In [2]:
```
print ('Hello World')
```

```
Hello World
```

In [8]:
```
import sys
print(sys.version)
```

```
3.9.12 (main, Apr  5 2022, 01:53:17)
[Clang 12.0.0 ]
```

In [10]:
```
#This is my introduction
print ('My name is Maryam')
print ('I am a student')
```

```
My name is Maryam
I am a student
```

In [14]:
```
#integer
print (23)
```

```
23
```

In [20]:
```
# Type of 12344.34
type(12344.34)
```

Out[20]:
```
float
```

In [22]:
```
#verify this is an integer
type (456)
```

Out[22]:
```
int
```

In [25]:
```
#convert the following to float

float(34)
```

Out[25]:
```
34.0
```

In [26]:
```
#convert interger to float and check its type
type (float(34567))
```

Out[26]:
```
float
```

In [28]:
```
# Convert a string into an integer with error

int('1 person')
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Input In [28], in <cell line: 3>()
      1 # Convert a string into an integer with error
----> 3 int('1 person')

ValueError: invalid literal for int() with base 10: '1 person'
```

In [29]:
```
# Convert the string "1.2" into a float

float('1.2')
```

Out[29]:
```
1.2
```

In [30]:
```
# Convert an integer to a string

str(2222)
```

Out[30]:
```
'2222'
```

In [31]:
```
# Convert a float to a string

str(1345.2)
```

Out[31]:
```
'1345.2'
```

In [32]:
```
# Type of True

type(True)
```

Out[32]:
```
bool
```

In [33]:
```
# Convert True to int

int(True)
```

Out[33]:
```
1
```

In [34]:
```
# Convert 1 to boolean

bool(1)
```

Out[34]:
```
True
```

In [35]:
```
# Convert 0 to boolean

bool(0)
```

Out[35]:
```
False
```

In [36]:
```
# Convert True to float

float(True)
```

Out[36]:
```
1.0
```

In [37]:
```
type (print (6/2))
```

Out[37]:
```
3.0
NoneType
```

In [38]:
```
type(6/2) # float
```

Out[38]:
```
float
```

In [39]:
```
type(6//2) # int, as the double slashes stand for integer division
```

Out[39]:
```
int
```

In [42]:
```
# As seen in the cell above, we can use the double slash for integer division, where the result is rounded down
# Integer division operation expression

25 // 5
```

Out[42]:
```
5
```

In [41]:
```
# Integer division operation expression

25 // 6
```

Out[41]:
```
4
```

In [43]:
```
#how many hours there are in 160 minutes:
160/60
```

Out[43]:
```
2.6666666666666665
```

In [44]:
```
40 +70 *34
```

Out[44]:
```
2420
```

In [45]:
```
# Mathematical expression
#And just like mathematics, expressions enclosed in parentheses have priority. So the following multiplies 32 b
(30 + 2) * 60
```

Out[45]:
```
1920
```

In [46]:
```
# store value in a variable
x = 34+76 +45 - 50
```

In [47]:
```
# Print out the value in variable

x
```

Out[47]:
```
105
```

In [48]:
```
# perform operation on the value of x

y = x/ 24 #float
```

In [49]:
```
# Print out the value in variable

y
```

Out[49]:
```
4.375
```

In [50]:
```
x = 3 + 2 * 2
```

In [50]:
```
x
```

Out[50]:
```
7
```

In [51]:
```
y = (3 + 2) * 2
y
```

Out[51]:
```
10
```

In [52]:
```
z = x + y
z
```

Out[52]:
```
17
```

In [53]:
```
# Use quotation marks for defining string

"My name is Maryam"
```

Out[53]:
```
'My name is Maryam'
```

In [58]:
```
# Concatenate two strings
name = "Maryam"
statement = name + " is the best"
statement
```

Out[58]:
```
'Maryam is the best'
```

In [59]:
```
# New line escape sequence

print(" Maryam \n is the best" )
```

```
 Maryam
 is the best
```

In [60]:
```
# tab escape sequence

print(" Maryam \t is the best" )
```

```
 Maryam          is the best
```

In [61]:
```
# Include back slash in string
print ("Maryam \\ is the best")
```

```
Maryam \ is the best
```

In [62]:
```
# Convert all the characters in string to upper case

a = "Thriller is the sixth studio album"
print("before upper:", a)
b = a.upper()
print("After upper:", b)
```

```
before upper: Thriller is the sixth studio album
After upper: THRILLER IS THE SIXTH STUDIO ALBUM
```

