Python_Practice_Day_3

December 30, 2021

1 Loops

[1]:
```
print(1)
print(2)
print(3)
print(4)
print(5)
print(6)
print(7)
print(8)
print(9)
print(10)
```

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

[2]:
```
# Observe the pattern.
# Loop is used to perform any action repeatedly, in terms of a certain␣
 ↪condition.
# Two types of loops in python.
```

[3]:
```
# for loop:
# Print numbers from 1 to 10
```

[4]:
```
for i in range(1,10):
    print(i)
```

```
1
2
3
```

```
4
5
6
7
8
9
```

[5]:
```
# range excludes the boundary value, so 10 is not printed.
```

[6]:
```
for i in range(1,11):
    print(i)
```

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

[7]:
```
for i in range(1,101):
    print(i,end = " ")
```

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57
58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84
85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100
```

[8]:
```
# continue keyword to skip a value
for i in range(1,11):
    if (i == 5):
        continue
    print(i)
```

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

[9]:
```
for i in range(1,11):
    if (i == 5 or i == 10):
```

```
        continue
    print(i)
```

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

[11]:
```
name = "Sri Harsha"
for i in name:
    print(i, end = "")
```

```
Sri Harsha
```

[12]:
```
name = "Sri Harsha"
for i in name:
    if (i == 'S'):
        continue
    print(i)
```

```
r
i

H
a
r
s
h
a
```

[13]:
```
#break
for i in range(1,11):
    if (i == 5):
        break     # whatever is executed before 5 will be printed.
    print(i)
```

```
1
2
3
4
```

[14]:
```
# Task
# create atleast 5 different scenarios for implementing loops.
```

[15]:
```
# while loop
```

[16]:
```
i = 1
while(i < 11):
    print(i)
    i = i + 1
```

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

[17]:
```
i = 1                               # initial value
while(i < 101):                     # boolean value
    print(i , end = " ")            # printing the value
    i = i + 1                       # incrementing the value
```

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57
58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84
85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100
```

2 Functions

[1]:
```
print("Harsha")
```

```
Harsha
```

[ ]:
```

```

[2]:
```
num1 = int(input("Enter a number"))
num2 = int(input("Enter another number"))
num3 = num1 + num2
print(num3)
```

```
Enter a number10
Enter another number20
30
```

[ ]:
```

```

[ ]:
```
# Function is a block of code which gets executed whenever we call it.
# There are two types of functions:
    # pre defined function -- print(), input(), upper()
    # user defined functions
```

3 User defined function:

[3]:
```
# function has 3 compenents:
    # function definition
    # body
    # calling statement
```

[4]:
```
# write a program to define function and print something

def myfunction():                              # function definition
    print("Hey, Here is your first function") # function body
```

[5]:
```
myfunction() # calling statement
```

```
Hey, Here is your first function
```

[6]:
```
# example 2

def sum(num1 ,  num2):
    print(num1 + num2)
```

[7]:
```
sum()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_12652/3398259504.py in <module>
----> 1 sum()

TypeError: sum() missing 2 required positional arguments: 'num1' and 'num2'
```

[8]:
```
sum(50 , 100)
```

```
150
```

[9]:
```
sum(400,800)
```

```
1200
```

[10]:
```
# example 3
def details(fname,lname):
    print(fname,lname)
details(sri,harsha)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_12652/4156577300.py in <module>
      2 def details(fname,lname):
      3     print(fname,lname)
```

```
----> 4 details(sri,harsha)

NameError: name 'sri' is not defined
```

[11]:
```
details("sri","harsha")
```

```
sri harsha
```

[12]:
```
# function with return keyword
```

[13]:
```
def addition(num1,num2):
    return num1 + num2
```

[14]:
```
print(addition(200,500))
```

```
700
```

4 Numpy

[15]:
```
# Numpy stands for numerical python.
# python interpreter directly do not support array.
# numpy is a python library used for working with arrays.
```

[16]:
```
# Why not list and why numpy ?
    # Numpy is faster because most of the code for numpy is written in C␣
 ↪language.

# the array object in numpy is called ndarray.
```

[19]:
```
import numpy as np   # np -- alias
arr = np.array([1,2,3,4,5])
print(arr)
```

```
[1 2 3 4 5]
```

[20]:
```
list1 = [1,2,3,4,5]
print(list1)
```

```
[1, 2, 3, 4, 5]
```

[ ]:
```
# It might seem to be same answer for limited number of values.
# But for thousands of values , array plays a crucial role.
```

[21]:
```
# version of numpy
print(np.__version__)
```

```
1.20.3
```

[22]:
```
# checking type
arr = np.array([1,2,3,4,5])
print(type(arr))   # n dimensional array
```

```
<class 'numpy.ndarray'>
```

[23]:
```
# can we pass tuple inside array ?
import numpy as np   # np -- alias
arr = np.array((1,2,3,4,5))
print(arr)
```

```
[1 2 3 4 5]
```

5 Dimensions in numpy array

[ ]:
```
# A dimension in array is one level of array depth. (nested array)
```

[24]:
```
# 0-D array --> scalars
# Each elemnt of 1-D array is a 0-D array

arr = np.array(500)
print(arr)
```

```
500
```

[25]:
```
# 1-D array

arr = np.array([1,2,3,4,5])
print(arr)
```

```
[1 2 3 4 5]
```

[28]:
```
# 2-D array --> 2 dimensional

arr = np.array([[1,2,3,4,5],[6,7,8,9,10]])
print(arr)
```

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

[29]:
```
# 3-D array

arr = np.array([[[1,2,3],[4,5,6]],[[7,8,9],[10,11,12]]])
print(arr)
```

```
[[[ 1  2  3]
  [ 4  5  6]]

 [[ 7  8  9]
  [10 11 12]]]
```

[ ]:
```
# Task -- Create 10 different 3d arrays
```

[30]:
```
# to check the dimension of the arrays:

arr = np.array([[[1,2,3],[4,5,6]],[[7,8,9],[10,11,12]]])
print(arr)
print(arr.ndim)
```

```
[[[ 1  2  3]
  [ 4  5  6]]

 [[ 7  8  9]
  [10 11 12]]]
3
```

[32]:
```
arr = np.array([[1,2,3,4,5],[6,7,8,9,10]])
print(arr)
print(arr.ndim)
```

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

[33]:
```
arr = np.array([1,2,3,4,5])
print(arr)
print(arr.ndim)
```

```
[1 2 3 4 5]
1
```

[34]:
```
arr = np.array(500)
print(arr)
print(arr.ndim)
```

```
500
0
```

[35]:
```
# all the arrays have to be in a square matrix
# to facilitate the operation like multiplication etc.
```

[36]:
```
# How to access elements in the array ?

arr = np.array([1,2,3,4,5])
print(arr[3])
```

```
4
```

[37]:
```
arr = np.array([1,2,3,4,5])
print(arr[0] + arr[3])
```

```
5
```

[39]:
```
# accessing elements in 2-D array

arr = np.array([[1,2,3,4,5],[6,7,8,9,10]])
print(arr[1,1])
print(arr[0,2])
```

```
7
3
```

[41]:
```
# accessing elements in 3-D array

arr = np.array([[[1,2,3],[4,5,6]],[[7,8,9],[10,11,12]]])
print(arr[0,0,2])
print(arr[1,1,0])
```

```
3
10
```