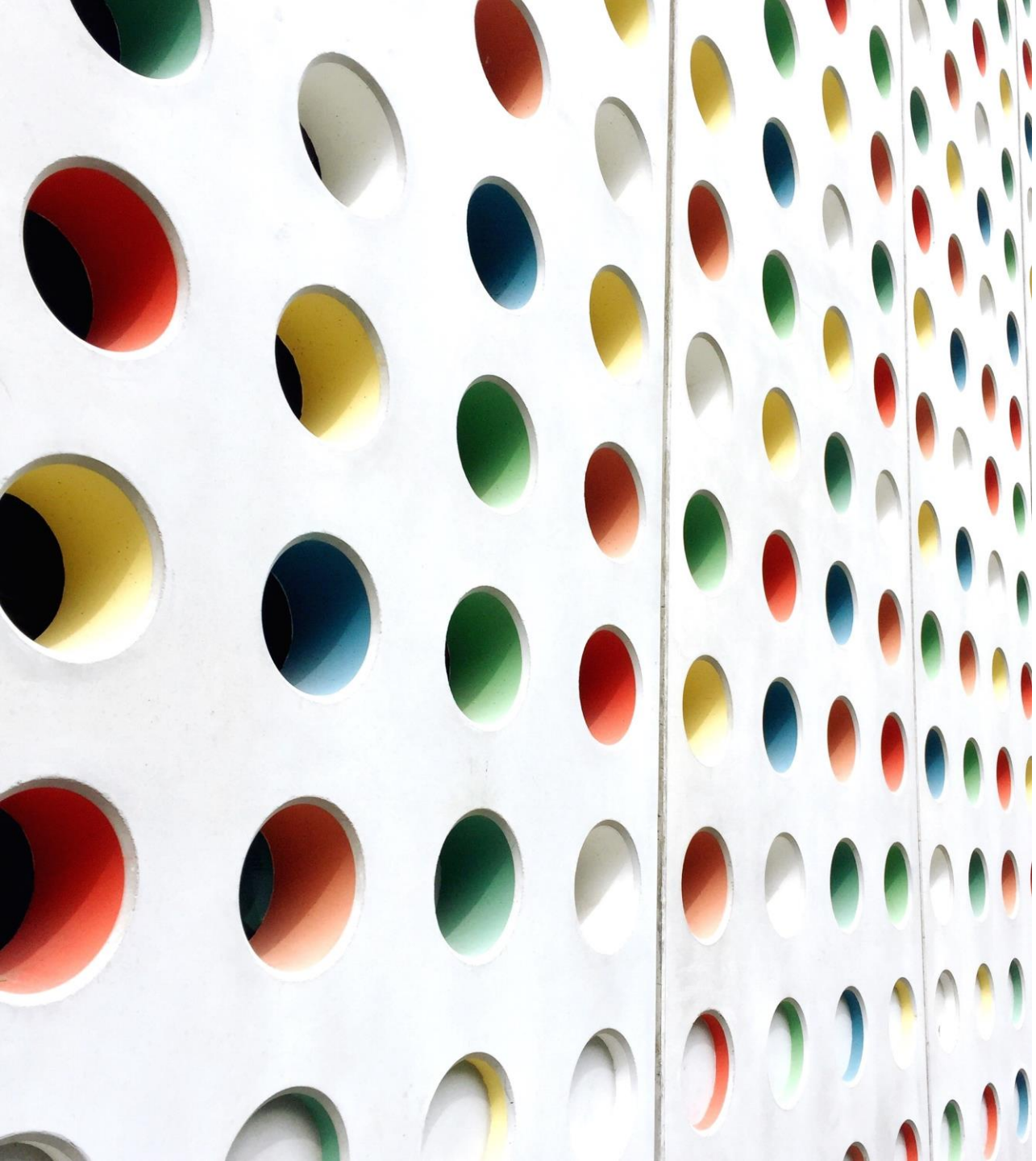

Functions in Python

NAGULLAS K S

SAMANYU MOHANTY

ROOPAK MAYYA

TEJAS C S

AMRITHA SATHIADEVAN

Type of Functions

- User defined
- Built-in
- Lambda
- Recursion

Python Built-in functions

Python has a set of built-in functions or methods that can be applied to various objects like strings, lists, dictionary, tuple, Set

- dir() – it returns a list of attributes of the specified object and its methods

```
1 dir(list)
```

```
['__add__',
 '__class__',
 '__contains__',
 '__delattr__',
 '__delitem__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__getitem__',
 '__gt__',
 '__hash__',
 '__iadd__',
 '__imul__',
 '__init__',
 '__init_subclass__',
 '__repr__',
 '__reversed__',
 '__rmul__',
 '__setattr__',
 '__setitem__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 'append',
 'clear',
 'copy',
 'count',
 'extend',
 'index',
 'insert',
 'pop',
 'remove',
 'reverse',
 'sort']
```

- help() – it returns interactive help or help documentation about specified object

```
1 help(dict)
```

```
Help on class dict in module builtins:

class dict(object)
 |  dict() -> new empty dictionary
 |  dict(mapping) -> new dictionary initialized from a mapping object's
 |      (key, value) pairs
 |  dict(iterable) -> new dictionary initialized as if via:
 |      d = {}
 |      for k, v in iterable:
 |          d[k] = v
 |  dict(**kwargs) -> new dictionary initialized with the name=value pairs
 |      in the keyword argument list.  For example:  dict(one=1, two=2)
 |
 |  Built-in subclasses:
 |      StgDict
 |
 |  Methods defined here:
 |
 |  __contains__(self, key, /)
 |      True if the dictionary has the specified key, else False.
```

- enumerate() – it returns an object of pairs of index and a value from specified iterable argument

```
1 coding_lang = ['Python', 'Java', 'R', 'Ruby', 'C++']
2 enum_obj = enumerate(coding_lang, start = 1)
3 print(list(enum_obj))
```

```
[(1, 'Python'), (2, 'Java'), (3, 'R'), (4, 'Ruby'), (5, 'C++')]
```

- format() – it formats a value into specified format

```
1 iphone12_price = 84900
2 print("Cost of iPhone12 is ₹{:,.2f}".format(iphone12_price))
```

```
Cost of iPhone12 is ₹84,900.00
```

- filter() – it returns an iterator from iterable items for which the specified function is true

```
1  import random
2  age = random.sample(range(20, 90), 25)
3
4  def eligibility(n):
5      if n >= 45:
6          return True
7      else:
8          return False
9
10 get_vaccine = list(filter(eligibility, age))
11 print(get_vaccine)
```

```
[77, 71, 49, 50, 81, 74, 76, 66, 69, 83, 85, 64, 54, 72]
```

Other commonly used Built-in functions:

abs(), chr(), dict(), eval(), float(), input(), isinstance(), len(), list(), map(), max(), min(), print(), range(), round(), sum(), type(), zip()

User Defined Functions

- def keyword is used to declare user defined functions.
- An indented block of statements follows the function name and arguments which contains the body of the function.
- Parameters can also be set to contain default values
- Return statement is optional in python, but if return doesn't have any expression it return None value.

User Defined Function

```
def userdefined(x):
    print ('square of number is ',x**2)

userdefined(2)
```

```
square of number is  4
```

Parameterized functions

```
def product (num1,num2):
    return num1*num2

num1,num2=10,20
ans = product(num1,num2)
print(ans)
```

```
200
```

Keyword Arguments

```
def student(firstname, lastname):
    print(firstname, lastname)

student(firstname ='Hello', lastname ='Everyone')
student(lastname ='Good day', firstname ='Have a')
```

```
Hello Everyone
Have a Good day
```

Variable length parameter

```
def myFun1(*argv): # stores value as a list
    for arg in argv:
        print (arg,end ='\t')

def myFun2(**kwargs): # stores value as a dict
    for key, value in kwargs.items():
        print ("% s - % s" %(key, value))

print("Result of * args: ")
myFun1('Hello', 'Welcome', 'to', 'Presentation')
print("\nResult of **kwargs")
myFun2(first ='Functions', mid ='with example', last ='presentation')
```

```
Result of * args:
Hello   Welcome to      Presentation
Result of **kwargs
first - Functions
mid - with example
last - presentation
```

Passing parameter value by refrence

```
def myFun(x, arr):
    print("Inside function")
    x += 10
    print("Value received", x, "Id", id(x))
    print("List received", arr, "Id", id(arr))

x,arr = 10,[1, 2, 3]
print("Before calling function")
print("Value passed", x, "Id", id(x))
print("Array passed", arr, "Id", id(arr))
print()
myFun(x, arr)
print("\nAfter calling function")
print("Value passed", x, "Id", id(x))
print("Array passed", arr, "Id", id(arr))
```

```
Before calling function
Value passed 10 Id 140733398132800
Array passed [1, 2, 3] Id 1716414153984

Inside function
Value received 20 Id 140733398133120
List received [1, 2, 3] Id 1716414153984

After calling function
Value passed 10 Id 140733398132800
Array passed [1, 2, 3] Id 1716414153984
```

Lambda Functions – Anonymous , Single expression Function

Usage :Lambda arguments : expression

-any number of input arguments
-single and small expression
-implicit return statement
-lambda instead of def keyword
-throw away function as it has no name
-use for short and simple code
-used with built-in functions that take function as an argument

Simple Lambda usage

```
y=lambda x:x*x
print(y(4))
```

```
16
```

Recursion using Lambda

```
factorial=lambda i: 1 if i==0 else i*factorial(i-1)
factorial(5)
```

```
120
```

Print using Lambda

```
x="lambda cannot print me"
y=lambda x: print(x)
print(y)
```

```
<function <lambda> at 0x0000023D095031F0>
```

```
x="lambda can print me if i am passed through a function"
y=(lambda x: print(x))(x)
print("Return value of lambda is",y)
```

```
lambda can print me if i am passed through a function
Return value of lambda is None
```

Single liner using Lambda

```
list1=[1,2,3,4,5]
def fn1(x,y,l):
    list2=[]
    for f in l:
        list2.append(f*x/y)
    return(list2)
res=fn1(10,2,list1)
print(res)
```

```
[5.0, 10.0, 15.0, 20.0, 25.0]
```

```
list2=list(map(lambda l,a=10,b=2:l*a/b,list1))
print(list2)
```

```
[5.0, 10.0, 15.0, 20.0, 25.0]
```

Lambda with Map, Filter , Reduce & Accumulate

```
a=[4,2,3,4,5]
map_ex=map(lambda x:x*3,a)
print(list(map_ex))
filter_ex=filter(lambda x:x>3,a)
print(set(filter_ex))

from functools import reduce
reduce_ex=reduce(lambda x,y:x*y,a)
print(reduce_ex)

import itertools as it
accumulate_ex=it.accumulate(a,lambda x,y:x*y)
print(list(accumulate_ex))
```

```
[12, 6, 9, 12, 15]
{4, 5}
480
[4, 8, 24, 96, 480]
```

Recursion Functions

- A programming strategy that helps in solving certain critical problems.
- It basically means a function calling itself within itself with reduced problem intensity with an exit condition in place.
- A simple example would be to find a factorial of a number.
- It adds a lot of memory stress on the system if the number of iterations is too huge as it keeps adding function overhead.

- Popular examples include
- Finding Factorial, solving the "Tower of Hanoi" problem.
- Quick sort of a list
- Finding the Fibonacci Series. etc

```
n1 = int(input())
def factorial_recursion(n):
    if(n==1):
        return n
    return n*factorial_recursion(n-1)

res = factorial_recursion(n1)
print(res)
```

```
6
720
```