In [66]:
```
# Convert all the characters in string to upper case

a = "Thriller is the sixth studio album"
print("before upper:", a)
b = a.upper()
print("After upper:", b)
```

```
before upper: Thriller is the sixth studio album
After upper: THRILLER IS THE SIXTH STUDIO ALBUM
```

In [67]:
```
# Convert all the characters in string to upper case

a = ("I want to go to UK")
print ("before cap;",a)
b = a.upper()
print ("after cap:",b)
```

```
before cap; I want to go to UK
after cap: I WANT TO GO TO UK
```

In [72]:
```
# The method <code>replace</code> replaces a segment of the string, i.e. a substring  with a new string. We inp

a = ("I want to go to UK")
b = a.replace("I", "Jenny")
b
```

Out[72]:
```
'Jenny want to go to UK'
```

In [73]:
```
a = "1"
```

In [77]:
```
b = "2"
```

In [78]:
```
c = a + b
c
```

Out[78]:
```
'12'
```

In [80]:
```
#Consider the variable d use slicing to print out the first three elements:
d = "ABCDEFG"
print (d[:3])
```

```
ABC
```

In [81]:
```
#Use a stride value of 2 to print out every second character of the string e:


e = 'clocrkr1e1c1t'
print (e[::2])
```

```
correct
```

In [85]:
```
#Print out a backslash:
print("\\\\\\\")
```

```
\\\
```

In [90]:
```
#Convert the variable f to uppercase:
f = "You are wrong"
f.upper()
```

Out[90]:
```
'YOU ARE WRONG'
```

In [92]:
```
# Consider the variable g, and find the first index of the sub-string snow:
g = "Mary had a little lamb Little lamb, little lamb Mary had a little lamb \
Its fleece was white as snow And everywhere that Mary went Mary went, Mary went \
Everywhere that Mary went The lamb was sure to go"

g.find("snow")
```

Out[92]:
```
95
```

In [95]:
```
#In the variable g, replace the sub-string Mary with Bob:
g = "Mary had a little lamb Little lamb, little lamb Mary had a little lamb \
Its fleece was white as snow And everywhere that Mary went Mary went, Mary went \
Everywhere that Mary went The lamb was sure to go"
g.replace("Mary", "Bob")
```

Out[95]:
```
'Bob had a little lamb Little lamb, little lamb Bob had a little lamb Its fleece was white as snow And everywhe
re that Bob went Bob went, Bob went Everywhere that Bob went The lamb was sure to go'
```

In [97]:
```
#What is the value of x after the following lines of code?

x=2

x=x+2
x
```

Out[97]:
```
4
```

In [98]:
```
 1+3*2
```

Out[98]:
```
7
```

In [99]:
```
type(int(12.3))
```

Out[99]:
```
int
```

In [100…
```
int(True)
```

Out[100]:
```
1
```

In [101…
```
'1'+'2'
```

Out[101]:
```
'12'
```

In [102…
```
myvar = 'hello'
#how would you return myvar as uppercase?
myvar.upper()
```

Out[102]:
```
'HELLO'
```

In [103…
```
str(1)+str(1)
```

Out[103]:
```
'11'
```

In [104…
```
"123".replace("12", "ab")
```

Out[104]:
```
'ab3'
```

In [107…
```
x=1/1
type (x)
```

Out[107]:
```
float
```

In [106…
```
type(x)
```

Out[106]:
```
float
```

In [108…
```
B=[1,2,[3,'a'],[4,'b']]
B[3][1]
```

Out[108]:
```
'b'
```

In [109…
```
[1,2,3]+[1,1,1]
```

Out[109]:
```
[1, 2, 3, 1, 1, 1]
```

In [111…
```
A = [1]
A.append([2,3,4,5])
A
```

Out[111]:
```
[1, [2, 3, 4, 5]]
```

In [112…
```
# Create a list

L = ["Michael Jackson", 10.1, 1982]
L
```

Out[112]:
```
['Michael Jackson', 10.1, 1982]
```

In [115…
```
print('the same element using negative and positive indexing:\n Postive:',L[0],
'\n Negative:' , L[-3]  )
print('the same element using negative and positive indexing:\n Postive:',L[1],
'\n Negative:' , L[-2]  )
print('the same element using negative and positive indexing:\n Postive:',L[2],
'\n Negative:' , L[-1]  )
```

```
the same element using negative and positive indexing:
 Postive: Michael Jackson
 Negative: Michael Jackson
the same element using negative and positive indexing:
 Postive: 10.1
 Negative: 10.1
the same element using negative and positive indexing:
 Postive: 1982
 Negative: 1982
```

In [117…
```
L = ["Michael Jackson", 10.1, 1982]
L.append(['pop', 10])
L
```

Out[117]:
```
['Michael Jackson', 10.1, 1982, ['pop', 10]]
```

In [118…
```
# Use append to add elements to list

L.append(['a','b'])
L
```

Out[118]:
```
['Michael Jackson', 10.1, 1982, ['pop', 10], ['a', 'b']]
```

In [119…
```
# Change the element based on the index

A = ["disco", 10, 1.2]
A[0] = 'hard rock'
A
```

Out[119]:
```
['hard rock', 10, 1.2]
```

In [120…
```
# Delete the element based on the index

A = ["disco", 10, 1.2]
del(A[0])
A
```

Out[120]:
```
[10, 1.2]
```

In [121…
```
# Split the string, default is by space

'hard rock'.split()
```

Out[121]:
```
['hard', 'rock']
```

In [122…
```
# Split the string by comma

'A,B,C,D'.split(',')
```

Out[122]:
```
['A', 'B', 'C', 'D']
```

In [123…
```
#copy and clone
# Copy (copy by reference) the list A

A = ["hard rock", 10, 1.2]
B = A
print('A:', A)
print('B:', B)
```

```
A: ['hard rock', 10, 1.2]
B: ['hard rock', 10, 1.2]
```

In [127…
```
# Examine the copy by reference

print('B[0]:', B[0])
A[0] = "banana"
print('B[0]:', B[0])
```

```
B[0]: banana
B[0]: banana
```

In [128…
```
# Clone (clone by value) the list A

B = A[:]
B
```

Out[128]:
```
['banana', 10, 1.2]
```

In [131…
```
# Now if you change A, B will not change:
print('B[0]:', B[0])
A[0] = "hard rock"
print('B[0]:', B[0])
A
```

```
B[0]: banana
B[0]: banana
```

Out[131]:
```
['hard rock', 10, 1.2]
```

In [135…
```
# Create a list a_list, with the following elements 1, hello, [1,2,3] and True
a_list= [1, "hello", [1,2,3], True ]
a_list
```

Out[135]:
```
[1, 'hello', [1, 2, 3], True]
```

In [136…
```
#Find the value stored at index 1 of a_list.
a_list [1]
```

Out[136]:
```
'hello'
```

In [137…
```
#Retrieve the elements stored at index 1, 2 and 3 of a_list.
a_list [1:4]
```

Out[137]:
```
['hello', [1, 2, 3], True]
```

In [138…
```
#Concatenate the following lists A = [1, 'a'] and B = [2, 1, 'd']:
A = [1, 'a']
B = [2, 1, 'd']
A + B
```

Out[138]:
```
[1, 'a', 2, 1, 'd']
```

In [139…
```
# Create your first tuple

tuple1 = ("disco",10,1.2 )
tuple1
```

Out[139]:
```
('disco', 10, 1.2)
```

In [140…
```
# Print the type of the tuple you created

type(tuple1)
```

Out[140]:
```
tuple
```

In [141…
```
# Print the variable on each index

print(tuple1[0])
```

```
print(tuple1[1])
print(tuple1[2])
```

```
disco
10
1.2
```

In [142…
```
# Print the type of value on each index

print(type(tuple1[0]))
print(type(tuple1[1]))
print(type(tuple1[2]))
```

```
<class 'str'>
<class 'int'>
<class 'float'>
```

In [143…
```
# Use negative index to get the value of the last element

tuple1[-1]
```

Out[143]:
```
1.2
```

In [144…
```
# Concatenate two tuples

tuple2 = tuple1 + ("hard rock", 10)
tuple2
```

Out[144]:
```
('disco', 10, 1.2, 'hard rock', 10)
```

In [145…
```
# Slice from index 0 to index 2

tuple2[0:3]
```

Out[145]:
```
('disco', 10, 1.2)
```

In [146…
```
# Slice from index 3 to index 4

tuple2[3:5]
```

Out[146]:
```
('hard rock', 10)
```

In [147…
```
# Get the length of tuple

len(tuple2)
```

Out[147]:
```
5
```

In [149…
```
# A sample tuple

Ratings = (0, 9, 6, 5, 10, 8, 9, 6, 2)
Ratings
```

Out[149]:
```
(0, 9, 6, 5, 10, 8, 9, 6, 2)
```

In [150…
```
# Sort the tuple

RatingsSorted = sorted(Ratings)
RatingsSorted
```

Out[150]:
```
[0, 2, 5, 6, 6, 8, 9, 9, 10]
```

In [154…
```
# Create a nest tuple

NestedT =(1, 2, ("pop", "rock") ,(3,4),("disco",(1,2)))
NestedT
```

