18.SpecialFunctions

May 9, 2020

[1]:
```
lang = [ 'java', 'c', 'c++', 'ruby', 'perl', 'python']
```

Linear Search

[7]:
```
def linear_search(lst, key):
    for i,item in enumerate(lst): # key
        if item == key: # key
            return i
    return -1
```

[10]:
```
linear_search(lang, 'python')
```

[10]: 5

[11]:
```
lang
```

[11]: ['java', 'c', 'c++', 'ruby', 'perl', 'python']

[15]:
```
def linear_recursion(lst, n, key, i=0):
    if i < n:
        if lst[i] == key:
            return i
        else:
            return linear_recursion(lst, n, key, i+1)
    return -1
```

[19]:
```
linear_recursion(lang, len(lang), 'abc')
```

[19]: -1

Advance Functions in Python

Lambda

Anonymous

Inline Function

syntax:

```
fun_name = lambda arg1, arg2, ...: return_value with statement
```

[20]:
```
def hello():
    print("Hello World")
```

[21]:
```
hello
```

[21]:
```
<function __main__.hello()>
```

[22]:
```
hello()
```

```
Hello World
```

[23]:
```
h = lambda : print("Hello World")
```

[24]:
```
h
```

[24]:
```
<function __main__.<lambda>()>
```

[25]:
```
h()
```

```
Hello World
```

[26]:
```
def hello(name):
    print("Hello , ", name)
```

[27]:
```
hello('sachin')
```

```
Hello ,  sachin
```

[28]:
```
h = lambda name : print("Hello, ", name)
```

[29]:
```
h('sachin')
```

```
Hello,  sachin
```

[30]:
```
def square(num):
    return num ** 2
```

[31]:
```
ans = square(5)
print(ans)
```

```
25
```

[32]:
```
sq = lambda num : num**2
```

[35]:
```
ans = sq(5)
print(ans)
```

```
25
```

[34]:
```
(lambda num: num**2)(int(input("Enter a number: ")))
```

```
Enter a number: 12
```

[34]:
```
144
```

[36]:
```
print("hello") if True else print("bye")
```

```
hello
```

[42]:
```
a,b,c=1,7,6

print(a) if a >= b and a >= c else print(b) if b >= c else print(c)
```

```
7
```

[43]:
```
def biggest(a, b, c):
    if a >= b and a >=c:
        return a
    elif b >= c:
        return b
    else:
        return c
```

[44]:
```
biggest(13, 10, 15)
```

[44]:
```
15
```

[45]:
```
big = lambda a, b, c : a if a >= b and a >= c else b if b >= c else c
```

[46]:
```
big(13, 10, 15)
```

[46]:
```
15
```

List Comprehnsion

[47]:
```
[ var**2 for var in range(1, 11)]
```

[47]:
```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

[48]:
```
def square(lst):
    new_list = []
    for item in lst:
        new_list.append(item**2)
    return new_list
```

[49]:
```
square([1, 2, 3, 4, 5])
```

[49]:
```
[1, 4, 9, 16, 25]
```

[50]:
```
sq = lambda lst: [ item**2 for item in lst ]
```

[51]:
```
sq([1, 2, 3 , 4, 5])
```

[51]:
```
[1, 4, 9, 16, 25]
```

Recursion

[54]:
```
def factorial(num):
    if num:
        return num*factorial(num-1)
    return 1
```

[55]:
```
factorial(5)
```

[55]:
```
120
```

[56]:
```
fact = lambda num: num*fact(num-1) if num else 1
```

[57]:
```
fact(5)
```

[57]:
```
120
```

Prime Using Recursion

[59]:
```
from math import sqrt, ceil
```

[68]:
```
def prime(num, check=2):
    if num <= 1:
        return False
    else:
        if check == ceil(sqrt(num))+1:
            return True
        elif num % check == 0:
            return False
        else:
            return prime(num, check+1)
```

[69]:
```
prime(1)
```

[69]:
```
False
```

[70]:
```
prime(2)
```

[70]:
```
False
```

[71]:
```
prime(3)
```

[71]:
```
True
```

[72]:
```
prime(13)
```

[72]:
```
True
```

[73]:
```
prime(12)
```

[73]:
```
False
```

[74]:
```
prime(121)
```

[74]:
```
False
```

[75]:
```
prime(127)
```

[75]:
```
True
```

[76]:
```
ceil(1.1)
```

[76]:
```
2
```

[77]:
```
ceil(1.01)
```

[77]:
```
2
```

[92]:
```
def prime(num, check=2):
    if num <= 1:
        return False
    if num <= 3:
        return True
    if check == ceil(sqrt(num))+1:
        return True
    elif num % check == 0:
        return False
    else:
        return prime(num, check+1)
```

[97]:
```
prime(127)
```

[97]:
```
True
```

[102]:
```
p = lambda num, check=2: False if num <= 1 else True if num <= 3 else True if \
check == num//2 else False if num % check == 0 else p(num, check+1)
```

[103]:
```
p(121)
```

[103]: False

[104]:
```
p(127)
```

[104]: True

[105]:
```
p(123)
```

[105]: False

[110]:
```
print((lambda num, check=2: False if num <= 1 else True if num <= 3 else True␣
 ↪if \
check == num//2 else False if num % check == 0 else p(num, check+1)␣
 ↪)(int(input("Enter a Number: "))) )
```

```
Enter a Number: 1
False
```

[113]:
```
[var for var in range(10) if var % 2 == 0]
```

[113]: [0, 2, 4, 6, 8]

[114]:
```
from random import randint
```

[115]:
```
l = [ randint(1, 50) for var in range(10) ]
```

[116]:
```
l
```

[116]: [35, 1, 1, 34, 24, 26, 45, 12, 21, 6]

[117]:
```
even = lambda lst: [ item for item in lst if item % 2 == 0]
```

[119]:
```
even(l)
```

[119]: [34, 24, 26, 12, 6]

[121]:
```
eve_list =  (lambda lst: [ item for item in lst if item % 2 == 0])(l)
```

[122]:
```
print(eve_list)
```

```
[34, 24, 26, 12, 6]
```

[123]:
```
a = [
      #c1  #c2    #3
    [ 1,    2,    3], # r1
    # 0,0   0,1   0,2
    [ 4,    5,   6], # r2
    # 1,0  1,1   1,2
    [ 7, 8, 9], # r3

    [ 10, 11, 12] # r4

]

# n = 4
# m = 3
```

[124]:
```
n = int(input("Enter number of rows: "))
m = int(input("Enter number of cols: "))

arr = [ ]

for row in range(n):
    r = []
    for col in range(m):
        value = int(input(f"arr[{row}][{col}]: "))
        r.append(value)
    arr.append(r)
```

```
Enter number of rows: 2
Enter number of cols: 3
arr[0][0]1
arr[0][1]2
arr[0][2]3
arr[1][0]4
arr[1][1]5
arr[1][2]6
```

[125]:
```
print(arr)
```

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

[126]:
```
n = int(input("Enter number of rows: "))
m = int(input("Enter number of cols: "))

arr = [
       [ int(input(f"arr[{row}][{col}]: "))  for col in range(m) ]
       for row in range(n)
      ]
```

```
Enter number of rows: 2
Enter number of cols: 3
arr[0][0]: 1
arr[0][1]: 2
arr[0][2]: 3
arr[1][0]: 4
arr[1][1]: 5
arr[1][2]: 6
```

[127]:
```
print(arr)
```

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

[129]:
```
arr_input = lambda n, m : [  [ int(input(f"arr[{row}][{col}]: "))  for col in␣
 ↪range(m) ]     for row in range(n) ]
arr = arr_input(3, 3)
print(arr)
```

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

[135]:
```
%%writefile matrix_input.py
print((lambda n, m : [  [ int(input(f"arr[{row}][{col}]: "))  for col in␣
 ↪range(m) ]    \
                      for row in range(n) ])(int(input("Enter rows: ")),␣
 ↪int(input("Enter cols: "))))
```

```
Writing matrix_input.py
```

[136]:
```
pwd
```

[136]:
```
'C:\\Batches\\Batch_7pm_online'
```

[134]:
```
x = 10; y = 20; print(x, y)
```

```
10 20
```

0.1 Map

[137]:
```
l1 = [ 1, 2, 3, 4, 5]
l2 = [ 6, 5, 4, 1, 2]
```

[138]:
```
r = []

for i in range(len(l1)): # 0, 1, 2, 3, 4
    item1 = l1[i]
    item2 = l2[i]
    r.append(item1**2+item2**2)
```

[139]:
```
print(r)
```

```
[37, 29, 25, 17, 29]
```

[140]:
```
x = [ '1', '2', '3', '4', '5']
```

[142]:
```
y = []
for item in x:
    y.append(int(item))

print(x)
print(y)
```

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

[143]:
```
x = [ '1', '2', '3', '4', '5']
```

[144]:
```
y = [ int(item)   for item in x ]
```

[145]:
```
print(y)
```

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

[146]:
```
y = map(int, x)
```

[147]:
```
print(y)
```

```
<map object at 0x000002370CE29E88>
```

[148]:
```
print(*y)
```

```
1 2 3 4 5
```

[149]:
```
x = [ '1', '2', '3', '4', '5']
```

```
y = [ int(item) for item in x ]
print(y)
```

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

Syntax:

```
map_object = map( func, iterable)
```

[150]:
```
y  = list( map( int,  x ) )
print(y)
```

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

[151]:
```
data = [ '10', '0b10101', '0xabc', '0o123']

encode = [ 10, 2, 16, 8 ]
```

[152]:
```
int('abc', 16)
```

[152]: 2748

[153]:
```
decimal = list( map(  int, data, encode )  )
print(decimal)
```

```
[10, 21, 2748, 83]
```

[154]:
```
a = [ randint(1, 10) for var in range(10)]
b = [ randint(1, 10) for var in range(10)]
c = [ randint(1, 10) for var in range(10)]
```

[155]:
```
print(a, b, c ,sep='\n')
```

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

[156]:
```
def biggest(a, b, c):
    if a >= b and a >= c:
        return a
    elif b >= c:
        return b
    else:
        return c
```

[161]:
```
bb  = lambda a,b,c : a if a>=b and a>=c else b if b>=c else c
```

[162]:
```
big = list( map( bb, a, b, c ))
print(big)
```

```
[8, 6, 10, 9, 5, 10, 6, 9, 10, 7]
```

[163]:
```
big = list( map( lambda a,b,c : a if a>=b and a>=c else b if b>=c else c, a, b,␣
 ↪c))
print(big)
```

```
[8, 6, 10, 9, 5, 10, 6, 9, 10, 7]
```

[166]:
```
x = input("Five space seprated numbers: ").split(' ')
```

```
Five space seprated numbers: 12 15 16 23 45
```

[167]:
```
x
```

[167]:
```
['12', '15', '16', '23', '45']
```

[169]:
```
y = list(map(int,  x ))
print(x)
print(y)
```

```
['12', '15', '16', '23', '45']
[12, 15, 16, 23, 45]
```

[170]:
```
arr = list( map( int, input().split() ) )
print(arr)
```

```
1 2 3
[1, 2, 3]
```

[171]:
```
# m --> columns

arr = [ list(map(int, input(f"row[{row}]").split()))  for row in␣
 ↪range(int(input("Rows: "))) ]
```

```
Rows: 4
row[0]1 2 3 4
row[1]2 3 4 5
row[2]3 4 5 6
row[3]4 5 6 7
```

[172]:
```
print(arr)
```

```
[[1, 2, 3, 4], [2, 3, 4, 5], [3, 4, 5, 6], [4, 5, 6, 7]]
```

filter

reduce

closures and decorators

[ ]: