python 3 basics

February 21, 2025

1 string methods

below are are methods not modifiers, it returns a modified string but does not modify the original so need assign

[1]:
```
s = ""
l = ['a','b','c','d']
s = " ".join(l)
print(s)
s = s.replace(" ","")
print(s)
l = s.split("_")
print(l)
```

```
a b c d
abcd
['abcd']
```

#to convert between strings and lists

[2]:
```
print(list("abc"))
print("abc".split())
```

```
['a', 'b', 'c']
['abc']
```

[3]:
```
s = "              abcd"
s = s.upper()
s = s.strip()
print(s)
```

```
ABCD
```

[4]:
```
s = "atatatata"
print(s.isdigit())
s = s.replace("a","A")
print(s)
print(s.find("A"))
print(s.count("At"))
```

```
False
AtAtAtAtA
0
4
```

[5]:
```
s = "012345678"
print(s.isalpha())
print(s[::2])
print(s[::-1])
print(s[-3:])
```

```
False
02468
876543210
678
```

2 files

[6]:
```
s="""hello everyone,
My name is ultrasonicfats.
I am a year x undergraduate studying at Imperial College London.
I take the 4 year MEng Biomedical Engineering course.
Nice to meet you!"""
f_name="revision.txt"
def read(f):
    f = open("revision.txt","r")
    now = f.read()
    print("contents:\n",now.strip(),"\nword count:",wordcount(f),"\n")
    f.close()
def writeapp(f,c,md):
    f = open("revision.txt",md)
    f.write(c)
    f.close()
def wordcount(f):
    f = open("revision.txt","r")
    line=f.readline()
    sum=0
    while line!='':
        sum+=len(line.split(" "))
        line=f.readline()
    f.close()
    return sum
writeapp(f_name,s,"w")
read(f_name)
writeapp(f_name,s,"a")
read(f_name)
writeapp(f_name,"","w")
```

```
read(f_name)
```

```
contents:
 hello everyone,
My name is ultrasonicfats.
I am a year x undergraduate studying at Imperial College London.
I take the 4 year MEng Biomedical Engineering course.
Nice to meet you!
word count: 30

contents:
 hello everyone,
My name is ultrasonicfats.
I am a year x undergraduate studying at Imperial College London.
I take the 4 year MEng Biomedical Engineering course.
Nice to meet you!hello everyone,
My name is ultrasonicfats.
I am a year x undergraduate studying at Imperial College London.
I take the 4 year MEng Biomedical Engineering course.
Nice to meet you!
word count: 59

contents:

word count: 0
```

[7]:
```
# using with statement
with open('file_path', 'w') as file:
    file.write('hello world !')
```

Notice that unlike the first two implementations, there is no need to call file.close() when using with statement

csv

[ ]:
```

```

3 Random Libary

[8]:
```
import random
# does not include upper bound in this case 10
x = random.randrange(1,10)
print(x)
# includes upper bound
x = random.randint(1,10)
print(x)
l = ["gamma","alpha","beta","delta"]
```

```
x = random.choice(l)
print(x)
# note this a modifer it does not return anything, it modifies directly
random.shuffle(l)
print(l)
# range does not include upperbound just like randrange
l2 = list(range(5,20,2))
random.shuffle(l2)
print(l2)
#ascending
# note that "sort()" is a modifier while "sorted()" is the method that returns␣
 ↪the sorted list
print(sorted(l2))
# descending
print(sorted(l2,reverse=True))
```

```
2
9
delta
['beta', 'gamma', 'alpha', 'delta']
[17, 15, 5, 13, 7, 19, 11, 9]
[5, 7, 9, 11, 13, 15, 17, 19]
[19, 17, 15, 13, 11, 9, 7, 5]
```

sometimes you may wanna use randrange instead as it allows you to specify step

[9]:
```
import random
# using list comprehension to initialize list
a = [i for i in range(1, 6)]
b = [random.randrange(1, 50, 3) for i in range(7)]
print(a)
print(b)
```

```
[1, 2, 3, 4, 5]
[43, 37, 19, 1, 49, 16, 40]
```

4 Math library

[10]:
```
import math
print(math.cos(math.pi))
print(math.factorial(4))
# <=5 roudn down. second arg is number of dp to round to
print(round(5.4523,2))
print(round(5.4523,1))
v = ["hello",88,{"a":100,'b':600},(1,"a"),[4,"b"]]
for i in v:
    print(type(i),end=" ")
```

```
-1.0
24
5.45
5.5
<class 'str'> <class 'int'> <class 'dict'> <class 'tuple'> <class 'list'>
```

5 Lists and Dictionaries

[11]:
```
import random
grades = {90:"A",80:"B",70:"C"}
print(list(grades.keys()))
print(list(grades.values()))
#used to insert or read value from dict
grades[60]="D"
l=list(grades.items())
random.shuffle(l)
lr=sorted(l,key=lambda x:x[0])
print(l)
print(lr)
grades.pop(90)
print(grades)
grades.clear()
print("contents: ",grades)
```

```
[90, 80, 70]
['A', 'B', 'C']
[(80, 'B'), (70, 'C'), (90, 'A'), (60, 'D')]
[(60, 'D'), (70, 'C'), (80, 'B'), (90, 'A')]
{80: 'B', 70: 'C', 60: 'D'}
contents:  {}
```

[12]:
```
# watch out for aliasing effect of lists
a = [10,20,30]
b = a
a[1]=99
print("whoops aliasing")
print(a)
print(b)
# how to tell aliasing
print(a is b,": thus a points to b")
# to prevent this use a copy instead
a = [10,20,30]
b = a[::1]
a[1]=99
print("no aliasing")
print(a)
print(b)
```

```
whoops aliasing
[10, 99, 30]
[10, 99, 30]
True : thus a points to b
no aliasing
[10, 99, 30]
[10, 20, 30]
```

[13]:
```python
# the following are MODIFIERS
a = [10,20,30]
b = [60,70,80]
a.append(40)
a.extend(b)
print(a)
#if no argument in pop the index is assumed to be last element
#RETURNS popped element
a.pop(len(a)//2)
print(a)
#note that // is FLOOR devision while / is float division
# so if used len(a)/2 in pop above will run error
```

```
[10, 20, 30, 40, 60, 70, 80]
[10, 20, 30, 60, 70, 80]
```

Initializing

[14]:
```python
import random
# using list comprehension to initialize list
a = [i for i in range(1, 6)]
b = [random.randrange(1, 50, 1) for i in range(7)]
print(a)
print(b)
```

```
[1, 2, 3, 4, 5]
[7, 3, 31, 36, 4, 13, 34]
```

6 Control Statements

[15]:
```python
grades = {90:"A",80:"B",70:"C"}
for i in grades.items():
    #ascii num to char
    print(chr(i[0]))
    #char to ascii num
    s = "\n"+"_"*10+"\n"
    print(ord(i[1]),end=s)
```

```
Z
65

----------
```

```
P
66

----------
F
67

----------
```

[16]:
```
def example(required,option1=2,option2=3):
    print(required,option1,option2)
example(1)
example(1,4) #override second
example(1,4,5) #override second and third
example(1,option2=5,option1=4) #override by keyword
```

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

[17]:
```
#nth fibonacci number eg. 1 1 2 3 5 8 13
def fib(n):
    if n<3:
        return 1
    else:
        return fib(n-1)+fib(n-2)
res = fib(5)
print(res)
```

```
5
```

7 Operators

[18]:
```
# None=Null
print(None==None)
# NaN(not a number) as a float not string!!
print(float('NaN') == float('NaN'))
print(float('NaN') is float('NaN'))
#NaN as a string
print('NaN'=='NaN')
```

```
True
False
False
True
```

[19]:
```
# why NaN, basically something that cant be compared(no binary number rep?), so␣
  ↪if it equals itself just gets false
def isNaN(num):
```

```
    print(num != num)
isNaN(5)
isNaN("hello")
isNaN(float('NaN'))
```

```
False
False
True
```

[20]:
```
# binary operators(note that arithmetic operators ur pamdas takes precendence)
#<< left shift(multiply by 2)
print(2<<3)
#>> right shift(divide by 2)
print(12>>2)
#^ bitwise XOR

#"this is not the way to do 2 complement,in logic gates they did this bit by␣
 ↪bit!"
print((-1)^5+1)
print(bin(-1)[2:],"^",bin(5)[2:],"+",bin(1)[2:],"=",bin((-1)^5)[2:])

#| bitwise OR
#~ bitwise complement
print(~5+1)
# logical operators
# OR, AND

#membership and identity operators
# in,not in
# is,is not
```

```
16
3
-7
b1 ^ 101 + 1 = b110
-5
```

[21]:
```
# fast way to convert to binary etc
h=hex(5)
b=bin(5)
o=oct(5)
l=[h,b,o]
for i in l:
    print(i,end=" ")
print("\n")
for i in l:
    print(i[2:],end=" ")
print("\n")
```

```
print(int(h,16))
print(int(b,2))
print(int(o,8))
```

```
0x5 0b101 0o5

5 101 5

5
5
5
```

[22]:
```
print("hello\b")
while True:
    print(1)
    break
print("continue")
for j in range(5):
    if j==2:
        continue
    else:
        print(j)
print("break")
for j in range(5):
    if j==2:
        break
    else:
        print(j)
```

```
hell
1
continue
0
1
3
4
break
0
1
```

8 OOP

[23]:
```
class Automobile:
# The __init__ method accepts arguments for the make, model,
#mileage, and price.It initializes the data attributes with
#these values.
    def __init__(self, make, model, mileage, price):
```

```
        self._make = make
        self._model = model
        self._mileage = mileage
        self._price = price
    # The following methods are mutators for the class's data
    #attributes.
    # The following methods are the accessors for the class's data
    # attriburtes
    def set_make(self, make):
        self._
        make = make
    def set_model(self, model):
        self._model = model
    def set_mileage(self, mileage):
        self._mileage = mileage
    def set_price(self, price):
        self._price = price
    def get_make(self):
        return self._make
    def get_model(self):
        return self._model
    def get_mileage(self):
        return self._mileage
    def get_price(self):
        return self._price
```

inheritance

[24]:
```
class Car(Automobile):
    # The __init__ method accepts arguments for the car's make,
    # model, mileage,price, and doors.
    def __init__(self, make, model, mileage, price, doors):
        # Call the superclass's __init__ method and pass the
        #required arguments.
        super().__init__(make, model, mileage, price)
        # Initialize the __doors attribute.
        self._doors = doors
        # The set_doors method is the mutator for the __doors
        #attribute.
    def set_doors(self, doors):
        self._doors = doors
    # The get_doors method is the accessor for the __doors
    # attribute.
    def get_doors(self):
        return self._doors
```

polymorphism

[25]:
```
# The Mammal class represents a generic mammal.
class Mammal:
    # The __init__ method accepts an argument for the mammal's
    #species.
    def __init__(self,species):
        self._species = species
        # The show_species method displays a message indicating the
        # mammal's species.
    def show_species(self):
        print ('I am a', self.species)
        # The make_sound method is the mammal's way of making a
        # generic sound.
    def make_sound(self):
        print ('Grrrrr')
    # The Dog class is a subclass of the Mammal class.
class Dog(Mammal):
# The __init__ method calls the superclass's __init__ method
#passing 'Dog' as the species.
    def __init__(self):
        super().__init__('Dog')
        # The make_sound method overrides the superclass'smake_sound
        #method.
    def make_sound(self):
        print ('Woof! Woof!')
        # The Cat class is a subclass of the Mammal class.
class Cat(Mammal):
    # The __init__ method calls the superclass's __init__method
    #passing 'Cat' as thespecies.
    def __init__(self):
        super().__init__('Cat')
        # The make_sound method overrides the superclass'smake_sound
        #method.
    def make_sound(self):
        print ('Meow')
```

8.1 magic methods

getitem allows indexing of object

[26]:
```
class MyList:
   def __init__(self, data):
      self.data = data

   def __getitem__(self, index):
      return list(self.data[index])

my_list = MyList([1, 2, 3, 4, 5])
```

```
print(my_list[1:3])
```

```
[2, 3]
```

[27]:
```
class myObj:
  name = "John"

y = myObj()

x = isinstance(y, myObj)

print(x)
```

```
True
```

[28]:
```
class Person:
  name = "John"
  age = 36
  country = "Norway"

for attr in vars(Person):
        print(attr, getattr(Person, attr))
x = hasattr(Person, 'age')
print(x)
```

```
__module__ __main__
name John
age 36
country Norway
__dict__ {'__module__': '__main__', 'name': 'John', 'age': 36, 'country':
'Norway', '__dict__': <attribute '__dict__' of 'Person' objects>, '__weakref__':
<attribute '__weakref__' of 'Person' objects>, '__doc__': None}
__weakref__ <attribute '__weakref__' of 'Person' objects>
__doc__ None
True
```

deepcopy

[29]:
```
import copy

class DNA:
    def __init__(self, sequence, mutations=None):
        self.sequence = sequence
        self.mutations = mutations if mutations is not None else []  # A␣
 ↪mutable list

    def mutate(self):
        dna_new = copy.deepcopy(self)  # Deep copy of the object
```

```
        dna_new.sequence = dna_new.sequence.replace("A", "T")  # Modify the␣
 ↪sequence
        dna_new.mutations.append("A->T")  # Modify the mutations list
        return dna_new

dna = DNA("AGCT")
mutated_dna = dna.mutate()

print("Original DNA:")
print("Sequence:", dna.sequence)          # Outputs: AGCT
print("Mutations:", dna.mutations)        # Outputs: []

print("\nMutated DNA:")
print("Sequence:", mutated_dna.sequence) # Outputs: TGCT
print("Mutations:", mutated_dna.mutations) # Outputs: ['A->T']
```

```
Original DNA:
Sequence: AGCT
Mutations: []

Mutated DNA:
Sequence: TGCT
Mutations: ['A->T']
```

shallowcopy

[30]:
```
import copy

class DNA:
    def __init__(self, sequence, mutations=None):
        self.sequence = sequence
        self.mutations = mutations if mutations is not None else []  # A␣
 ↪mutable list

    def mutate(self):
        dna_new = copy.copy(self)  # Shallow copy of the object
        dna_new.sequence = dna_new.sequence.replace("A", "T")  # Modify the␣
 ↪sequence
        dna_new.mutations.append("A->T")  # Modify the mutations list
        return dna_new

dna = DNA("AGCT")
mutated_dna = dna.mutate()

print("Original DNA:")
print("Sequence:", dna.sequence)          # Outputs: AGCT
print("Mutations:", dna.mutations)        # Outputs: ['A->T']
```

```
print("\nMutated DNA:")
print("Sequence:", mutated_dna.sequence) # Outputs: TGCT
print("Mutations:", mutated_dna.mutations) # Outputs: ['A->T']
```

```
Original DNA:
Sequence: AGCT
Mutations: ['A->T']

Mutated DNA:
Sequence: TGCT
Mutations: ['A->T']
```

to see why aliasing happened for the second case consider

[31]:
```
import copy
li1 = [1, 2, [3, 5], 4]
li2 = copy.copy(li1)
print("li2 ID: ", id(li2), "Value: ", li2)
li3 = copy.deepcopy(li1)
print("li3 ID: ", id(li3), "Value: ", li3)
```

```
li2 ID:  2649520250112 Value:  [1, 2, [3, 5], 4]
li3 ID:  2649520377088 Value:  [1, 2, [3, 5], 4]
```

9 Error handling

[32]:
```
#The try block will generate an error, because unknownvariable is not defined:

try:
  print(unknownvariable)
except:
  print("An exception occurred")
```

```
An exception occurred
```

10 Time

[33]:
```
import time


print('time test')


t1 = time.time()

data = []
for i in range(0,1000):
        for j in range(0,1000):
```

```
                data.append(i)

t2 = time.time()

print(t2 - t1)
```

```
time test
0.06372189521789551
0.06372189521789551
```

11 Sorting

Insertion Sort

[34]:
```
def insertion_sort(A): # Insertion sort array A
    for i in range(1, len(A)): # O(n) loop over array
        j = i # O(1) initialize pointer
        while j > 0 and A[j] < A[j - 1]: # O(i) loop over prefix
            A[j - 1], A[j] = A[j], A[j - 1] # O(1) swap
            j = j - 1 # O(1) decrement j
```

Selection Sort

[35]:
```
def selection_sort(A): # Selection sort array A
    for i in range(len(A) - 1, 0, -1): # O(n) loop over array
        m = i # O(1) initial index of max
        for j in range(i): # O(i) search for max in A[:i]
            if A[m] < A[j]: # O(1) check for larger value
                m = j # O(1) new max found
        A[m], A[i] = A[i], A[m] # O(1) swap
```

12 Decorators(OOP)

[36]:
```
def simple_decorator(func):
    def wrapper():
        print("Before the function call")
        func()
        print("After the function call")
    return wrapper

@simple_decorator
def say_hello():
    print("Hello!")

say_hello()
```

```
Before the function call
Hello!
```

```
After the function call
```

[37]:
```python
def debug(func):
    def wrapper(*args, **kwargs):
        print(f"Calling {func.__name__} with {args} and {kwargs}")

        result = func(*args, **kwargs)
        print(f"{func.__name__} returned {result}")
        return result
    return wrapper

@debug
def add(x, y,z):
    l = [item for item in z.items()]
    print(x + y + l[0][0] + l[0][1])

add(5, 10,{2:3})
# Outputs:
# Calling add with (5, 10) and {}
# add returned 15
```

```
Calling add with (5, 10, {2: 3}) and {}
20
add returned None
```

[38]:
```python
car = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}

x = car.items()

item_list = [item for item in x]
print(item_list[0][0])
```

```
brand
```

[39]:
```python
class Circle:
    def __init__(self, radius, weed):
        self._radius = radius
        self.weed= weed

    @property
    def radius(self):
        return self._radius

    @radius.setter
```

```
    def radius(self, value):
        if value < 0:
            raise ValueError("Radius cannot be negative")
        self._radius = value

    @property
    def area(self):
        return 3.1416 * self._radius ** 2

circle = Circle(10,5)
print(circle.weed)
circle.weed=6
print(circle.weed)
```

```
5
6
```

13 Classmethods(OOP)

[40]:
```
class Geeks:
    course = 'DSA'
    list_of_instances = []

    def __init__(self, name):
        self.name = name
        Geeks.list_of_instances.append(self)

    @classmethod
    def get_course(cls):
        return f"Course: {cls.course}"

    @classmethod
    def get_instance_count(cls):
        return f"Number of instances: {len(cls.list_of_instances)}"

    @staticmethod
    def welcome_message():
        return f"Welcome to geeks for Geeks!"

# Creating instances
g1 = Geeks('Alice')
g2 = Geeks('Bob')

# Calling class methods
print(Geeks.get_course())
print(Geeks.get_instance_count())
```

```
# Calling static method
print(Geeks.welcome_message())
```

```
Course: DSA
Number of instances: 2
Welcome to geeks for Geeks!
```

docstring class

[41]:
```
import math

class Point:
    '''Represents a point in two-dimensional Euclidean space

    attributes:
        x: distance along one axis
        y: distance along the other perpendicular axis
    '''

    def __init__(self, x=0.0, y=0.0):
        self.x = x
        self.y = y

    def __str__(self):
        return f'({self.x},{self.y})'

    def polar(self):
        theta = math.atan(self.y/self.x)/math.pi*180
        r = math.sqrt(self.x**2 + self.y**2)
        return r, theta

    def __add__(self, point):
        total = Point()
        if isinstance(point, Point):
            total.x = self.x + point.x
            total.y = self.y + point.y
        else:
            total.x = self.x + float(point)
            total.y = self.y + float(point)
        return total

    def __radd__(self, point):
        total = Point()
        if isinstance(point, Point):
            total.x = self.x + point.x
            total.y = self.y + point.y
        else:
            total.x = self.x + float(point)
```

```
            total.y = self.y + float(point)
        return total

    def __sub__(self, point):
        total = Point()
        if isinstance(point, Point):
            total.x = self.x - point.x
            total.y = self.y - point.y
        else:
            total.x = self.x - float(point)
            total.y = self.y - float(point)
        return total

    def __rsub__(self, point):
        total = Point()
        if isinstance(point, Point):
            total.x = point.x - self.x
            total.y = point.y - self.y
        else:
            total.x = float(point) - self.x
            total.y = float(point) - self.y
        return total

    def __mul__(self, factor):
        total = Point()
        total.x = self.x*factor
        total.y = self.y*factor
        return total

    def __rmul__(self, factor):
        total = Point()
        total.x = self.x*factor
        total.y = self.y*factor
        return total

def distance(point1, point2):
    vector = point2-point1
    return math.sqrt(vector.x**2 + vector.y**2)

def print_attributes(obj):
    for attr in vars(obj):
        print(attr, getattr(obj, attr))
```

see that the doc string is automatically extracted when help is called

[42]:
```
A=['a','b','c']
B=['e','f','g']
```

```
A.extend(B)
print(A)
```

```
['a', 'b', 'c', 'e', 'f', 'g']
```