Out[154]:
```
(1, 2, ('pop', 'rock'), (3, 4), ('disco', (1, 2)))
```

In [155…
```
# Print element on each index

print("Element 0 of Tuple: ", NestedT[0])
print("Element 1 of Tuple: ", NestedT[1])
print("Element 2 of Tuple: ", NestedT[2])
print("Element 3 of Tuple: ", NestedT[3])
print("Element 4 of Tuple: ", NestedT[4])
```

```
Element 0 of Tuple:  1
Element 1 of Tuple:  2
Element 2 of Tuple:  ('pop', 'rock')
Element 3 of Tuple:  (3, 4)
Element 4 of Tuple:  ('disco', (1, 2))
```

In [157…
```
# Print element on each index

```

```
print("Element 0 of Tuple: ", NestedT[0])
print("Element 1 of Tuple: ", NestedT[1])
print("Element 2 of Tuple: ", NestedT[2][0])
print("Element 2 of Tuple: ", NestedT[2][1])
print("Element 3 of Tuple: ", NestedT[3][0])
print("Element 3 of Tuple: ", NestedT[3][1])
print("Element 4 of Tuple: ", NestedT[4][0])
print("Element 4 of Tuple: ", NestedT[4][1])
```

```
Element 0 of Tuple:  1
Element 1 of Tuple:  2
Element 2 of Tuple:  pop
Element 2 of Tuple:  rock
Element 3 of Tuple:  3
Element 3 of Tuple:  4
Element 4 of Tuple:  disco
Element 4 of Tuple:  (1, 2)
```

In [160…
```
print("Element 2, 0 of Tuple: ",   NestedT[2][0])
print("Element 2, 1 of Tuple: ",   NestedT[2][1])
print("Element 3, 0 of Tuple: ",   NestedT[3][0])
print("Element 3, 1 of Tuple: ",   NestedT[3][1])
print("Element 4, 0 of Tuple: ",   NestedT[4][0])
print("Element 4, 1 of Tuple: ",   NestedT[4][1])
```

```
pop
```

In [161…
```
print(NestedT[2][0])
```

```
pop
```

In [162…
```
print (NestedT[4][1])
```

```
(1, 2)
```

In [165…
```
# Print the first element in the second nested tuples
NestedT[2][1][0]
```

Out[165]:
```
'r'
```

In [166…
```
# Print the second element in the second nested tuples
NestedT [2][1][1]
```

Out[166]:
```
'o'
```

In [167…
```
# sample tuple

genres_tuple = ("pop", "rock", "soul", "hard rock", "soft rock", \
                "R&B", "progressive rock", "disco")
genres_tuple
```

Out[167]:
```
('pop',
 'rock',
 'soul',
 'hard rock',
 'soft rock',
 'R&B',
 'progressive rock',
 'disco')
```

In [173…
```
#Find the length of the tuple, genres_tuple:
len(genres_tuple)
```

Out[173]:
```
8
```

In [179…
```
#Access the element, with respect to index 3:
genres_tuple[3]
```

Out[179]:
```
'hard rock'
```

In [180…
```
#Use slicing to obtain indexes 3, 4 and 5:
genres_tuple[3:6]
```

Out[180]:
```
('hard rock', 'soft rock', 'R&B')
```

In [182…
```
#Find the first two elements of the tuple genres_tuple:
genres_tuple[0:2]
```

Out[182]:
```
('pop', 'rock')
```

In [185…
```
#Find the first index of "disco":

genres_tuple.index("disco")
```

Out[185]:
```
7
```

In [187…
```
#Generate a sorted List from the Tuple C tuple=(-5, 1, -3):
```

```
C_tuple=(-5, 1, -3)
C_tuple1 = sorted(C_tuple)
C_tuple1
```

Out[187]:
```
[-5, -3, 1]
```

In [188…
```
say_what=('say',' what', 'you', 'will')
#what is the result of the following
say_what[-1]
```

Out[188]:
```
'will'
```

In [189…
```
#Consider the following tuple
A=(1,2,3,4,5)
#What is the result of the following:
A[1:4]
```

Out[189]:
```
(2, 3, 4)
```

In [190…
```
#Consider the following tuple
A=(1,2,3,4,5)
#what is the result of the following:
len(A)
```

Out[190]:
```
5
```

In [191…
```
#Consider the following list
B=[1,2,[3,'a'],[4,'b']]
#what is the result of the following:
B[3][1]
```

Out[191]:
```
'b'
```

In [198…
```
#What is the length of the list
A = [1]
#after the following operation:
A.append([2,3,4,5])
A
len(A)
```

Out[198]:
```
2
```

In [ ]: