PACKAGES

In [ ]:
```
import datetime
datetime.datetime.now()
from datetime import datetime
```

In [8]:
```
import math
```

In [9]:
```
math.sin(3.14)
```

Out[9]:
```
0.0015926529164868282
```

In [4]:
```
import random
```

In [5]:
```
random.randint(1,10)
```

Out[5]:
```
3
```

In [6]:
```
import random as rd
rd.randint(1,10)
```

Out[6]:
```
8
```

In [ ]:
```
#import only cos/sin from each
from math import cos
from math import sin
```

In [11]:
```
from math import cos,sin
```

In [13]:
```
import math
import random
```

In [ ]:
```
#import math ,random #not good
```

In [15]:
```
import datetime
```

In [17]:
```
from datetime import datetime
```

In [18]:
```
dir(datetime)
```

Out[18]:
```
['__add__',
 '__class__',
 '__delattr__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__getstate__',
 '__gt__',
 '__hash__',
 '__init__',
 '__init_subclass__',
 '__le__',
 '__lt__',
 '__ne__',
 '__new__',
 '__radd__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__rsub__',
 '__setattr__',
 '__sizeof__',
 '__str__',
 '__sub__',
 '__subclasshook__',
 'astimezone',
 'combine',
 'ctime',
 'date',
 'day',
 'dst',
 'fold',
 'fromisocalendar',
 'fromisoformat',
 'fromordinal',
 'fromtimestamp',
 'hour',
 'isocalendar',
 'isoformat',
 'isoweekday',
 'max',
 'microsecond',
 'min',
 'minute',
 'month',
 'now',
 'replace',
 'resolution',
 'second',
 'strftime',
 'strptime',
 'time',
 'timestamp',
 'timetuple',
 'timetz',
 'today',
 'toordinal',
 'tzinfo',
 'tzname',
 'utcfromtimestamp',
 'utcnow',
```

```
 'utcoffset',
 'utctimetuple',
 'weekday',
 'year']
```

In [20]:
```
import time
```

In [25]:
```
time.time()
```

Out[25]:
```
1767924596.9955287
```

In [26]:
```
# what is the time required to run a code?
start_time = time.time()
time.sleep()
end_time = time.time()
start_time = time.time()
end_time - start_time
```

Out[26]:
```
-7.963180541992188e-05
```

In [ ]:
```
# power of jupyter
```

In [27]:
```
%%time # notebook magic function
a = 10
```

```
CPU times: user 5 µs, sys: 0 ns, total: 5 µs
Wall time: 9.06 µs
```

In [29]:
```
%%timeit?
a = 10
```

In [30]:
```
time?
```

serialization

In [31]:
```
import json
```

In [39]:
```
a = [1,2,3,4]
type(a)
```

Out[39]:
```
list
```

In [34]:
```
 val = json.dumps(a)
```

In [37]:
```
val
```

Out[37]:
```
'[1, 2, 3, 4]'
```

In [40]:
```
type(val)
```

Out[40]:
```
str
```

In [38]:
```
b = json.loads(val)
```

Out[38]:
```
[1, 2, 3, 4]
```

In [ ]:
```
# loads / dumps > from/to string
```

In [42]:
```
with open("date.json","w")as f:
    json.dump(a,f)
```

In [43]:
```
with open("date.json","r")as f:
    b = json.load(f)
```

In [44]:
```
b
```

Out[44]:
```
[1, 2, 3, 4]
```

In [51]:
```
d = {"a": 1, "b": 2, "c": 3}
import json
json.dump(d, open("data.json", "w"))
```

In [52]:
```
print(d)
```

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

In [59]:
```
yes = {"isStudent": true,"isMarried": false}
import json
json.dump(yes,open("data.json","w"))
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[59], line 1
----> 1 yes = {"isStudent": true,"isMarried": false}
      2 import json
      3 json.dump(yes,open("data.json","w"))

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

In [ ]:
```
#save username,score and dateline to json file
#load the data from file and show the data in which the score was made like
#date | name | score
#2026-01-10|ram |100
```

In [71]:
```
import json
import datetime

data = [
    {
    "datetime":datetime.datetime.now().isoformat(), #But datetime.datetime.n
        ##which JSON cannot serialize So we convert datetime to string using
    "username":"ram",
    "score":"100"}

]

with open("scores.json","w")as file:
    json.dump(data,file) #converts python data to json format.
```

In [74]:
```
import json

with open("scores.json", "r")as file:
   loaded = json.load #now loading json into python
print("date | name | score")
print(loaded)
```

```
date | name | score
<function load at 0x7994f4226700>
```

In [77]:
```
False and hdhihysfi
```

Out[77]:
```
False
```

In [78]:
```
True or jpayetehi
```

Out[78]:
```
True
```

In [ ]:
```
if 0:
    print("test")
```

In [80]:
```
1 or 0
```

Out[80]:
```
1
```

In [81]:
```
0 or 10
```

Out[81]:
```
10
```

In [82]:
```
a = []
if a:
    print("i am  not empty")
else:
    print("i am empty")
```

```
i am empty
```

In [83]:
```
id(a)
```

Out[83]:
```
133680460815616
```

In [84]:
```
id(b)
```

Out[84]:
```
133680460813760
```

In [85]:
```
a, *b = 1, 2,3,4,5
```

In [86]:
```
a
```

Out[86]:
```
1
```

In [87]:
```
b
```

Out[87]:
```
[2, 3, 4, 5]
```

In [1]:
```
#wap to count the occurance of number in a list
#a = [1,2,3,1,1,1,2,3,4,3]
#count 1->4,2->3,3->3,4->1

a = [1,2,3,1,1,1,2,3,4,3]
count={}
for num in a:
    if num in count:
        count[num] +=1
    else:
        count[num]=1
for k,v in count.items():
```

```
    print(f"{k}-> {v}")


#k → number (key)

#v → occurrence count (value)

#count.items() gives all key–value pairs

#f"{k}-> {v}" prints in required format
```

```
1-> 4
2-> 2
3-> 3
4-> 1
```

In [2]:
```
a = [1,2,3,1,1,1,2,3,4,3]
count={}
for i in a:
        count[i] =count.get(i,0) + 1
count
```

Out[2]:
```
{1: 4, 2: 2, 3: 3, 4: 1}
```

In [3]:
```
from collections import defaultdict
```

In [4]:
```
a = [1,2,3,1,1,1,2,3,4,3]
count = defaultdict(int)
for i in a:
    count[i] += 1
count
```

Out[4]:
```
defaultdict(int, {1: 4, 2: 2, 3: 3, 4: 1})
```

In [5]:
```
from collections import Counter
```

In [6]:
```
a = [1,2,3,1,1,1,2,3,4,3]
count = Counter(a)
print(count)
```

```
Counter({1: 4, 3: 3, 2: 2, 4: 1})
```

In [7]:
```
#function

'''python
def function_names(arguments,arg1,arg2,..)
    processing
    return something
'''
```

Out[7]:
```
'python\ndef function_names(arguments,arg1,arg2,..)\n    processing\n    re
turn something\n'
```

In [13]:
```
def print_x():
    print("X")
print_x()
```

```
X
```

In [12]:
```
def print_star(n):
    print("*" * n)

print_star(20)
```

```
********************
```

In [14]:
```
def print_char(char):
    print(char)
print_char("A")
```

```
A
```

In [16]:
```
def print_char(char,n):
    print(char * n)
print_char("A",1)
```

```
A
```

In [17]:
```
def func(a,b,c):
    return a + b + c
```

In [18]:
```
x = func(1,2,3)
print(x)
```

```
6
```

In [19]:
```
x = func(a = 1,b= 7,c= 3)
print(x)
```

```
11
```

In [20]:
```
#print(1,2,3, sep="backslasht")
```

In [21]:
```
#x  = func(a=1,2,3) not done
```

In [22]:
```
func(1, c=2, b=3) #good
```

Out[22]:
```
6
```

In [24]:
```
def func(a,b,c):
    return a + b, a*c
func(1,2,3)
```

Out[24]:
```
(3, 3)
```

In [25]:
```
#wap to get sum and diff of two numbers & explore diff ways of calling the f
def func(d,e,f):
    return d + e, f-e
```

In [26]:
```
func(5,6,7)
```

Out[26]:
```
(11, 1)
```

In [27]:
```
help(len)
```

```
Help on built-in function len in module builtins:

len(obj, /)
    Return the number of items in a container.
```

In [28]:
```
print(len.__doc__)
```

```
Return the number of items in a container.
```

In [29]:
```
##1. Write a function square_list(numbers) that takes a list of numbers and
#squares of those numbers using a list comprehension.
```

```
def square_list(numbers):
    return[num ** 2 for num in numbers]
```

In [30]:
```
nums = [1,2,3,4,5]
print(square_list(nums))
```

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

In [31]:
```
##2. Write a function get_employee_name(employee_dict, employee_id) that tak
#employees (where keys are IDs and values are names) and an employee ID, and
#of the employee with that ID.

def get_employee_name(employee_dict, employee_id):
    return employee_dict.get(employee_id)
```

In [32]:
```
employees = { # never use : this use =
    101:"Ram",
    102:"Hari",
    103:"Sita"
}
name = get_employee_name(employees, 101)
print(name)
```

```
Ram
```

In [33]:
```
##3. Write a function filter_high_scores(scores) that takes a dictionary of
#new dictionary with only the students who scored more than 80.
def filter_high_scores(scores):
    high_scores = {student: score for student, score in scores.items() if sc
    return high_scores
```

In [34]:
```
scores = {
    "Ram":88,
    "Hari":70,
    "Sita":90,
    }
print(filter_high_scores(scores))
```

```
{'Ram': 88, 'Sita': 90}
```

In [35]:
```
##4. Write a function common_elements(set1, set2) that takes two sets and re
def common_elements(set1, set2):
    return set1 & set2
```

In [36]:
```
set_a = {1,2,4,5}
set_b = {4,5,7,9}
print(common_elements(set_a,set_b))
```

```
{4, 5}
```

In [37]:
```
##5 5. Write a function lcm(a, b) that takes two integers and returns their
import math
def lcm(a,b):
    return abs(a * b) // math.gcd(a, b)
print(lcm(12,16))
print(lcm(40,22))
```

```
48
440
```

In [38]:
```
abs(-1)
```

Out[38]:
```
1
```

In [39]:
```
eval("print(1)") #most dangerous
```

```
1
```

In [40]:
```
sum([1,2,3,4,5])
```

Out[40]:
```
15
```

In [41]:
```
max([1,2,3,4,5])
```

Out[41]:
```
5
```

In [42]:
```
min([], default = 0)
```

Out[42]:
```
0
```

In [43]:
```
all([])
```

Out[43]:
```
True
```

In [44]:
```
any([])
```

Out[44]:
```
False
```

In [45]:
```
print(1,2,3,4,5,6,7,8,9)
```

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

In [46]:
```
def func(*args):
    print(args)
    print(type(args))
func(1,2,3,4,5,6,7,8,9)
```

```
(1, 2, 3, 4, 5, 6, 7, 8, 9)
<class 'tuple'>
```

In [48]:
```
a = "abc"
b = "def"
c = a+b
type(c), type(a), type(b),
```

Out[48]:
```
(str, str, str)
```

In [49]:
```
import math
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __add__(self, other):
        return Point(self.x + other.x, self.y + other.y)

    def __mul__(self, scalar):
        return Point(self.x * scalar, self.y * scalar)

    def __eq__(self, other):
        return self.x == other.x and self.y == other.y

    def distance_from_origin(self):
```

```
        return math.sqrt(self.x**2 + self.y**2)

    def __lt__(self, other):
        return self.distance_from_origin() < other.distance_from_origin()

    def __gt__(self, other):
        return self.distance_from_origin() > other.distance_from_origin()

    def __eq__(self, other):
        return self.distance_from_origin() == other.distance_from_origin()


p1 = Point(2,4)
p2 = Point(6,8)

p3 = p1 + p2
print("Addition:", p3.x, p3.y)

p4 = Point(2,4)
print("p1 == p4:", p1 == p4)

p5 = p2 * 2
print("Multiply:", p5.x, p5.y)

p1 = Point(1, 2)
p2 = Point(3, 4)

print(p1 < p2)
print(p1 > p2)
print(p1 == p2)
```

```
Addition: 8 12
p1 == p4: True
Multiply: 12 16
True
False
False
```

In [50]:
```
# hash value
print(hash(10))
print(hash("hello"))
#A hash value is a numeric representation of an object used to quickly
#identify and store it in hash-based data structures like sets and dictiona
```

```
10
-6285048362849347225
```

In [51]:
```
def func1():
    def func2():
        pass
    return func2

func1
```

Out[51]:
```
<function __main__.func1()>
```

In [52]:
```
##FUNCTION SCOPE
## for update, it can only access things that have been made available loca
```

In [53]:
```
x = 1
def func():
    x = 2
    def inner():
        print("inner",x)
```

```
    print("func1",x)
    inner()
    print("func1",x)
print(x)
func()
```

```
1
func1 2
inner 2
func1 2
```

In [54]:
```
##nonlocal : only needeed if you need to update values
# you can access the values directly
#del x : just to make sure that there is no global x
```

In [55]:
```
def func(n):
    def inner():
        return n ** 2
    return inner

v1 = func(10)
v2 = func(20)
v1 , v2
```

Out[55]:
```
(<function __main__.func.<locals>.inner()>,
 <function __main__.func.<locals>.inner()>)
```

In [56]:
```
v1(), v2() #has memory
```

Out[56]:
```
(100, 400)
```

In [60]:
```
# can we build counter based on this? build a closure that can be used for c
# everytime the function is called it should increase count by 1

def counter():

    my_counter = counter()
my_counter()  = print(1)
my_counter()  = print(2)

#everytime i call my_counter it should print the current count and inc count

def counter(initial):

    my_counter = counter(10)
my_counter()
my_counter()
```

```
  Cell In[60], line 7
    my_counter()  = print(1)
    ^
SyntaxError: cannot assign to function call here. Maybe you meant '==' inst
ead of '='?
```

In [59]:
```
def counter():
    return counter
my_counter = counter()
my_counter()
my_counter()
```

Out[59]:
```
<function __main__.counter()>
```

In [61]:
```
def counter(count): ## count as an argument to take values & it stays as mer

    def increase():
        nonlocal count
        count += 1
        print(count)

    return increase
## non local allows modifying count:c1 = counter(0)
f1 = counter(10)
f1()
f1()
f1()

f2 = counter(20)
f2()
f2()
f2()

#A closure is a function that remembers and uses variables
#from its outer function, even after the outer function has finished running



#notes:
    #Closure = function + remembered variables

#Inner function uses outer variables

#Variables stay alive after outer function exits

#increase() is the closure, not counter()
```

```
11
12
13
21
22
23
```

map,reduce,filter,lambda

In [62]:
```
a = [1,2,3,4,5,6]
b = [i**2 for i in a]
print(b)
```

```
[1, 4, 9, 16, 25, 36]
```

In [63]:
```
def square (x):
    return x*x
b = [square(i) for i in a]
print(b)
```

```
[1, 4, 9, 16, 25, 36]
```

In [64]:
```
for j in map(square,a):
    print(j)
```

```
1
4
9
16
25
36
```

In [65]:
```
list(map(square,a))
```

Out[65]:
```
[1, 4, 9, 16, 25, 36]
```

In [66]:
```
sum(map(square,a))
```

Out[66]:
```
91
```

In [67]:
```
elements = ["asdf","a","bdc","f","test"]
# write a map function to get len of each element in list
[4,1,3,1,4]
[len(x) for x in elements]
```

Out[67]:
```
[4, 1, 3, 1, 4]
```

In [68]:
```
list(map(len,elements))
```

Out[68]:
```
[4, 1, 3, 1, 4]
```

In [69]:
```
#in list comprehension
elements = [1,2,3,4,5,6,7,8,10,12]
even = [i for i in elements if not i % 2]
even
```

Out[69]:
```
[2, 4, 6, 8, 10, 12]
```

In [70]:
```
def is_even(x):
    return not x % 2
list(filter(is_even,elements))
```

Out[70]:
```
[2, 4, 6, 8, 10, 12]
```

In [71]:
```
list(filter(is_even,elements))
```

Out[71]:
```
[2, 4, 6, 8, 10, 12]
```

In [72]:
```
is_even = lamda x : not x % 2
is_even(3)
#this is not good practice due to memory management
#rule of thumb donot assign lamda func to a variable
```

```
  Cell In[72], line 1
    is_even = lamda x:not x % 2
                      ^
SyntaxError: invalid syntax
```

In [74]:
```
#wap that takes a list of dictionary as input
# each element of the list has name and age of the candidate
# based on this filter, make list of candidate eligible to vote

data = [
    {"name":"ram","age":88},
    {"name":"hari","age":17},
    {"name":"sita","age":20},
```

```
    {"name":"gita","age":16},
]

for i in range(0,4):
    if data[i]["age"] >= 18:
        print(data[i])
        print(list(data[i]))
```

```
{'name': 'ram', 'age': 88}
['name', 'age']
{'name': 'sita', 'age': 20}
['name', 'age']
```

In [75]:
```
from functools import reduce
```

In [76]:
```
a =[1,2,3,4]
def add_(x,y):
    print(x,y)
    return x + y
reduce(add_,a)
reduce(add_,[1])
```

```
1 2
3 3
6 4
```

Out[76]:
```
1
```

In [77]:
```
reduce(add_,[],0)
```

Out[77]:
```
0
```

In [78]:
```
reduce(add_,["test","test","test"])
```

```
test test
testtest test
```

Out[78]:
```
'testtesttest'
```

In [79]:
```
try:
    a = 10
    #b = a / 0

except Exception as e:
    print("I got error",e)

else:
    print("I got no error")
```

```
I got no error
```

In [80]:
```
try:
    a = 10
    b = a / 0
    df/=10
except Exception as e:
    print("I got error",e)
```

```
I got error division by zero
```

In [81]:
```
def func(div):
    try:
        print("IN try Block")
    a = 10
    b = a / div
```

```
        return "No error"
    except Exception as e:
        print("I got error",e)
        return "error"
    else:
        print("I got no error")
    finally:
        print("I will always run")

func(10)
```

```
  Cell In[81], line 4
    a = 10
    ^
SyntaxError: expected 'except' or 'finally' block
```

In [82]:
```
a = [1,2,3,4,5,6]
for i in a:
    print(i)
```

```
1
2
3
4
5
6
```

In [83]:
```
b = iter(a)
```

In [84]:
```
next(b)
```

Out[84]:
```
1
```

In [85]:
```
next(b)
```

Out[85]:
```
2
```

In [86]:
```
import re
```

In [88]:
```
match = re.search("test","this is test string 123")
```

In [89]:
```
dir(match)
```

Out[89]:
```
['__class__',
 '__class_getitem__',
 '__copy__',
 '__deepcopy__',
 '__delattr__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__getitem__',
 '__getstate__',
 '__gt__',
 '__hash__',
 '__init__',
 '__init_subclass__',
 '__le__',
 '__lt__',
 '__module__',
 '__ne__',
 '__new__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__setattr__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 'end',
 'endpos',
 'expand',
 'group',
 'groupdict',
 'groups',
 'lastgroup',
 'lastindex',
 'pos',
 're',
 'regs',
 'span',
 'start',
 'string']
```

In [90]:
```
match = re.search(r"\d+","this is test string 456 string 456")
match
```

Out[90]:
```
<re.Match object; span=(20, 23), match='456'>
```

In [91]:
```
re.findall(r"\d+","this is 3is test string 123 string 123")
```

Out[91]:
```
['3', '123', '123']
```

In [92]:
```
string = "Contact details = 202 555 1234"
#extract phone number from this string
```

In [93]:
```
re.findall(r"\d\d\d \d\d\d \d\d\d\d",string)
```

Out[93]:
```
['202 555 1234']
```

In [94]:
```
re.search(r"\d{3} \d{3} \d{4}", string)
```

Out[94]:
```
<re.Match object; span=(18, 30), match='202 555 1234'>
```

In [95]:
```
string1 = "Contact details = +1 202 555 1234"
string2 = "Contact details = +1 (202) 555 1234"
```

In [96]:
```
re.search(r"(\+1 )?\d{3} \d{3} \d{4}", string1)
```

Out[96]:
```
<re.Match object; span=(18, 33), match='+1 202 555 1234'>
```

In [97]:
```
import re
```

In [98]:
```
string2 = "Contact details = +1 (202) 555 1234"
```

In [99]:
```
re.search(r"(\+1 )?\(?d{3}\)? \d{3} \d{4}", string2)
```

In [100…
```
string = "Lina lina"
```

In [101…
```
re.findall("[Ll]ina",string)
```

Out[101]:
```
['Lina', 'lina']
```

In [102…
```
class Person:
    def __init__(self, name):
        self.name = name

    def get_name(self):
        return self.name
class Manager(Person):
    def __init__(self,name,position):
        self.position = position
        #option 1 :old method
        # Person.__init__(self,name)
        #better more pythonic in python3
        super().__init__(name)
    def info(self):
        return f"Name: {self.name}, Position: {self.position}"
```

In [103…
```
manager = Manager("Ram","Teller")
```

In [104…
```
manager.info()
```

Out[104]:
```
'Name: Ram, Position: Teller'
```

In [105…
```
manager.get_name()
```

Out[105]:
```
'Ram'
```

In [106…
```
#stack in a class
#push & pop
#list
#also queue visualization
```

In [107…
```
class stack:
    def __init__(self):
        self.item = []

    def push(self, item):
        self.item.append(item)
```

```
        print("pushed:",item)
        print("stack:",self.item)

    def pop(self, item):

            if not self.item:
                print("No elements to pop")
            else:
                removed = self.item.pop()
                print("popped:",item)
                print("Stack:", self.item)



s = stack()
s.push(10)
s.push(20)
s.push("no elements to pop")
s.pop(30)
s.push(50)
s.push(60)
s.pop(60)
```

```
pushed: 10
stack: [10]
pushed: 20
stack: [10, 20]
pushed: no elements to pop
stack: [10, 20, 'no elements to pop']
popped: 30
Stack: [10, 20]
pushed: 50
stack: [10, 20, 50]
pushed: 60
stack: [10, 20, 50, 60]
popped: 60
Stack: [10, 20, 50]
```

In [108…

```
##JOSEPHUS PROBLEM

class Circle:
    def __init__(self):
        self.person = []

    def add_persons(self, n):
        for i in range(1, n+1):
            self.person.append(i)

    def josephus(self, k):
        index = 0
        while len(self.person) > 1:
            index = (index + k - 1) % len(self.person)
            removed = self.person.pop(index)
            print("Removed:", removed, "Remaining:", self.person)

        print("Survivor:", self.person[0])


# Example run
c = Circle()
c.add_persons(7)   # 7 people
c.josephus(3)      # Every 3rd person eliminated
```

```
Removed: 3 Remaining: [1, 2, 4, 5, 6, 7]
Removed: 6 Remaining: [1, 2, 4, 5, 7]
Removed: 2 Remaining: [1, 4, 5, 7]
Removed: 7 Remaining: [1, 4, 5]
Removed: 5 Remaining: [1, 4]
Removed: 1 Remaining: [4]
Survivor: 4
```

In [109…

```python
##fifo queue
class Queue:
    def __init__(self):
        self.item = []

    def enqueue(self, item):
        self.item.append(item)
        print("Enqueued:", item)
        print("Queue:", self.item)

    def dequeue(self):
        if not self.item:
            print("No elements to dequeue")
        else:
            removed = self.item.pop(0)   # FIFO
            print("Dequeued:", removed)
            print("Queue:", self.item)


s = Queue()
s.enqueue(10)
s.enqueue(20)
s.enqueue("no elements to dequeue")
s.dequeue()
s.enqueue(50)
s.enqueue(60)
s.dequeue()
```

```
Enqueued: 10
Queue: [10]
Enqueued: 20
Queue: [10, 20]
Enqueued: no elements to dequeue
Queue: [10, 20, 'no elements to dequeue']
Dequeued: 10
Queue: [20, 'no elements to dequeue']
Enqueued: 50
Queue: [20, 'no elements to dequeue', 50]
Enqueued: 60
Queue: [20, 'no elements to dequeue', 50, 60]
Dequeued: 20
Queue: ['no elements to dequeue', 50, 60]
```

In [110…

```python
##parenthesis problem
def is_balanced(s):
    stack = []
    for ch in s:
        if ch in "({[":
            stack.append(ch)
        else:
            if not stack:
                return False
            top = stack.pop()
            if (top == '(' and ch != ')') or \
               (top == '{' and ch != '}') or \
               (top == '[' and ch != ']'):
```

```
                return False
    return len(stack) == 0

print(is_balanced("{[()]}"))
print(is_balanced("(]"))       #
```

```
True
False
```

In [111…

```
#stack in a class
#push & pop
#list
#also queue visualization
```

In [112…

```
class stack:
    def __init__(self):
        self.item = []

    def push(self, item):
        self.item.append(item)
        print("pushed:",item)
        print("stack:",self.item)

    def pop(self, item):

            if not self.item:
                print("No elements to pop")
            else:
                removed = self.item.pop()
                print("popped:",item)
                print("Stack:", self.item)



s = stack()
s.push(10)
s.push(20)
s.push("no elements to pop")
s.pop(30)
s.push(50)
s.push(60)
s.pop(60)
```

```
pushed: 10
stack: [10]
pushed: 20
stack: [10, 20]
pushed: no elements to pop
stack: [10, 20, 'no elements to pop']
popped: 30
Stack: [10, 20]
pushed: 50
stack: [10, 20, 50]
pushed: 60
stack: [10, 20, 50, 60]
popped: 60
Stack: [10, 20, 50]
```

In [113…

```
class stack:
    def __init__(self):
        self.stack = []

    def display_stack(self):
        print(self.stack)
```

```
    def pushstack(self,element):
        self.stack.append(element)

    def pop(self):
        if not self.stack:
            print("the stack is empty")
        else:
            return self,stack.pop()


s=stack()
s.push(happy)
s.push(100)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[113], line 19
     15             return self,stack.pop()
     18 s=stack()
---> 19 s.push(happy)
     20 s.push(100)

AttributeError: 'stack' object has no attribute 'push'
```

In [1]:
```
##JOSEPHUS PROBLEM

class Circle:
    def __init__(self):
        self.person = []

    def add_persons(self, n):
        for i in range(1, n+1):
            self.person.append(i)

    def josephus(self, k):
        index = 0
        while len(self.person) > 1:
            index = (index + k - 1) % len(self.person)
            removed = self.person.pop(index)
            print("Removed:", removed, "Remaining:", self.person)

        print("Survivor:", self.person[0])


# Example run
c = Circle()
c.add_persons(7)   # 7 people
c.josephus(3)      # Every 3rd person eliminated
```

```
Removed: 3 Remaining: [1, 2, 4, 5, 6, 7]
Removed: 6 Remaining: [1, 2, 4, 5, 7]
Removed: 2 Remaining: [1, 4, 5, 7]
Removed: 7 Remaining: [1, 4, 5]
Removed: 5 Remaining: [1, 4]
Removed: 1 Remaining: [4]
Survivor: 4
```

In [2]:
```
#fifo queue
class Queue:
    def __init__(self):
        self.item = []
```

```
    def enqueue(self, item):
        self.item.append(item)
        print("Enqueued:", item)
        print("Queue:", self.item)

    def dequeue(self):
        if not self.item:
            print("No elements to dequeue")
        else:
            removed = self.item.pop(0)   # FIFO
            print("Dequeued:", removed)
            print("Queue:", self.item)


s = Queue()
s.enqueue(10)
s.enqueue(20)
s.enqueue("no elements to dequeue")
s.dequeue()
s.enqueue(50)
s.enqueue(60)
s.dequeue()
```

```
Enqueued: 10
Queue: [10]
Enqueued: 20
Queue: [10, 20]
Enqueued: no elements to dequeue
Queue: [10, 20, 'no elements to dequeue']
Dequeued: 10
Queue: [20, 'no elements to dequeue']
Enqueued: 50
Queue: [20, 'no elements to dequeue', 50]
Enqueued: 60
Queue: [20, 'no elements to dequeue', 50, 60]
Dequeued: 20
Queue: ['no elements to dequeue', 50, 60]
```

In [3]:

```
def is_balanced(s):
    stack = []
    for ch in s:
        if ch in "({[":
            stack.append(ch)
        else:
            if not stack:
                return False
            top = stack.pop()
            if (top == '(' and ch != ')') or \
               (top == '{' and ch != '}') or \
               (top == '[' and ch != ']'):
                return False
    return len(stack) == 0

print(is_balanced("{[()]}"))
print(is_balanced("(]"))       #
```

```
True
False
```

Sequential Search

o(n) #for loop and check element

In [4]:
```
new_list = [25,26,36,47,40,94,67]
```

In [5]:
```
new_list.index(47)
```

Out[5]:
```
3
```

In [6]:
```
target = 40
for i,element in enumerate(new_list):
    if target == element:
        print("found element in position",i)
```

```
found element in position 4
```

In [7]:
```
sorted(new_list)
```

Out[7]:
```
[25, 26, 36, 40, 47, 67, 94]
```

In [9]:
```
# O(log(N))
# sorting is expensive -> O(log(n))
# Faster to search element

#why use binnary search?

