page 5
```

Question 3. What PdfFileReader variable stores the number of pages in the PDF document?

Answer:

`getNumPages()` **method of** `PdfFileReader` **class stores the number of pages in a PDF document.**

In [ ]:

```
#Example Code:
from PyPDF2 import PdfFileReader
pdf_reader = PdfFileReader(file_path)
print(pdf_reader.getNumPages()) # Prints the number of pages in an input PDF document
```

Question 4. If a PdfFileReader object's PDF is encrypted with the password swordfish, what must you do before you can obtain Page objects from it?

Answer:

If a `PdfFileReader` **object's PDF is encrypted with the password** `swordfish` **and we're not aware of it, then at first we should read the Pdf using the** `PdfFileReader` **Class.** `PdfFileReader` **class provides a attribute called** `isEncrypted` **to check whether a pdf is encrypted or not. The method returns** `True` **if a pdf is encrypted and** `False` **if the PDF file is not encrypted.**

If pdf is encrypted, we use the `decrypt()` **method provided by** `PdfFileReader` **class first and then try to read the contents/pages of the pdf, otherwise** `PyPDF2` **will raise the following error:**

`PyPDF2.utils.PdfReadError: file has not been decrypted`

In [ ]:

```
#Example Code:
from PyPDF2 import PdfFileReader
pdf_reader = PdfFileReader(file_path)
if pdf_reader.isEncrypted: # to check whether the pdf is encrypted or not
    pdf_reader.decrypt("swordfish")
for page in pdf_reader.pages:
    print(page.extractText()) # to print the text data of a page from pdf
```

Question 5. What methods do you use to rotate a page?

Answer:

`PyPDF2` **Package provides 2 methods to rotate a page:**

1. `rotateClockWise()` -> ***For Clockwise rotation***
2. `rotateCounterClockWise()` -> ***For Counter Clockwise rotation***

The `PyPDF2` **package only allows to rotate a page in increments of 90 degrees, otherwise we receive an**

AssertionError.

Question . What is the difference between a Run object and a Paragraph object?

Answer:

The structure of a document is represented by three different data types in `python-Docx`. At the highest level, a `Document` object represents the entire document. The `Document` object contains a list of `Paragraph` objects for the paragraphs in the document. (A new paragraph begins whenever the user presses `ENTER` or `RETURN` while typing in a Word document.) Each of these Paragraph objects contain a list of one or more `Run` objects.

The text in a Word document is more than just a string. It has font, size, color, and other styling information associated with it. A `style` in Word is a collection of these attributes. A `Run` object is a contiguous groups of characters within a paragraph with the same style. A new Run object is needed whenever the text style changes.

Question 7. How do you obtain a list of Paragraph objects for a Document object that's stored in a variable named doc?

Answer:

We obtain a list of Paragraph objects for a Document object that's stored in a variable named doc by using `doc.paragraphs`.

In [ ]:

```
# Example Program
#!pip install python-docx (if docx module is not installed)
from docx import Document
doc = Document("sample_file.docx") # Path of the Docx file (NOTE: the extension may be .d
oc in case of a document file)
print(doc.paragraphs) # Prints the list of Paragraph objects for a Document
for paragraph in doc.paragraphs:
    print(paragraph.text) # Prints the text in the paragraph
```

Question 8. What type of object has bold, underline, italic, strike, and outline variables?

Answer:

A `Run` object has bold, underline, italic, strike, and outline variables.

Question 9. What is the difference between False, True, and None for the bold variable?

Answer:

Runs can be further styled using text attributes. Each attribute can be set to one of three values:

1. **True (the attribute is always enabled, no matter what other styles are applied to the run, i.e. style Set to Bold)**
2. **False (the attribute is always disabled, i.e. Style Not Set to Bold)**
3. **None (defaults to whatever the run's style is set to, i.e. Style is Not Applicable)**

`True` always makes the `Run` object bolded and `False` makes it always not bolded, no matter what the style's bold setting is. `None` will make the `Run` object just use the style's bold setting.

Question 10. How do you create a Document object for a new Word document?

Answer:

We can create a Document object for a new Word document by calling the `docx.Document()` function.

In [ ]:

```
# Example Program
from docx import Document
document = Document()
document.add_paragraph("iNeuron Full Stack DataScience Course")
document.save('myDocument.docx')
```

Question 11. How do you add a paragraph with the text 'Hello, there!' to a Document object stored in a variable named doc?

Answer:

In [ ]:

```
# Example Program
from docx import Document
doc = Document()
doc.add_paragraph('Hello, there!')
doc.save('hello.docx')
```

Question 12. What integers represent the levels of headings available in Word documents?

Answer:

The levels for a heading in a word document can be specified by using the `level` attribute inside the `add_heading` method. There are a total of `5` levels statring for `0 t0 4`, where level 0 makes a headline with the horizontal line below the text, whereas the heading level 1 is the main heading. Similarly, the other headings are sub-heading with their's font-sizes in decreasing order.

Python Basic Assignment - 13

Question 1. What advantages do Excel spreadsheets have over CSV spreadsheets?

Answer:

Following advantages do Excel spreadsheets have over CSV spreadsheets:

1. **It is a binary file that holds information about all the worksheets in a workbook.**
2. **An Excel not only stores data but can also do operations on the data.**
3. **Files saved in excel cannot be opened or edited by text editors.**
4. **Large files user is much easier in Excel for the end user. Also, one can have additional functions like selecting individual cells for import, converting dates and time automatically, reading formulas and their results, filtering, sorting, etc.**
5. **Apart from text, data can also be stored in form of charts and graphs**
6. **Excel can connect to external data sources to fetch data. You can use custom add-in in Excel to increase its functionality.**
7. **Excel allows for Review of Data with detailed tracking and commenting feature.**
8. **In Excel, spreadsheets can have values of data types other than strings; cells can have different fonts, sizes, or color settings; cells can have varying widths and heights; adjacent cells can be merged.**
9. **Excel (XLS and XLSX) file formats are better for storing and analysing complex data.**
10. **An Excel not only stores data but can also do operations on the data using macros, formulas etc.**
11. **CSV files are plain-text files, Does not contain formatting, formulas, macros, etc. It is also known as flat files.**

Question2.What do you pass to csv.reader() and csv.writer() to create reader and writer objects?

Answer:

We pass a File object and a delimeter object if necessary, to csv.reader() and csv.writer() to create reader and writer objects.

In [ ]:

```
#code example:

import csv
with open('text.csv','r') as file:
    csv_file = csv.reader(file,delimiter=',')
    for ele in csv_file:
        print(ele)
```

Question3. What modes do File objects for reader and writer objects need to be opened in?

Answer:

For `csv.reader(iterable_file_object)` , the file objects need to be opened in read mode i.e. `mode='r'` , whereas for `csv.writer(iterable_file_object)` , the file objects need to be opened in write mode i.e. `mode='w'` .

Question4. What method takes a list argument and writes it to a CSV file?

Answer:

`csv.writer` class provides two methods for writing to CSV. They are `writerow()` and `writerows()`. `writerow()` method writes a single row at a time, whereas `writerows()` method is used to write multiple rows at a time.

In [ ]:

```
# Example Program:

import csv
fields = ['Name', 'Branch', 'Year_Of_Passing', 'Marks(CGPA)'] #column names
rows = [
            ['Utkarsh', 'CE', '2019', '9.0'],  # data rows of csv file
            ['Sanjeev', 'CE', '2019', '9.1'],
            ['Anshuman', 'CSE', '2019', '9.3']
       ]
with open("academic_records.csv", 'w') as csvfile:
    csvwriter = csv.writer(csvfile) # creating a csv writer object
    csvwriter.writerow(fields) # writing the fields
    csvwriter.writerows(rows) # writing the data rows
```

Question5. What do the keyword arguments delimiter and line terminator do?

Answer:

The delimiter argument changes the string used to separate cells in a row.

The lineterminator argument changes the string used to separate rows.

Question6. What function takes a string of JSON data and returns a Python data structure?

Answer:

`loads()` **method takes a string of JSON data and returns a Python data structure**

In [1]:

```
# Example of json.loads() method

import json
my_details_json ='''{
    "Name": "Utkarsh",
    "Qualification": "Bachelor of Engineering",
    "Stream": "Civil Engineering"
}'''

print(my_details_json)
print()
print(f'Type of my_details_json is {type(my_details_json)}')
print()
```

```
my_details = json.loads(my_details_json)
print(my_details)
print()
print(f'Type of my_details is {type(my_details)}')
```

```
{
    "Name": "Utkarsh",
    "Qualification": "Bachelor of Engineering",
    "Stream": "Civil Engineering"
}

Type of my_details_json is <class 'str'>

{'Name': 'Utkarsh', 'Qualification': 'Bachelor of Engineering', 'Stream': 'Civil Engineer
ing'}

Type of my_details is <class 'dict'>
```

Question7. What function takes a Python data structure and returns a string of JSON data?

Answer:

`dumps()` **method takes a python data structure and returns a string of JSON data**

In [2]:

```
# Example of json.dumps() method

import json
my_details = {
    "Name": "Utkarsh",
    "Qualification": "Bachelor of Engineering",
    "Stream": "Civil Engineering"
}

print(my_details)
print()
print(f'Type of my_details is {type(my_details)}')
print()

my_details_json = json.dumps(my_details, indent=4, sort_keys=True)
print(my_details_json)
print()
print(f'Type of my_details_json is {type(my_details_json)}')
```

```
{'Name': 'Utkarsh', 'Qualification': 'Bachelor of Engineering', 'Stream': 'Civil Engineer
ing'}

Type of my_details is <class 'dict'>

{
    "Name": "Utkarsh",
    "Qualification": "Bachelor of Engineering",
    "Stream": "Civil Engineering"
}

Type of my_details_json is <class 'str'>
```

Python Basic Assignment - 14

Question 1. What does RGBA stand for?

Answer:

RGBA stands for red, green, blue, and alpha.

An RGBA value is a tuple of 4 integers, each ranging from 0 to 255. The four integers correspond to the amount of red, green, blue, and alpha (transparency) in the color .

Question 2. From the Pillow module, how do you get the RGBA value of any images?

Answer:

`ImageColor.getcolor()` **gives rgba value of any image.**

In [1]:

```
#code example:

from PIL import ImageColor
ImageColor.getcolor('red', 'RGBA')
ImageColor.getcolor('green', 'RGBA')
```

Out[1]:

```
(0, 128, 0, 255)
```

Question 3. What is a box tuple, and how does it work?

Answer:

A box tuple is a tuple value of four integers: the left-edge x-coordinate, the top-edge y-coordinate,the width, and the height, respectively.

Question 4. Use your image and load in notebook then, How can you find out the width and height of an Image object?

Answer:

In [2]:

```
#Example Program:

from PIL import Image
pic = Image.open('ukcimage.jpg')
print(f'Width, Height -> {pic.size}') # Approach 1
print(f'Width, Height -> {pic.width},{pic.height}') # Approach 2
width,height = pic.size
print(f'Width, Height -> {width},{height}') # Approach 3
```

```
Width, Height -> (1000, 1000)
Width, Height -> 1000,1000
Width, Height -> 1000,1000
```

Question 5. What method would you call to get Image object for a 100×100 image, excluding the lower-left quarter of it?

Answer:

`ImageObject.crop((0, 50, 50, 50))` **method can be called to get Image object for a 100×100 image, excluding the lower-left quarter of it.**

In [3]:

```
#example program:

from PIL import Image
img = Image.open('ukcimage.jpg')
new_img = img.crop((0,50,50,50))
```

Question 6. After making changes to an Image object, how could you save it as an image file?

Answer:

By Calling the `imageObj.save('new_filename.png')` **method of the Image object, one can save it as an image file.**

In [4]:

```
#Example Program:

from PIL import Image
pic = Image.open('ukcimage.jpg')
pic.save('new_pic.jpg')
```

Question 7. What module contains Pillow's shape-drawing code?

Answer:

The `ImageDraw` **module contains code to draw on images and Pillow's shape-drawing code too.**

Question 8. Image objects do not have drawing methods. What kind of object does? How do you get this kind of object?

Answer:

`ImageDraw` **objects have shape-drawing methods such as** `point()`, `line()`, **or** `rectangle()`**.They are returned by passing the Image object to the** `ImageDraw.Draw()` **function.**

Python Basic Assignment - 15

Question1.

How many seconds are in an hour? Use the interactive interpreter as a calculator and multiply the number of seconds in a minute (60) by the number of minutes in an hour (also 60).

sol. 60

Answer:

In [1]:

```
Minutes_Per_Hour = 60
Seconds_Per_Minute = 60

Minutes_Per_Hour * Seconds_Per_Minute
```

Out[1]:

```
3600
```

Question2.

Assign the result from the previous task (seconds in an hour) to a variable called seconds_per_hour.

Answer:

In [2]:

```
Seconds_Per_Hour = 3600
```

Question3.

How many seconds do you think there are in a day? Make use of the variables seconds per hour and minutes per hour.

Answer:

In [3]:

```
Hours_Per_Day = 24

Seconds_Per_Hour * Hours_Per_Day
```

Out[3]:

```
86400
```

Question4.

Calculate seconds per day again, but this time save the result in a variable called seconds_per_day

Answer:

In [4]:

```
Hours_Per_Day = 24

Seconds_Per_Day = Seconds_Per_Hour * Hours_Per_Day
Seconds_Per_Day
```

Out[4]:

```
86400
```

Question5.

Divide seconds_per_day by seconds_per_hour. Use floating-point (/) division.

Answer:

In [5]:

```
Seconds_Per_Day/Seconds_Per_Hour
```

Out[5]:

```
24.0
```

Question6.

Divide seconds_per_day by seconds_per_hour, using integer (//) division. Did this number agree with the floating-point value from the previous question, aside from the final .0?

Answer:

In [6]:

```
Seconds_Per_Day//Seconds_Per_Hour
```

Out[6]:

```
24
```

Yes, the number in the above result agreed with the floating-point value from the previous question, aside from the final .0

Question7.

Write a generator, genPrimes, that returns the sequence of prime numbers on successive calls to its next() method: 2, 3, 5, 7, 11, ...

Answer:

In [7]:

```
def genPrimes():

    """
    It is a function that returns the sequence of prime numbers on successive calls to it
s next() method.
```

```
    """

    Sequence_Of_Prime_Numbers = []
    n = 2
    Last_Number_Checked = n

    while True:
        for i in Sequence_Of_Prime_Numbers:
            if n % i == 0:
                n = n + 1
                break

        else:
            Sequence_Of_Prime_Numbers.append(n)
            Last_Number_Checked = n
            n = n + 1
            yield Sequence_Of_Prime_Numbers

p = genPrimes()
```

Crosscheck by running the program and calling the function by next() method:

In [8]:

```
next(p)
```

Out[8]:

```
[2]
```

In [9]:

```
next(p)
```

Out[9]:

```
[2, 3]
```

In [10]:

```
next(p)
```

Out[10]:

```
[2, 3, 5]
```

In [11]:

```
next(p)
```

Out[11]:

```
[2, 3, 5, 7]
```

In [12]:

```
next(p)
```

Out[12]:

```
[2, 3, 5, 7, 11]
```

Python Basic Assignment - 16

Question1.

Create a list called years_list, starting with the year of your birth, and each year thereafter until the year of your

Create a list called years_list, starting with the year of your birth, and each year thereafter until the year of your fifth birthday. For example, if you were born in 1980. the list would be years_list = [1980, 1981, 1982, 1983, 1984, 1985].

Answer:

In [1]:

```
years_list = [1998]
i = 1
while i<6:
    years_list.append(1998+i)
    i = i + 1
years_list
```

Out[1]:

```
[1998, 1999, 2000, 2001, 2002, 2003]
```

OR

In [2]:

```
years_list = [i for i in range(1998, 1998 + 6)]
years_list
```

Out[2]:

```
[1998, 1999, 2000, 2001, 2002, 2003]
```

Question2.

In which year in years_list was your third birthday? Remember, you were 0 years of age for your first year.

Answer:

In [3]:

```
years_list[3]
```

Out[3]:

```
2001
```

Question3.

In the years list, which year were you the oldest?

Answer:

In [4]:

```
years_list[-1]
```

Out[4]:

```
2003
```

Question4.

Make a list called things with these three strings as elements: "mozzarella", "cinderella", "salmonella".

Answer:

In [5]:

```
String_elements = ["mozzarella", "cinderella", "salmonella"]
String_elements
```

Out[5]:

```
['mozzarella', 'cinderella', 'salmonella']
```

Question5.

Capitalize the element in things that refers to a person and then print the list. Did it change the element in the list?

Answer:

In [6]:

```
for i in range(len(String_elements)):
    String_elements[i] = String_elements[i].capitalize()

String_elements
```

Out[6]:

```
['Mozzarella', 'Cinderella', 'Salmonella']
```

Question6.

Make a surprise list with the elements "Groucho", "Chico", and "Harpo."

Answer:

In [7]:

```
Surprise_List = [ "Groucho", "Chico", "Harpo"]
Surprise_List
```

Out[7]:

```
['Groucho', 'Chico', 'Harpo']
```

Question7.

Lowercase the last element of the surprise list, reverse it, and then capitalize it.

Answer:

In [8]:

```
#Lowercasing the last element of the surprise list
Surprise_List[-1].lower()
```

Out[8]:

```
'harpo'
```

In [9]:

```
#Reversing the last element of the surprise list after it was lowercased
Surprise_List[-1].lower()[::-1]
```

Out[9]:

```
'oprah'
```

In [10]:

```
#Capitalizing the last element of the surprise list after it was lowercased and reversed
Surprise_List[-1].lower()[::-1].capitalize()
```

Out[10]:

```
'Oprah'
```

Question8.

Make an English-to-French dictionary called e2f and print it. Here are your starter words: dog is chien, cat is chat, and walrus is morse.

Answer:

In [11]:

```
e2f = {"dog" : "chien", "cat" : "chat", "walrus" : "morse"}
print(e2f)
```

```
{'dog': 'chien', 'cat': 'chat', 'walrus': 'morse'}
```

Question9.

Write the French word for walrus in your three-word dictionary e2f.

Answer:

In [12]:

```
e2f["walrus"]
```

Out[12]:

```
'morse'
```

Question10.

Make a French-to-English dictionary called f2e from e2f. Use the items method.

Answer:

In [13]:

```
f2e = dict((key,value) for value,key in e2f.items())
f2e
```

Out[13]:

```
{'chien': 'dog', 'chat': 'cat', 'morse': 'walrus'}
```

Question11.

Print the English version of the French word chien using f2e.

Answer:

In [14]:

```
f2e["chien"]
```

Out[14]:

```
'dog'
```

Question12.

Make and print a set of English words from the keys in e2f.

Answer:

In [15]:

```
print(set(e2f.keys()))
```

```
{'walrus', 'dog', 'cat'}
```

Question13.

Make a multilevel dictionary called life. Use these strings for the topmost keys: 'animals', 'plants', and 'other'. Make the 'animals' key refer to another dictionary with the keys 'cats', 'octopi', and 'emus'. Make the 'cats' key refer to a list of strings with the values 'Henri', 'Grumpy', and 'Lucy'.
Make all the other keys refer to empty dictionaries.

Answer:

In [16]:

```
life ={'animals':{'cats':['Henri', 'Grumpy', 'Lucy'],
                  'octopi':{},
                  'emus':{}
                  },
       'plants' :{},
       'other' :{}
        }
life
```

Out[16]:

```
{'animals': {'cats': ['Henri', 'Grumpy', 'Lucy'], 'octopi': {}, 'emus': {}},
 'plants': {},
 'other': {}}
```

Question14.

Print the top-level keys of life.

Answer:

In [17]:

```
print(life.keys())
```

```
dict_keys(['animals', 'plants', 'other'])
```

Question15.

Print the keys for life['animals'].

Answer:

In [18]:

```
print(life['animals'].keys())
```

```
dict_keys(['cats', 'octopi', 'emus'])
```

Question16.

Print the values for life['animals']['cats']

Answer:

In [19]:

```
life['animals']['cats']
```

Out[19]:

```
['Henri', 'Grumpy', 'Lucy']
```