#if repeated search is needed
```

In [11]:
```
# there is a 100 floor tall building. you have an egg and want to findout at
#floor the egg will break. the value for each batch of egg is different and

#- find the floor where egg breaks from sequential and binary search
#- which one is better why?
#- is the algorithm useless?
```

In [12]:
```
import random


break_floor = random.randint(1, 100)

def sequential_search():
    tries = 0
    for floor in range(1, 101):
        tries += 1
        if floor >= break_floor:
            print("Breaking floor:", floor)
            print("Attempts:", tries)
            return

sequential_search()
```

```
Breaking floor: 29
Attempts: 29
```

In [13]:
```
import random


break_floor = random.randint(1, 100)

def binary_search():
    low = 0
    high = 100
    tries = 0
```

In [14]:
```
import random

break_floor = random.randint(1, 100)

def binary_search():
    low = 1
    high = 100
    tries = 0

    while low <= high:
        mid = (low + high) // 2
        tries += 1

        if mid < break_floor:
            low = mid + 1
        else:
            high = mid - 1

    print("Breaking floor:", low)
    print("Attempts:", tries)

binary_search()
```

```
Breaking floor: 97
Attempts: 7
```

In [15]:
```
l, h = 0,10
search_ = 11
count = 0
values = list(range(0,111,10))

while True and count < 10:
    mid = (h+l)//2
    print(l,mid,h,breaking)
    if search_  == values[mid]:
        print("Breaks at :", mid)
        break
    elif search_  < values[mid]:
        h = mid
    elif search_  > values[mid]:
        l = mid + 1
    count += 1
if search_  != values[mid]:
    print("no value found")
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[15], line 8
      6 while True and count < 10:
      7     mid = (h+l)//2
----> 8     print(l,mid,h,breaking)
      9     if search_  == values[mid]:
     10         print("Breaks at :", mid)

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

In [16]:
```
#hash table
int(str(199)[-2:])
```

Out[16]:
```
99
```

In [17]:
```
def func(number):
    return number % 100
func(1000000010302050325)
```

Out[17]:
```
25
```

In [18]:
```
func(25)
```

Out[18]:
```
25
```

In [19]:
```
# "def" -> 0-100??
sum(map(ord, "def")) % 100
```

Out[19]:
```
3
```

In [20]:
```
sum_ = 0
string = "def"
for ch in string:
    sum_+= ord(ch)
sum_ % 100
```

Out[20]:
```
3
```

In [21]:
```
sum_ = 0
string = "def"
sum_ = len(string)
sum_ % 100
```

Out[21]:
```
3
```

In [22]:
```
def hash_(string):
    sum_ = 0

    for ch in string:
        sum_ += ord(ch)
        return (sum_ ) % 100



# O(M).. if M<<<N; then O(M)- O(1)
```

In [23]:
```
my_list = ["a","b","c","10","Ram","Hari","hari"]
```

In [24]:
```
list(map(hash_,my_list))
```

Out[24]:
```
[97, 98, 99, 49, 82, 72, 4]
```

In [25]:
```
k = 10
hash_table = []
for i in range(k):
    hash_table.append([])
```

In [26]:
```
hash_table
```

Out[26]:
```
[[], [], [], [], [], [], [], [], [], []]
```

In [27]:
```
my_list
```

Out[27]:
```
['a', 'b', 'c', '10', 'Ram', 'Hari', 'hari']
```

In [28]:
```
list(map(hash_,my_list))
```

Out[28]:
```
[97, 98, 99, 49, 82, 72, 4]
```

In [30]:
```
## create a custom hash data structure that stores element.implement functi

class HashTable:
    def __init__(self):

        self.size = 10
        self.table = [None] * self.size


    def hash_function(self, key):
        return hash(key) % self.size


    def put(self, key, value):
        index = self.hash_function(key)
        self.table[index] = (key, value)
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: