Advanced Python: Functional Programming

Pure Functions • Map: • Filter: • Zip: • Reduce: Lambda Expressions List Comprehensions Set Comprehensions Dictionary Comprehensions Decorators • Higher Order Function: • Decorators - With Parameters Error Handling • Exceptions usage • Raising Exceptions Generators • Creating our own For Loop • Creating our own Range Function Modules Packages __name__ Built-in Modules • random • sys Python Package index Virtual Environments Debugging File I/O • Read, write, append • File IO Errors • Translator Regular Expressions • The search() function • The split() Function • The sub() Function • Match Object • Password Validation Testing in Python

Pure Functions:

Pure function has two rules:

1. given the same input, it will always return the same output
2. The idea of a function should not produce any side effects.

"For example, if I was to print something inside of this function, it affects the outside world, right? Because I'm printing something onto a screen.The screen is the outside world"

In [1]:
```
def mulby2(lst):
    newlst=[]
    for i in lst:
        newlst.append(i*2)
    return newlst

mulby2([1,2,3])
```

Out[1]:
```
[2, 4, 6]
```

Map:

In [4]:
```
def mulby2(lst):
    newlst=[]
    for i in lst:
        newlst.append(i*2)
    return newlst
print(map(mulby2, [1,2,3]))
```

```
<map object at 0x00000190CE5CFB20>
```

In [1]:
```
def mulby2(lst):
    newlst=[]
    for i in lst:
        newlst.append(i*2)
    return newlst
print(list(map(mulby2, [1,2,3])))
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[1], line 6
      4         newlst.append(i*2)
      5     return newlst
----> 6 print(list(map(mulby2, [1,2,3])))

Cell In[1], line 3, in mulby2(lst)
      1 def mulby2(lst):
      2     newlst=[]
----> 3     for i in lst:
      4         newlst.append(i*2)
      5     return newlst

TypeError: 'int' object is not iterable
```

In [7]:
```
def mulby2(i):
    return i*2
print(list(map(mulby2, [1,2,3])))
```

```
[2, 4, 6]
```

The `map()` function in Python is a built-in function that allows you to apply a function to all the elements in an input list.

```
def mulby2(i):
    return i*2
print(list(map(mulby2, [1,2,3])))
```

is similar to:

```
def mulby2(i):
    return i*2
mulby2(1)

def mulby2(i):
    return i*2
```

```
mulby2(2)

def mulby2(i):
    return i*2
mulby2(33)
```

In [1]:
```
b= list(map(int,input().strip().split()))
print("\nList is - ", b)
print(b[2])
```

```
5 7 9

List is -  [5, 7, 9]
9
```

In [2]:
```
b= list(map(int,input().split()))
print("\nList is - ", b)
```

```
5 4 8

List is -  [5, 4, 8]
```

In [1]:
```
# Reads two numbers from input and typecasts them to int using
# map function
x, y = map(int, input('Enter Numbers: ').split())
print(x,y)
```

```
Enter Numbers: 2 5
2 5
```

- The `map()` function in Python is a built-in function that allows you to apply a function to all the elements in an input list.

The function is defined as follows:

```
map(function, iterable)
```

Here, `function` is the function you want to apply to each element of the iterable (which can be a list, a tuple, etc.), and `iterable` is the input sequence.

The `map()` function returns a map object, which is an iterator. You can convert this map object to a list by using the `list()` function if you need to.

Here's an example:

```
# Define a function that squares a number
def square(x):
    return x * x

# Create a list of numbers
numbers = [1, 2, 3, 4, 5]

# Use map() to apply the square function to each element in the list
squared_numbers = map(square, numbers)

# Convert the result to a list and print it
print(list(squared_numbers))
```

This will output:

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

In this example, the `square()` function is applied to each element in the `numbers` list, and the result is a new list where each element is the square of the corresponding element in the original list.

In [1]:
```
def square(x):
    return x * x
numbers = [1, 2, 3, 4, 5]
squared_numbers = map(square, numbers)
print(squared_numbers)
```

```
<map object at 0x0000021BC96D6260>
```

In [2]:
```
def square(x):
    return x * x
numbers = [1, 2, 3, 4, 5]
squared_numbers = map(square, numbers)
for i in squared_numbers:
    print(i)
```

```
1
4
9
16
25
```

In [1]:
```
def square(x):
    return x * x
numbers = [1, 2, 3, 4, 5]
```

```
squared_numbers = map(square, numbers)
print(list(squared_numbers))
```

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

Filter:

> The `map()` function allows you to apply a function to all the elements in an input list.

> The `filter()` function filters elements from an iterable (like list, tuple, etc) if they satisfy a specific condition.
>
> > - Filters all the values of True and gives as o/p

In [3]:
```
def odd(i):
    return i%2!=0
print(odd(4))
print(odd(9))
```

```
False
True
```

In [7]:
```
def only_odd(i):
    return i%2 != 0  #Returns True or False

result= only_odd(4)
print(type(result)) #returns bool type - True or False

print((filter(only_odd, [1,2,3,5,6,8,7,9])))

print(type(filter(only_odd, [1,2,3,5,6,8,7,9])))

print(list((filter(only_odd, [1,2,3,5,6,8,7,9]))))
```

```
<class 'bool'>
<filter object at 0x0000019F8B343D30>
<class 'filter'>
[1, 3, 5, 7, 9]
```

`filter()` in Python is a built-in function that filters elements from an iterable (like list, tuple, etc) if they satisfy a specific condition.

- The condition is defined in a function (called filter function).
- It returns an iterator of the same type as the input iterable. We can convert this iterator to a list or tuple using the `list()` or `tuple()` functions respectively.

Here's a basic syntax of the `filter()` function:

```
filter(function, iterable)
```

Where:

- "function" is the filter function that defines the filtering condition.
- "iterable" is the iterable (like list, tuple, etc.) you want to filter.

Here's an example:

Suppose we have a list of numbers and we want to filter out only the even numbers.

```
# Define the filter function
def is_even(n):
    return n % 2 == 0

# Define the list of numbers
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

# Use filter()
even_numbers = filter(is_even, numbers)

# Convert the result to a list and print it
print(list(even_numbers))
```

This will output: `[2, 4, 6, 8, 10]`.

In this example, `is_even` is the filter function that checks if a number is even. `filter(is_even, numbers)` returns an iterator that produces the elements of the `numbers` list that satisfy the condition (are even). The `list()` function is then used to convert this iterator to a list.

In [29]:
```
def only_odd(i):
    return i%2 != 0  #Returns True or False
print(list((filter(only_odd, [1,2,3,5,6,8,7,9]))))


#Filters all the values of True and gives as o/p
```

```
[1, 3, 5, 7, 9]
```

The return type of the filter() function in Python is actually a Filter object, which is an iterable. However, if you convert it to a list or iterate over it, the elements are of the same type as those in the original iterable.

Here's an example:

```
def is_even(x):
    return x % 2 == 0`

result = filter(is_even, [1, 2, 3, 4, 5])

# If printed result directly, it shows <filter object at 0x7f9c9d8ad990>

# To access the elements and check the return type, we can convert it to a list:
result_list = list(result)
print(result_list) # Output: [2, 4]

# Or iterate over it:
for num in result:
    print(num)
    # Output: 2
    #         4
```

As you can see, the elements in the result are of the same type as those in the original iterable (in this case, integers), not Filter objects.

Zip:

In [30]:
```
lst1=[1,2,3]
lst2=[10,20,30]
print(list((zip(lst1,lst2))))
```

```
[(1, 10), (2, 20), (3, 30)]
```

In [8]:
```
lst1=[1,2,3]
lst2=[10,20]
print(list((zip(lst1,lst2))))
```

```
[(1, 10), (2, 20)]
```

In [9]:
```
lst1=[1,2]
lst2=[10,20,70]
print(list((zip(lst1,lst2))))
```

```
[(1, 10), (2, 20)]
```

The `zip()` function in Python is a built-in function that allows you to iterate over two or more lists (or other iterable objects) in parallel, aggregating their elements into tuples. It's often used for merging lists or other data structures.

Here's a simple example:

```
names = ['Alice', 'Bob', 'Charlie']
ages = [25, 30, 35]

for name, age in zip(names, ages):
    print(f'{name} is {age} years old.')
```

This would output:

```
Alice is 25 years old.
Bob is 30 years old.
Charlie is 35 years old.
```

In this example, `zip(names, ages)` returns an iterator of tuples, where the i-th tuple contains the i-th element from each of the argument sequences or iterables.

- The iterator stops when the shortest input iterable is exhausted.

You can also use `zip()` with three or more arguments:

```
names = ['Alice', 'Bob', 'Charlie']
ages = [25, 30, 35]
cities = ['New York', 'San Francisco', 'Los Angeles']

for name, age, city in zip(names, ages, cities):
    print(f'{name} is {age} years old and lives in {city}.')
```

This would output:

```
Alice is 25 years old and lives in New York.
Bob is 30 years old and lives in San Francisco.
Charlie is 35 years old and lives in Los Angeles.
```

Note: If the iterables are of unequal length, `zip()` truncates the output to the length of the shortest iterable. To avoid this and fill in missing values with a default value, you can use the `itertools.zip_longest()` function.

Reduce:

https://towardsdatascience.com/using-reduce-in-python-a9c2f0dede54

https://thepythonguru.com/python-builtin-functions/reduce/

https://www.geeksforgeeks.org/reduce-in-python/

`reduce()` is a higher-order function in Python, which means it takes in other functions as arguments, or returns a function as its result. The `reduce()` function in the `functools` module is used for performing some computation on a list and returning the result, but it is not limited to lists - it works on any iterable.

The `reduce()` function takes in two required arguments:

1. A function that will be used to combine the items in the iterable.
2. An iterable (like list, tuple etc.).

It also takes in an optional third argument: a starting value. If this third argument is supplied, it is used as the initial value in the computation. If it's not supplied, the first item in the iterable is used as the initial value.

Here's an example of `reduce()` with a simple function that adds numbers:

```
from functools import reduce

def add(x, y):
    return x + y

numbers = [1, 2, 3, 4, 5]
print(reduce(add, numbers)) # Output: 15
```

Here, `reduce()` uses the `add()` function to progressively combine the numbers in the list, effectively summing them up.

Let's see another example using strings:

```
from functools import reduce

def concatenate(x, y):
    return x + y

strings = ['Hello', 'World']
print(reduce(concatenate, strings)) # Output: 'HelloWorld'
```

In this example, `reduce()` uses the `concatenate()` function to progressively combine the strings in the list, effectively concatenating them.

Note: The `reduce()` function is not a built-in function in Python 3+, so you need to import it from the `functools` module. In Python 2, `reduce()` is a built-in function.

In [34]:
```
from functools import reduce
lst1=[1,2,3]
def accumulator(acc,i): #bydefault 'acc' is the 1st value of iterable
    print (acc,i)
    return acc+i # this value will become 'acc' now
print(reduce(accumulator,lst1))
```

```
1 2
3 3
6
```

In [35]:
```
from functools import reduce
lst1=[1,2,3]
def accumulator(acc,i):
    print (acc,i)
    return acc+i # this value will become 'acc' now
print(reduce(accumulator,lst1,10))  #Here 10 is 'acc'.
```

```
10 1
11 2
13 3
16
```

The reduce() function accepts a function and a sequence and returns a single value calculated as follows:

Initially, the function is called with the first two items from the sequence and the result is returned.

The function is then called again with the result obtained in step 1 and the next value in the sequence. This process keeps repeating until there are items in the sequence.

The syntax of the reduce() function is as follows:

Syntax: reduce(function, sequence[, initial]) -> value

When the initial value is provided, the function is called with the initial value and the first item from the sequence.

In Python 2, reduce() was a built-in function. However, in Python 3, it is moved to functools module. Therefore to use it, you have to first import it as follows:

from functools import reduce

In [4]:
```
from functools import reduce
def do_sum(x1, x2):
    return x1 + x2
reduce(do_sum, [1, 2, 3, 4])
```

Out[4]:
```
10
```

Exercise:

In [41]:
```
from functools import reduce

#1 Capitalize all of the pet names and print the list
my_pets = ['sisi', 'bibi', 'titi', 'carla']

def cap(i):
    return i.upper()
print(list(map(cap,my_pets)))

#2 Zip the 2 lists into a list of tuples, but sort the numbers from lowest to highest.
my_strings = ['a', 'b', 'c', 'd', 'e']
my_numbers = [5,4,3,2,1]

print(list(zip(my_strings,sorted(my_numbers))))

#3 Filter the scores that pass over 50%
scores = [73, 20, 65, 19, 76, 100, 88]

def s50(i):
    return i>50
print(list(filter(s50,scores)))

#4 Combine all of the numbers that are in a list on this file using reduce
#(my_numbers and scores). What is the total?

def accumulator(acc, item):
    return acc + item

print(reduce(accumulator, (my_numbers + scores)))
```

```
['SISI', 'BIBI', 'TITI', 'CARLA']
[('a', 1), ('b', 2), ('c', 3), ('d', 4), ('e', 5)]
[73, 65, 76, 100, 88]
456
```

In [2]:
```
from functools import reduce
scores = [73, 20, 65, 19, 76, 100, 88,5,4,3,2,1]

def add_sc(a,b):
    return a+b
reduce(add_sc, scores)
```

Out[2]:
```
456
```

Lambda Expressions:

lambda expressions in Python are one time anonymous functions that you don't need more than once.

> lambda arguments : expression

https://www.w3schools.com/python/python_lambda.asp

In [6]:
```
x = lambda a, b : a * b
print(x(5, 6))
```

```
30
```

In [10]:
```
x = lambda a, b : a * b
print(x)
print(type(x))
```

```
<function <lambda> at 0x0000019F9B3FA9E0>
<class 'function'>
```

In [3]:
```
x = lambda a, b, c : a + b + c
print(x(5, 6, 2))
```

```
13
```

In [8]:
```
def myfunc(n):
  return lambda a : a * n

mydoubler = myfunc(2)

print(mydoubler(11))
```

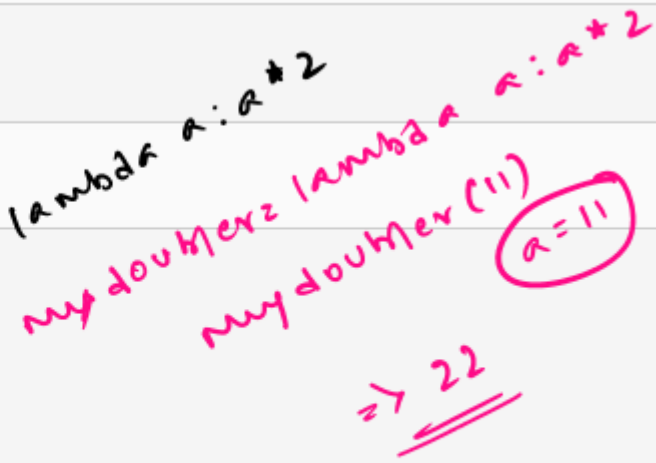
```
22
```

[7]:
```
x = lambda a, b, c : a + b + c
print(x(5, 6, 2))
```

```
13
```

[8]:
```
def myfunc(n):
  return lambda a : a * n

mydoubler = myfunc(2)

print(mydoubler(11))
```

```
22
```



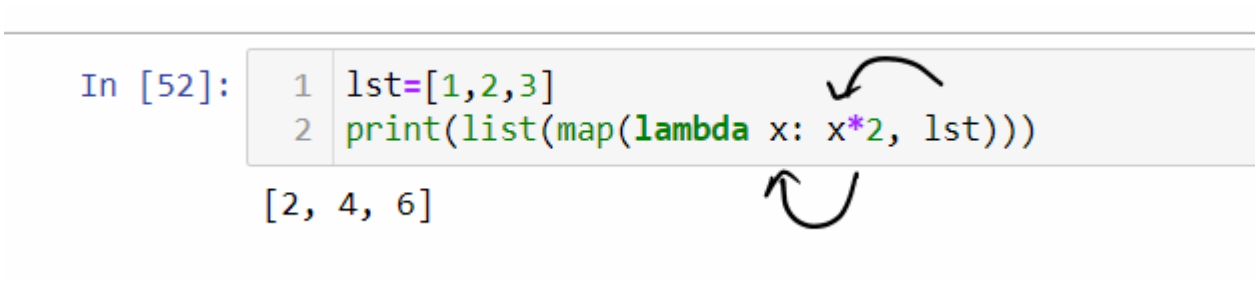
In [42]:
```
lst=[1,2,3]
def mulby2(i):
    return i*2
print(list(map(mulby2, lst)))
```

```
[2, 4, 6]
```

In [52]:
```
lst=[1,2,3]
print(list(map(lambda x: x*2, lst)))
```

```
[2, 4, 6]
```

In our case, we're saying, hey, I want to create a lambda expression. I'm going to take an item from my list and then I'm going to multiply that item with 2, and this automatically returns it again



In [54]:
```
lst=[1,2,3,8,9,11]
print(list(filter(lambda x: x%2 != 0, lst)))
```

```
[1, 3, 9, 11]
```

In [55]:
```
from functools import reduce
lst1=[1,2,3]
def accumulator(acc,i):
    print (acc,i)
    return acc+i # this value will become 'acc' now
print(reduce(accumulator,lst1,10))  #Here 10 is 'acc'.
```

```
10 1
11 2
13 3
16
```

In [56]:
```
from functools import reduce
lst1=[1,2,3]
print(reduce(lambda acc, x: acc+x, lst1))
```

```
6
```

IMP:

Exercise_lambda functions - Need to sort based on the 2nd element:

In [75]:
```
#Square
lst=[5,2,4]
print(list(map(lambda x: x**2, lst)))

#List sorting:
a = [(0, 2), (5, 2),(10, -1), (9, 9)]
a.sort() #sorts based on the 1st element (key=0)---here key is the index
print(a)

a.sort(key=lambda i: i[1]) #need to sort based on the 2nd element
print(a)
```

```
[25, 4, 16]
[(0, 2), (5, 2), (9, 9), (10, -1)]
[(10, -1), (0, 2), (5, 2), (9, 9)]
```

In [76]:
```
a=(1,5)
print(a[1])
```

```
5
```

List Comprehensions:

In [78]:
```
lst=[]
for x in 'hello':
    lst.append(x)
print(lst)
```

```
['h', 'e', 'l', 'l', 'o']
```

```
lst= [param for param in iterable]
```

So the way that I like to think about it is, hey, create a variable and then we're going to do a 'for' loop saying, hey, for each variable in the iterable added to the list.

In [80]:
```
lst=[z for z in 'hello']
print(lst)
```

```
['h', 'e', 'l', 'l', 'o']
```

In [88]:
```
lst=[z for z in 'hello']
print(lst)
lst2=[n for n in range(100)]
print(lst2)
lst3=[n**2 for n in range(100)]
print(lst3)
```

```
['h', 'e', 'l', 'l', 'o']
[0, 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, 3
4, 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, 6
6, 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, 9
8, 99]
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196, 225, 256, 289, 324, 361, 400, 441, 484, 529, 576, 625, 676, 729, 7
84, 841, 900, 961, 1024, 1089, 1156, 1225, 1296, 1369, 1444, 1521, 1600, 1681, 1764, 1849, 1936, 2025, 2116, 2209, 2304, 2401, 2
500, 2601, 2704, 2809, 2916, 3025, 3136, 3249, 3364, 3481, 3600, 3721, 3844, 3969, 4096, 4225, 4356, 4489, 4624, 4761, 4900, 504
1, 5184, 5329, 5476, 5625, 5776, 5929, 6084, 6241, 6400, 6561, 6724, 6889, 7056, 7225, 7396, 7569, 7744, 7921, 8100, 8281, 8464,
8649, 8836, 9025, 9216, 9409, 9604, 9801]
```

In [87]:
```
lst4=[n**2 for n in range(100) if n%2==0]
print(lst4)
```

```
[0, 4, 16, 36, 64, 100, 144, 196, 256, 324, 400, 484, 576, 676, 784, 900, 1024, 1156, 1296, 1444, 1600, 1764, 1936, 2116, 2304,
2500, 2704, 2916, 3136, 3364, 3600, 3844, 4096, 4356, 4624, 4900, 5184, 5476, 5776, 6084, 6400, 6724, 7056, 7396, 7744, 8100, 84
64, 8836, 9216, 9604]
```

In [1]:
```
lst=['one','one','two','three','one','two']
{x:lst.count(x) for x in lst}
```

Out[1]:
```
{'one': 3, 'two': 2, 'three': 1}
```

Set Comprehensions:

In [1]:
```
st={z for z in 'hello'}
print(st)
st2={n for n in range(100)}
print(st2)
st3={n**2 for n in range(100)}
print(st3)
st4={n**2 for n in range(100) if n%2==0}
print(st4)
```

```
{'o', 'l', 'e', 'h'}
{0, 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, 3
4, 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, 6
6, 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, 9
8, 99}
{0, 1, 1024, 4096, 4, 9216, 9, 16, 529, 3600, 4624, 25, 36, 2601, 49, 7225, 3136, 64, 576, 1089, 1600, 2116, 5184, 6724, 7744, 9
801, 81, 8281, 6241, 100, 625, 121, 4225, 1156, 8836, 3721, 144, 1681, 2704, 5776, 4761, 2209, 676, 169, 3249, 9409, 196, 1225,
5329, 729, 225, 1764, 7396, 6889, 7921, 2809, 256, 2304, 6400, 3844, 4356, 784, 1296, 8464, 289, 3364, 4900, 5929, 1849, 9025, 3
24, 841, 1369, 2401, 2916, 5476, 361, 3969, 900, 9604, 4489, 400, 1936, 7056, 7569, 3481, 6561, 1444, 8100, 5041, 441, 961, 250
0, 6084, 8649, 3025, 484, 2025, 1521, 5625}
{0, 256, 1024, 2304, 4, 900, 1156, 3844, 4096, 4356, 8836, 9604, 16, 144, 400, 784, 1296, 1936, 2704, 3600, 4624, 5776, 9216, 3
6, 676, 1444, 3364, 4900, 8100, 7056, 7744, 64, 576, 1600, 3136, 196, 324, 2116, 2500, 5184, 6084, 6400, 6724, 8464, 100, 484, 1
764, 2916, 5476, 7396}
```

Dictionary Comprehensions:

In [104…
```
#my_dict={key:value}
dict1={
    'a':2,
    'b':3
}
dict2={key:value**2 for key,value in dict1.items()}
print(dict2)

dict3={k:v**2 for k,v in dict1.items() if v%2==0}
print(dict3)
```

```
{'a': 4, 'b': 9}
{'a': 4}
```

In [106…
```
dict3={num:num**2 for num in [1,2,3]}
print(dict3)
```

```
{1: 1, 2: 4, 3: 9}
```

Exercise:

In [108…
```
some_list = ['a', 'b', 'c', 'b', 'd', 'm', 'n', 'n']
duplicates = []
for value in some_list:
    if some_list.count(value) > 1:
        if value not in duplicates:
            duplicates.append(value)
print(duplicates)
```

```
['b', 'n']
```

In [110…
```
ls = ['a', 'b', 'c', 'b', 'd', 'm', 'n', 'n']
dup = [i for i in ls if ls.count(i) > 1]
print(dup)
```

```
['b', 'b', 'n', 'n']
```

In [115…
```
ls = ['a', 'b', 'c', 'b', 'd', 'm', 'n', 'n']
dup = set([i for i in ls if ls.count(i) > 1])
# print(dup)
dup2= list(set([i for i in ls if ls.count(i) > 1]))
print(dup2)
```

```
['b', 'n']
```

Decorators:

In [121…
```
def hello():
    print("hellooo")
greet=hello()
print(greet)
```

```
hellooo
None
```

In [120…
```
def hello():
    print("helloooo")
greet=hello()
del hello()
print(greet)
```

```
  File "<ipython-input-120-d6d9b696989b>", line 4
    del hello()
        ^
SyntaxError: cannot delete function call
```

In [119…
```
def hello():
    print("helloooo")
greet=hello()
del hello
#here the name is getting deleted but the function is not deleted
print(greet)
```

```
hellooo
None
```

Higher Order Function:

> Higher Order Function: A function that accepts another function as parameter

In [27]:
```
def hello(func):
    func()

def greet():
    print("heyyy")
hello(greet)
```

```
heyyy
```

> Another way it can be a higher order function is --- if it's a function that returns another function

In [7]:
```
def greet2():
    def func():
        return 5
    return func()
greet2()
```

Out[7]:
```
5
```

In [1]:
```
def greet2():
    def func():
        return 5
    return func
greet2()
```

Out[1]:
```
<function __main__.greet2.<locals>.func()>
```

In [29]:
```
def greet2():
    def func():
        return 5
    return func
greet2()()
```

Out[29]:
```
5
```

> `greet2()--->now greet2` function gets called and returns func but we used `greet2()() hence it is func()` --- func gets called

> **Note:** Map, Filter, Reduce are higher order functions

Decorator supercharges our function. It's simply a function that wraps another function and enhances it or changes it,

In [7]:
```
def hello():
    print("heyyy")
hello()
```

```
heyyy
```

In [6]:
```
#Decorator
def my_decorator(func):
    def wrap_func():
        func()
    return wrap_func
@my_decorator
def hello():
    print("heyyy")
hello()
```

```
heyyy
```

In [20]:
```
#Decorator
def my_decorator(func):#func == hello
    def wrap_func():
        print("***")
        func()
        print("***")
    return wrap_func
@my_decorator
def hello():
    print("heyyy")
hello() #my_decorator(hello)()
@my_decorator
def bye():
    print("see u later")
bye()
```

```
***
heyyy
***
***
see u later
***
```

> In above code block, we `used wrap_func but not wrap_func()` -- didnt call the function because at the end we will call the function

check my_findings for more about this

In [11]:
```
#Decorator
def my_decorator(func):
    def wrap_func():
        print("*****")
        func()
        print("*****")
    return wrap_func #hello2 == wrap_func

# @my_decorator
def hello():
    print("heyyy")

hello2=my_decorator(hello) #hello2 == wrap_func
hello2()
```

```
*****
heyyy
*****
```

All I'm doing is wrapping my hello function with my decorator and assigning it to a variable.

So if we go step by step, we see that 'my decorator' receives a function in our case the 'hello' function,it then returns this wrapped function that has hello in here.

So we return this wrapped function. So hello2 now equals this wrapped function and then this wrapped function gets called and what is it called with.

Well it print(stars) then it runs the hello function and then it print(stars).

In [10]:
```python
#Decorator
def my_decorator(func):
    def wrap_func():
        print("*****")
        func()
        print("*****")
    return wrap_func

# @my_decorator
def hello():
    print("eyyy")

my_decorator(hello)()
```

```
*****
eyyy
*****
```

In [1]:
```python
def my_decorator(func):
        func()
def hello():
    print("heyyy")
my_decorator(hello)
```

```
heyyy
```

In [15]:
```python
def my_decorator(func):
    def wrap_func():
        print("*****")
        func()
        print("*****")
    return wrap_func
def hello():
    print("heyyy")

hello2=my_decorator(hello)
hello2()
```

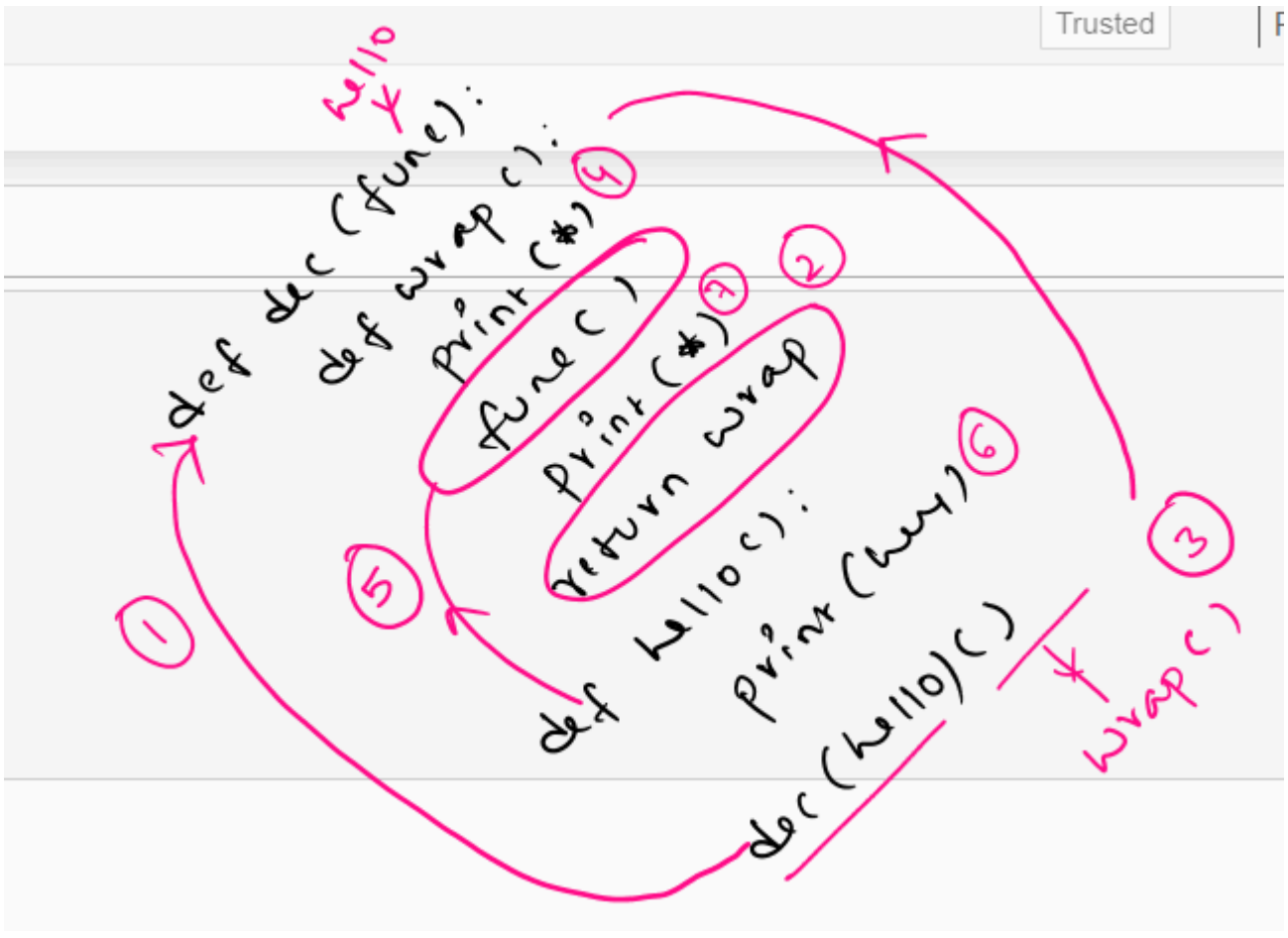
```
*****
heyyy
*****
```

In [5]:
```python
def my_decorator(func):
    def wrap_func():
        print("*****")
        func()
        print("*****")
    return wrap_func
def hello():
    print("heyyy")

my_decorator(hello)()
```

```
*****
heyyy
*****
```

> wrap_func equals to my_decorator(hello) and we are doing my_decorator(hello)() which means wrap_func().

> Now when wrap_func is called it prints stars, hello function gets called and prints stars



In [14]:
```
#Decorator
def my_decorator(func):
    def wrap_func():
        print("***")
        func()
        print("***")
    return wrap_func
@my_decorator
def hello(): #here hello function gets wrapped in the my_decorator function
    print("heyyy")
hello() #--->my_decorator(hello)()
```

```
***
heyyy
***
```

IMP:

> It can be a little (or a lot) confusing but wrap_func replaces the function that you decorated.

That's what the decorator does behind the scenes. So now whenever you call hello() in your code, wrap_func() gets called instead.

My_Findings: (8blocks below)

In [16]:
```
def my_decorator2(func):
    return func()
def hello2():
    print("heyyy")
my_decorator2(hello2)()#---func() which is hello2()
```

```
heyyy
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-16-7bc4e9d0950f> in <module>
      3 def hello2():
      4     print("heyyy")
----> 5 my_decorator2(hello2)()#---func() which is hello2()

TypeError: 'NoneType' object is not callable
```

In [12]:
```
def my_decorator2(func):
    return func()
def hello2():
    print("heyyy")
my_decorator2(hello2)
```

```
heyyy
```

In [14]:
```
def my_decorator2(func):
    return func
def hello2():
    print("heyyy")
my_decorator2(hello2)
```

Out[14]:
```
<function __main__.hello2()>
```

In [19]:
```
def my_decorator2(func): #func() is hello2
    return func
def hello2():
```

```
    print("heyyy")
my_decorator2(hello2)()
```

heyyy

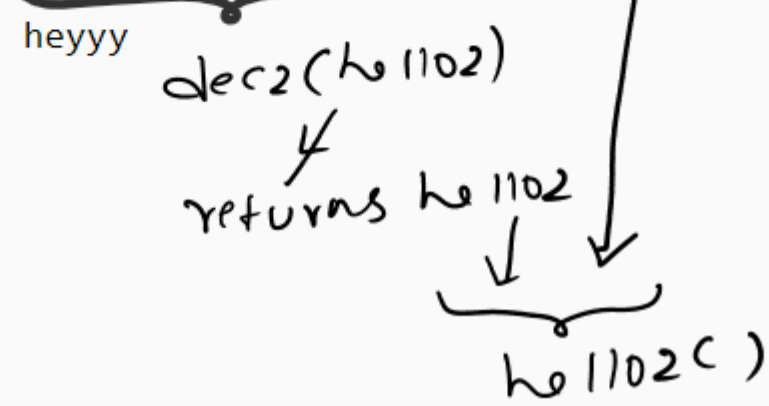
```
def my_decorator2(func):
    return func()
def hello2():
    print("heyyy")
my_decorator2(hello2)
```

heyyy

```
def my_decorator2(func):
    return func
def hello2():
    print("heyyy")
my_decorator2(hello2)
```

<function __main__.hello2()>

```
def my_decorator2(func):
    return func
def hello2():
    print("heyyy")
my_decorator2(hello2)()
```



In [24]:
```
def my_decorator2(func):
    return func
@my_decorator2
def hello2():
    print("heyyy")
hello2
```

Out[24]: <function __main__.hello2()>

In [26]:
```
def my_decorator2(func):
    return func
@my_decorator2
def hello2():
    print("heyyy")
hello2()
```

heyyy

In [23]:
```
def my_decorator(func):
    def wrap_func():
        print("*****")
        func()
        print("*****")
    return wrap_func
def hello():
    print("heyyy")

my_decorator(hello)()
```

```
*****
heyyy
*****
```

> wrap_func equals to my_decorator(hello) and we are doing my_decorator(hello)() which means wrap_func().

Now when wrap_func is called it prints stars, hello function gets called and prints stars

In [30]:
```
#Decorator
def my_decorator(func):
    def wrap_func():
        print("***")
        func()
        print("***")
    return wrap_func
@my_decorator
def hello(): #here hello function gets wrapped in the my_decorator function
```

```
    print("heyyy")
hello() # --- my_decorator(hello)() ---wrap()
```

```
***
heyyy
***
```

Decorators - With Parameters

In [31]:
```
#Decorator
def my_decorator(func):
    def wrap_func():
        print("***")
        func()
        print("***")
    return wrap_func
@my_decorator
def hello(greeting):
    print(greeting)
hello()
```

```
***
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-31-7b41f93094a8> in <module>
      9 def hello(greeting):
     10     print(greeting)
---> 11 hello()

<ipython-input-31-7b41f93094a8> in wrap_func()
      3     def wrap_func():
      4         print("***")
----> 5         func()
      6         print("***")
      7     return wrap_func

TypeError: hello() missing 1 required positional argument: 'greeting'
```

In [3]:
```
#Decorator
def my_decorator(func):
    def wrap_func():
        print("***")
        func(x)
        print("***")
    return wrap_func
@my_decorator
def hello(greeting):
    print(greeting)
hello('hii')
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-3-238407bd16d3> in <module>
      9 def hello(greeting):
     10     print(greeting)
---> 11 hello('hii')

TypeError: wrap_func() takes 0 positional arguments but 1 was given
```

In [4]:
```
#Decorator
def my_decorator(func):
    def wrap_func(x):
        print("***")
        func(x)
        print("***")
    return wrap_func
@my_decorator
def hello(greeting):
    print(greeting)
hello() #hello() is wrap_func()
#hello() is the wrapped function in decorator
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-4-65c14885e9f1> in <module>
      9 def hello(greeting):
     10     print(greeting)
---> 11 hello() #hello() is wrap_func()
     12 #hello() is the wrapped function in decorator

TypeError: wrap_func() missing 1 required positional argument: 'x'
```

In [34]:
```
#Decorator
def my_decorator(func):
    def wrap_func(x):
        print("***")
        func(x)
        print("***")
    return wrap_func
@my_decorator
def hello(greeting):
    print(greeting)
hello('hii')
```

```
***
hii
***
```

In [37]:
```
def my_decorator(func):
    def wrap_func(x):
        print("***")
        func(x)
        print("***")
    return wrap_func

def hello(greeting):
    print(greeting)
a=my_decorator(hello) #--- my_decorator(hello) is wrap_func
a('hi') #--- wrap_func('hi')
```

```
***
hi
***
```

In [39]:
```
#Decorator
def my_decorator(func):
    def wrap_func(x,y):
        print("***")
        func(x,y)
        print("***")
    return wrap_func
@my_decorator
def hello(greeting,emoji):
    print(greeting,emoji)
hello('hii',":)")
```

```
***
hii :)
***
```

In [40]:
```
#Decorator
def my_decorator(func):
    def wrap_func(*args, **kwargs):
        print("***")
        func(*args, **kwargs)
        print("***")
    return wrap_func
@my_decorator
def hello(greeting,emoji):
    print(greeting,emoji)
hello('hii',":)")
```

```
***
hii :)
***
```

IMP:

In [41]:
```
#Decorator
def my_decorator(func):
    def wrap_func(*args, **kwargs):
        print("***")
        func(*args, **kwargs)
        print("***")
    return wrap_func
@my_decorator
def hello(greeting,emoji=':('):
    print(greeting,emoji)
hello('hii')
```

```
***
hii :(
***
```

Decorator Pattern/syntax:

In [4]:
```
def my_decorator(func):
    def wrap_func(*args, **kwargs):
        func(*args, **kwargs)
    return wrap_func
```

Decorators Usage - Performance decorator:

In [1]:
```
#performance decorator.
from time import time
def performance(fn):
    def wrapper(*args, **kwargs):
        t1 = time()
        result = fn(*args, **kwargs)
        t2 = time()
        print(f'took {t2-t1}')
        return result
    return wrapper

@performance
```

```
def long_time():
    for i in range(10000):
        i*5

long_time()
```

```
took 0.0009989738464355469
```

In [1]:
```
#performance decorator.
from time import time
def performance(fn):
    def wrapper(*args, **kwargs):
        t1 = time()
        fn(*args, **kwargs)
        t2 = time()
        print(f'took {t2-t1}')
    return wrapper

@performance
def long_time():
    for i in range(10000):
        i*5

long_time()
```

```
took 0.0009980201721191406
```

```
# This is a simple decorator that will print the execution time of a function
import time

def timer_decorator(func):
    # This function will be used to wrap the original function
    def wrapper_func(*args, **kwargs):
        # Record the start time
        start_time = time.time()

        # Call the original function
        result = func(*args, **kwargs)

        # Record the end time
        end_time = time.time()

        # Calculate the elapsed time and print it
        print(f"Function {func.__name__} took {end_time - start_time} seconds to execute.")

        # Return the result of the original function
        return result

    # Return the new function that wraps the original function
    return wrapper_func

# Here's how we can use the decorator
@timer_decorator
def do_something(sleep_time):
    """A function that sleeps for a specified time."""
    time.sleep(sleep_time)

do_something(3)
```

In this example, `timer_decorator` is a decorator. It's a function that takes another function `func` as input and returns a new function `wrapper_func`. When the new function is called, it first records the start time, then calls the original function, then records the end time, calculates the elapsed time, and prints it. Finally, it returns the result of the original function.

The `@timer_decorator` line before the `do_something` function is a syntactic sugar that allows us to apply the decorator to the function without having to explicitly call it like this: `do_something = timer_decorator(do_something)`.

This is a simple example of how decorators can be used to modify the behavior of functions. They can be used for a wide variety of purposes, such as access control, caching, logging, etc.

Exercise:

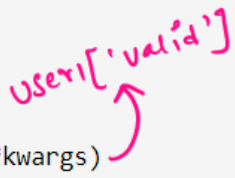
In [9]:
```
'''Create an @authenticated decorator that only allows the function to
run is user1 has 'valid' set to True:'''

user1 = {
    'name': 'Sorna',
    'valid': True
    #changing this will either run or not run the message_friends function.
}

def authenticated(fn):
  # code here
    def wrapped(*args,**kwargs):
        if args[0]['valid']:
```

```
                return fn(*args, **kwargs)
    return wrapped
@authenticated
def message_friends(user):
    print('message has been sent')

message_friends(user1)
```

```
message has been sent
```

In [9]:

```
'''Create an @authenticated decorator that only allows the function to
run is user1 has 'valid' set to True:'''

user1 = {
    'name': 'Sorna',
    'valid': True
    #changing this will either run or not run the message_friends function.
}

def authenticated(fn):
  # code here
    def wrapped(*args,**kwargs):
         if args[0]['valid']:
                return fn(*args, **kwargs)
    return wrapped
@authenticated
def message_friends(user):
    print('message has been sent')

message_friends(user1)
```




```
message has been sent
```

In [18]:

```
user1 = {
    'name': 'Sorna',
    'valid': True
    #changing this will either run or not run the message_friends function.
}
print(user1.items())
print(user1.keys())
print(user1['valid'])
```

```
dict_items([('name', 'Sorna'), ('valid', True)])
dict_keys(['name', 'valid'])
True
```

In [19]:

```
def fx(a):
    print(a+2)
fx(4)
```

```
6
```

Error Handling:

https://docs.python.org/3/library/exceptions.html

https://www.geeksforgeeks.org/errors-and-exceptions-in-python/

In [20]:

```
#SyntaxError:
def fun()
    pass
```

```
  File "<ipython-input-20-9d43d5256a1d>", line 2
    def fun()
            ^
SyntaxError: invalid syntax
```

In [22]:

```
#NameError
def fun():
    print(1+name)
fun()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-22-80ce75ecd3b7> in <module>
      2 def fun():
      3     print(1+name)
----> 4 fun()

<ipython-input-22-80ce75ecd3b7> in fun()
      1 #NameError
      2 def fun():
----> 3     print(1+name)
      4 fun()

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

In [23]:

```
#IndexError
lst=[1,2,3]
lst[8]
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-23-a47e7a97283b> in <module>
      1 lst=[1,2,3]
----> 2 lst[8]

IndexError: list index out of range
```

In [24]:
```
#KeyError
di={'a':4}
di['b']
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-24-dab76312e234> in <module>
      1 #KeyError
      2 di={'a':4}
----> 3 di['b']

KeyError: 'b'
```

In [26]:
```
#ZeroDivisionError
5/0
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-26-012fa6b14872> in <module>
      1 # ZeroDivisionError
----> 2 5/0

ZeroDivisionError: division by zero
```

Error Handling:

In [28]:
```
age=int(input("age: "))
print(age)
```

```
age: ers
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-28-712283a30419> in <module>
----> 1 age=int(input("age: "))
      2 print(age)

ValueError: invalid literal for int() with base 10: 'ers'
```

In [29]:
```
try:
    age=int(input("age: "))
    print(age)
except:
    print("Please enter numericals")
```

```
age: fds
Please enter numericals
```

In [ ]:
```
#To continue giving the input
while True:
    try:
        age=int(input("age: "))
        print(age)
    except:
        print("Please enter numericals")
#This code is continous loop
```

```
age: 12
12
age: fvsea
Please enter numericals
Please enter numericals
```

To break the loop when there are no errors

In [1]:
```
#To break the loop when there are no errors
while True:
    try:
        age=int(input("age: "))
        print(age)
    except:
        print("Please enter numericals")
    else:
        break
```

```
age: erd
Please enter numericals
age: eered
Please enter numericals
age: csd
Please enter numericals
age: 10
10
```

> Hey, 1st execute the 'try' block if any errors execute the 'except' block if no errors execute the 'else' block

The above loop will execute untill we get give correct input i.e., numbers

In [ ]:
```
while True:
    try:
        age=int(input("age: "))
        print(age)
        10/age
    except:
        print("Please enter numericals")
        #here we are getting the wrong error  message
    else:
        break
```

```
age: 0
0
Please enter numericals
Please enter numericals
```

In [3]:
```
while True:
    try:
        age=int(input("age: "))
        print(age)
        print(10/age)
    except ValueError:
        print("Please enter numericals")
        #now we get this message only when the value errors are raised
    else:
        break
```

```
age: df
Please enter numericals
age: 0
0
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-3-fc35f4b21aad> in <module>
      3         age=int(input("age: "))
      4         print(age)
----> 5         print(10/age)
      6     except ValueError:
      7         print("Please enter numericals") #now we get this message only when the value errors are raised

ZeroDivisionError: division by zero
```

Exceptions usage:

In [5]:
```
while True:
    try:
        age=int(input("age: "))
        print(age)
        print(10/age)
    except ValueError:
        print("Please enter numericals")
        #now we get this message only when the value errors are raised
    except ZeroDivisionError:
        print("Enter any number other than 0")
    else:
        print("No errors in the last iteration")
        break
```

```
age: few
Please enter numericals
age: 0
0
Enter any number other than 0
age: 2
2
5.0
No errors in the last iteration
```

> only 1 except block will be executed in an iteration, once the except block is executed, it returns to the start of while loop again

In [2]:
```
while True:
    try:
        age=int(input("age: "))
        print(age)
        print(10/age)
    except ValueError:
        print("Please enter numericals")
    except ValueError:
        print("!!!")
#only 1 except block will be executed in an iteration,
#once the except block is executed, it returns to the start of while loop again
    except ZeroDivisionError:
        print("Enter any number other than 0")
    else:
        print("No errors in the last iteration")
        break
```

```
age: gh
Please enter numericals
age: jk
Please enter numericals
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-2-19ad26d8eecb> in <module>
      1 while True:
      2     try:
----> 3         age=int(input("age: "))
      4         print(age)
      5         print(10/age)

~\anaconda3\lib\site-packages\ipykernel\kernelbase.py in raw_input(self, prompt)
    858                 "raw_input was called, but this frontend does not support input requests."
    859             )
--> 860         return self._input_request(str(prompt),
    861             self._parent_ident,
    862             self._parent_header,

~\anaconda3\lib\site-packages\ipykernel\kernelbase.py in _input_request(self, prompt, ident, parent, password)
    902             except KeyboardInterrupt:
    903                 # re-raise KeyboardInterrupt, to truncate traceback
--> 904                 raise KeyboardInterrupt("Interrupted by user") from None
    905             except Exception as e:
    906                 self.log.warning("Invalid Message:", exc_info=True)

KeyboardInterrupt: Interrupted by user
```

Error Handling 2:

In [16]:
```
def add(a, b):
    try:
        return a+b
    except:
        print("something is wrong")
add(1,'k')
```

```
something is wrong
```

In [19]:
```
def add(a, b):
    return a+b
add(1,'k')
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-19-fdc2f505c29a> in <module>
      1 def add(a, b):
      2     return a+b
----> 3 add(1,'k')

<ipython-input-19-fdc2f505c29a> in add(a, b)
      1 def add(a, b):
----> 2     return a+b
      3 add(1,'k')

TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

In [17]:
```
def add(a, b):
    try:
        return a+b
    except TypeError:
        print("please enter number")
add(1,'k')
```

```
please enter number
```

In [18]:
```
def add(a, b):
    try:
        return a+b
    except TypeError as err:
        print("please enter number & " + err)
#now we will get another error because we are concatinating a string to the error object
#(inbuilt exception)
add(1,'k')
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-18-e8c037afb6c7> in add(a, b)
      2     try:
----> 3         return a+b
      4     except TypeError as err:

TypeError: unsupported operand type(s) for +: 'int' and 'str'

During handling of the above exception, another exception occurred:

TypeError                                 Traceback (most recent call last)
<ipython-input-18-e8c037afb6c7> in <module>
      4     except TypeError as err:
      5         print("please enter number & " + err)
----> 6 add(1,'k')

<ipython-input-18-e8c037afb6c7> in add(a, b)
      3         return a+b
      4     except TypeError as err:
----> 5         print("please enter number & " + err)
      6 add(1,'k')

TypeError: can only concatenate str (not "TypeError") to str
```

In [22]:
```
def add(a, b):
    try:
        return a+b
    except TypeError as err:
        print(err)
        print(f"please enter number -  {err}")
add(1,'k')
```

```
unsupported operand type(s) for +: 'int' and 'str'
please enter number -  unsupported operand type(s) for +: 'int' and 'str'
```

In [34]:
```
#Multiple errors handling
def add(a, b):
    try:
        return a+b
    except(TypeError, ZeroDivisionError):
        print('oops')
add('fs',1)
def div(a, b):
    try:
        return a/b
    except(TypeError, ZeroDivisionError):
        print('oops')
div(1,0)
```

```
oops
oops
```

In [38]:
```
#Multiple errors handling
def add(a, b):
    try:
        return a/b
    except(TypeError, ZeroDivisionError) as err:
        print(err)
add('fs',1)
def div(a, b):
    try:
        return a/b
    except(TypeError, ZeroDivisionError) as err:
        print(err)
div(1,0)
```

```
unsupported operand type(s) for /: 'str' and 'int'
division by zero
```

try, except, finally

1. When there is any error in the "try" block, "except block" executes
2. when there is no errors in the "try" block, "else" block executes
3. after try,except, else --- at the end, "finally" block executes

In [40]:
```
while True:
    try:
        age=int(input("age: "))
        print(age)
        print(100/age)
    except ValueError:
        print("Please enter numericals")
    except ZeroDivisionError:
        print("Enter any number other than 0")
    else:
        print("No errors in the last iteration")
        break
    finally:#runs at the end after everything has been is executed
        print("ok, im finally done")
```

```
age: 25
25
4.0
No errors in the last iteration
ok, im finally done
```

In the above code, I get "No errors in the last iteration" because my try block succeeded, nothing has failed and now I break out of the While loop, but finally says, hey, no matter what, at the end of it all, I want you to finally do something.

In [43]:
```python
while True:
    try:
        age=int(input("age: "))
        print(age)
        print(100/age)
    except ValueError:
        print("Please enter numericals")
    except ZeroDivisionError:
        print("Enter any number other than 0")
    else:
        print("No errors in the last iteration")
        break
    finally:
        print("ok, im finally done")
```

```
age:
Please enter numericals
ok, im finally done
age: asd
Please enter numericals
ok, im finally done
age: 25
25
4.0
No errors in the last iteration
ok, im finally done
```

In [2]:
```python
while True:
    try:
        age=int(input("age: "))
        print(age)
        print(100/age)
    except ValueError:
        print("Please enter numericals")
        continue
    except ZeroDivisionError:
        print("Enter any number other than 0")
        break
    else:
        print("No errors in the last iteration")
        break
    finally:
        print("ok, im finally done")
    print("can you hear me?")
```

```
age: dfhb
Please enter numericals
ok, im finally done
age: 99
99
1.0101010101010102
No errors in the last iteration
ok, im finally done
```

In [3]:
```python
while True:
    try:
        age=int(input("age: "))
        print(age)
        print(100/age)
    except ValueError:
        print("Please enter numericals")
#         continue
    except ZeroDivisionError:
        print("Enter any number other than 0")
        break
    else:
        print("No errors in the last iteration")
        break
    finally:
        print("ok, im finally done")
    print("can you hear me?")
```

```
age: hjmh
Please enter numericals
ok, im finally done
can you hear me?
ok, im finally done
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-3-eac47bf64c40> in <module>
      1 while True:
      2     try:
----> 3         age=int(input("age: "))
      4         print(age)
      5         print(100/age)

~\anaconda3\lib\site-packages\ipykernel\kernelbase.py in raw_input(self, prompt)
    858                 "raw_input was called, but this frontend does not support input requests."
    859             )
--> 860         return self._input_request(str(prompt),
    861             self._parent_ident,
    862             self._parent_header,

~\anaconda3\lib\site-packages\ipykernel\kernelbase.py in _input_request(self, prompt, ident, parent, password)
    902             except KeyboardInterrupt:
    903                 # re-raise KeyboardInterrupt, to truncate traceback
--> 904                 raise KeyboardInterrupt("Interrupted by user") from None
    905             except Exception as e:
    906                 self.log.warning("Invalid Message:", exc_info=True)

KeyboardInterrupt: Interrupted by user
```

In [5]:

```
while True:
    try:
        age=int(input("age: "))
        print(age)
        print(100/age)
    except ValueError:
        print("Please enter numericals")
        continue
    except ZeroDivisionError:
        print("Enter any number other than 0")
        break
    else:
        print("No errors in the last iteration")
        break
    finally:
        print("ok, im finally done")
    print("can you hear me?")
# print never runs because, we break out of the loop.
```

```
age: 0
0
Enter any number other than 0
ok, im finally done
```

In [45]:

```
while True:
    try:
        age=int(input("age: "))
        print(age)
        print(100/age)
    except ValueError:
        print("Please enter numericals")
        continue
    except ZeroDivisionError:
        print("Enter any number other than 0")
        break
    else:
        print("No errors in the last iteration")
        break
    finally:
        print("ok, im finally done")
    print("can you hear me?")
#print never runs because, we break out of the loop.
```

```
age: gjhg
Please enter numericals
ok, im finally done
age: 5
5
20.0
No errors in the last iteration
ok, im finally done
```

> print never runs because, we break out of the loop.

> To print "can you hear me?", we need to give correct inputs (without any errors or errors whose except block doesn't have break)

In [46]:

```
while True:
    try:
        age=int(input("age: "))
        print(age)
        print(100/age)
    except ValueError:
        print("Please enter numericals")
        continue
    except ZeroDivisionError:
```

```
        print("Enter any number other than 0")
        break
    else:
        print("No errors in the last iteration")
    finally:
        print("ok, im finally done")
    print("can you hear me?")
    #To print this, we need to give correct inputs (without any errors or errors whose except block doesn't have break)
```

```
age: 25
25
4.0
No errors in the last iteration
ok, im finally done
can you hear me?
ok, im finally done
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-46-1eb21f5ca9fb> in <module>
      1 while True:
      2     try:
----> 3         age=int(input("age: "))
      4         print(age)
      5         print(100/age)

~\anaconda3\lib\site-packages\ipykernel\kernelbase.py in raw_input(self, prompt)
    858                 "raw_input was called, but this frontend does not support input requests."
    859             )
--> 860         return self._input_request(str(prompt),
    861             self._parent_ident,
    862             self._parent_header,

~\anaconda3\lib\site-packages\ipykernel\kernelbase.py in _input_request(self, prompt, ident, parent, password)
    902             except KeyboardInterrupt:
    903                 # re-raise KeyboardInterrupt, to truncate traceback
--> 904                 raise KeyboardInterrupt("Interrupted by user") from None
    905             except Exception as e:
    906                 self.log.warning("Invalid Message:", exc_info=True)

KeyboardInterrupt: Interrupted by user
```

Error Handling 3:

Raising Exceptions:

Well, if I type invalid number, I get "Please enter numericals", OK, I'm finally done" because I raise a value error and it goes into the 'except valueerror' block, 'finally' block

In [5]:

```
while True:
    try:
        age=int(input("age: "))
        print(age)
        print(100/age)
        raise ValueError("Hey check once")
    except ValueError:
        print("Please enter numericals")
        continue
    except ZeroDivisionError:
        print("Enter any number other than 0")
        break
    else:
        print("No errors in the last iteration")
    finally:
        print("ok, im finally done")
    print("can you hear me?")
    #To print this, we need to give correct inputs (without any errors)
```

```
age: dsf
Please enter numericals
ok, im finally done
ok, im finally done
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-5-b395b780be6b> in <module>
      1 while True:
      2     try:
----> 3         age=int(input("age: "))
      4         print(age)
      5         print(100/age)

~\anaconda3\lib\site-packages\ipykernel\kernelbase.py in raw_input(self, prompt)
    858                 "raw_input was called, but this frontend does not support input requests."
    859             )
--> 860         return self._input_request(str(prompt),
    861             self._parent_ident,
    862             self._parent_header,

~\anaconda3\lib\site-packages\ipykernel\kernelbase.py in _input_request(self, prompt, ident, parent, password)
    902             except KeyboardInterrupt:
    903                 # re-raise KeyboardInterrupt, to truncate traceback
--> 904                 raise KeyboardInterrupt("Interrupted by user") from None
    905             except Exception as e:
    906                 self.log.warning("Invalid Message:", exc_info=True)

KeyboardInterrupt: Interrupted by user
```

In [54]:

```
while True:
    try:
        age=int(input("age: "))
        print(age)
        raise ValueError("Hey check once")
    # Now we raised this valueerror and as there is no except valueerror block,
    # "Hey check once" prints
    except IndexError:
        print("Please check")
    else:
        print("No errors in the last iteration")
        break
    finally:
        print("ok, im finally done")
```

```
age: 5
5
ok, im finally done
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-54-8c5ae36116c0> in <module>
      3         age=int(input("age: "))
      4         print(age)
----> 5         raise ValueError("Hey check once")
      6     except IndexError:
      7         print("Please check")

ValueError: Hey check once
```

In [56]:

```
while True:
    try:
        age=int(input("age: "))
        print(age)
        raise Exception("Hey check once")
    except IndexError:
        print("Please check")
    else:
        print("No errors in the last iteration")
        break
    finally:
        print("ok, im finally done")
```

```
age: 5
5
ok, im finally done
---------------------------------------------------------------------------
Exception                                 Traceback (most recent call last)
<ipython-input-56-484bc79778cd> in <module>
      3         age=int(input("age: "))
      4         print(age)
----> 5         raise Exception("Hey check once")
      6     except IndexError:
      7         print("Please check")

Exception: Hey check once
```

Generators:

Generators are available in Python and allow us to generate a sequence of values over time.eg.. Range

> A generator is a special type of thing in Python that allows us to use a special keyword called `yield`, it can pause and resume functions.

In [58]:

```
def lst(num):
    res=[]
```

```
    for i in range(num):
        res.append(i)
    return res
mylst=lst(25)
print(mylst)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24]
```

My list is pointing to a location in memory.So this is taking up space right now, now, range is a generator and a generator is a little bit different because this is not being held in memory when we do this for loop, this range doesn't create on its own a giant list of, let's say, zero to ninety nine and then it starts iterating number with the for loop. A more efficient way is to use a generator and actually generate these one at a time without taking space and memory.

> Generators are actually iterable, that is everything that is a generator is iterable you can iterate over them, but not everything that is Iterable is a generator.

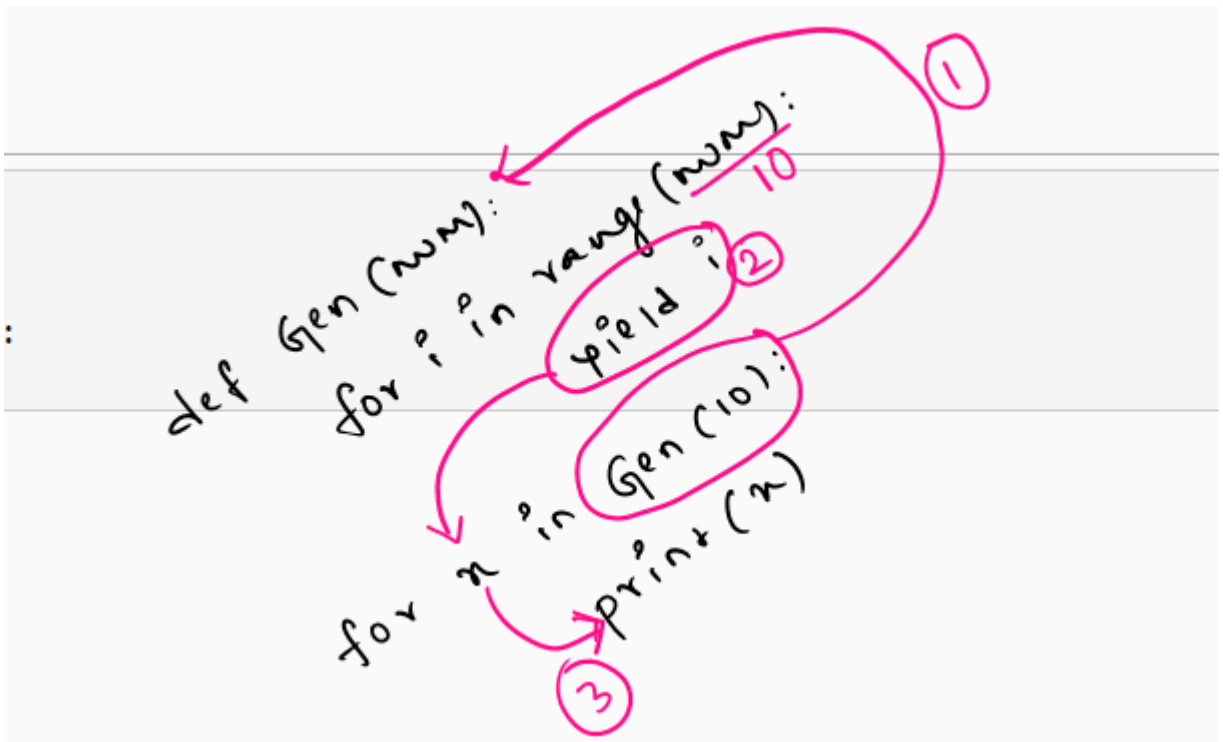
for example, a range is a generator.So that is always going to be iterable.But a list is an iterable, but it's not a generator.

So a generator is a subset of an iterable.

Generators are also functions

In [60]:
```
def Generator_Function(num):
    for i in range(num):
        yield i
for item in Generator_Function(10):
    print(item)
```

```
0
1
2
3
4
5
6
7
8
9
```



Did you see that what we just did here is similar to list, but instead of having to create that list in memory, it just kept going one by one. We only held one item in memory.

Well I can say for item in the generator function I'm going to Print the item.

Now, here, what's going to happen is we're going to loop over the generator function and we'll give it to the generator function, let's say a ten for now.

and we're going to print each item in the generator function because this is an iterable.

It's going to run and it's going to loop and for every item in this number, so that is range we're going to yield i

So it's going to keep looping and looping and we come back, when we loop over.

We're going to run this and the yield. Comes back and runs again.

In [62]:
```
def Generator_Function(num):
    for i in range(num):
        yield i
g= Generator_Function(10) #generator object
print(g)
```

```
<generator object Generator_Function at 0x00000142A8F207B0>
```

> **Note:** by using the `yield` keyword - we're able to turn a normal function into a generator function

In [68]:
```
def Generator_Function(num):
    for i in range(num):
        yield i*2
g= Generator_Function(10)
print(g)
print(next(g))
```

```
<generator object Generator_Function at 0x00000142A8F3FF20>
0
```

In [79]:
```
def Generator_Function(num):
    for i in range(num):
        yield i*2
g= Generator_Function(10)
print(g)
next(g)
next(g)
print(next(g))
```

```
<generator object Generator_Function at 0x00000142A9065900>
4
```

In [80]:
```
def Generator_Function(num):
    for i in range(num):
        yield i*2
g= Generator_Function(10)
print(g)
next(g)
print(g)
next(g)
print(g)
print(next(g))
```

```
<generator object Generator_Function at 0x00000142A9065E40>
<generator object Generator_Function at 0x00000142A9065E40>
<generator object Generator_Function at 0x00000142A9065E40>
4
```

IMP:

> We can see that in all the iteration the value is stored in the same

memory location, which means it only keeps the recent data erasing the old data.

You see yield pauses the function and come back to it when next is called. So if it has a yield keyword, it becomes a generator.

> It keeps track of this state, what we call the value, and it only keeps the most recent data in memory

In [80]:
```
def Generator_Function(num):
    for i in range(num):
        yield i*2
g= Generator_Function(10)
print(g)
next(g)
print(g)
next(g)
print(g)
print(next(g))
```

```
<generator object Generator_Function at 0x00000142A9065E40>
<generator object Generator_Function at 0x00000142A9065E40>
<generator object Generator_Function at 0x00000142A9065E40>
4
```

In [71]:
```
def Generator_Function(num):
    for i in range(num):
        yield i*2
g= Generator_Function(1)
print(g)
next(g)
next(g)
print(next(g))
```

```
<generator object Generator_Function at 0x00000142A90650B0>
```

```
---------------------------------------------------------------------------
StopIteration                             Traceback (most recent call last)
<ipython-input-71-ed4188c91f99> in <module>
      5 print(g)
      6 next(g)
----> 7 next(g)
      8 print(next(g))

StopIteration:
```

> **Note:** I get a stop iteration, you see next can be called as many times as we want, as many times until this range expires and when we exceed the number of items in the range.

Well, we get a stop iteration error.

Generators Performance:

In [74]:
```
from time import time
def performance(fn):
    def wrapper(*args, **kawrgs):
        t1 = time()
        result = fn(*args, **kawrgs)
        t2 = time()
        print(f'took {t2-t1} s')
        return result
    return wrapper

@performance
def long_time():
    print('1')
    for i in range(10000000):
        i*5
@performance
def long_time2():
    print('2')
    for i in list(range(10000000)):
        i*5


long_time()
long_time2()
```

```
1
took 0.44928669929504395 s
2
took 0.5824670791625977 s
```

We have 2Functions for us, a long_time that uses a range and another long_time2 that uses a range and then converts it into a list. long_time2 creates a list and then from that list that's created a memory on our computers, it's going to go one by one and multiply things by five long_time has range, which is a generator that comes built into Python that is going to just one by one, hold zero in memory and multiply by five, Hold one in memory, but multiply by five and keeps going, keeps going, keeps going and removes from memory any old numbers.

Creating our own For Loop:

In [76]:
```
def special_for(iterable):
  iterator = iter(iterable)
  while True:
    try:
        print(iterator)
        next(iterator)
    except StopIteration:
        break
special_for([1,2,3])
```

```
<list_iterator object at 0x00000142A90505B0>
<list_iterator object at 0x00000142A90505B0>
<list_iterator object at 0x00000142A90505B0>
<list_iterator object at 0x00000142A90505B0>
```

In [77]:
```
def special_for(iterable):
  iterator = iter(iterable)
  while True:
    try:
        print(iterator)
        print(next(iterator))
    except StopIteration:
        break
special_for([1,2,3])
```

```
<list_iterator object at 0x00000142A8E27070>
1
<list_iterator object at 0x00000142A8E27070>
2
<list_iterator object at 0x00000142A8E27070>
3
<list_iterator object at 0x00000142A8E27070>
```

Creating our own Range Function:

In [1]:
```
class MyGen:
    current = 0
    def __init__(self, first, last):
        self.first = first
        self.last = last
        MyGen.current = self.first
#this line allows us to use the current number as the starting point for the iteration

    def __iter__(self):
        return self

    def __next__(self):
        if MyGen.current < self.last:
            num = MyGen.current
            MyGen.current += 1
            return num
        raise StopIteration

gen = MyGen(1,10)
for i in gen:
    print(i)
```

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

In [3]:
```
# generator version
def fib(number):
    a =  0
    b = 1
    for i in range(number):
        yield a
        temp = a
        a = b
        b = temp + b

for x in fib(10):
    print(x)
```

```
0
1
1
2
3
5
8
13
21
34
```

In [4]:
```
def fib2(number):
    a =  0
    b = 1
    result = []
    for i in range(number):
        result.append(a)
        temp = a
        a = b
        b = temp + b
    return result

print(fib2(10))
```

```
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
```

In [8]:
```
from time import time
def performance(fn):
    def wrapper(*args, **kawrgs):
        t1 = time()
        result = fn(*args, **kawrgs)
        t2 = time()
        print(f'took {t2-t1} s')
        return result
    return wrapper
@performance
def fib(number):
    a =  0
    b = 1
    for i in range(number):
        yield a
        temp = a
        a = b
        b = temp + b

for x in fib(10):
    print(x)
@performance
```

```
def fib2(number):
    a =  0
    b = 1
    result = []
    for i in range(number):
        result.append(a)
        temp = a
        a = b
        b = temp + b
    return result

print(fib2(10))
```

```
took 0.0 s
0
1
1
2
3
5
8
13
21
34
took 0.0 s
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
```

Modules

Modules are used to Organize the code

IMP: each .py file is a module

> write the code in a file as save it as .py file - Import that file in other python file and can use its functionalities

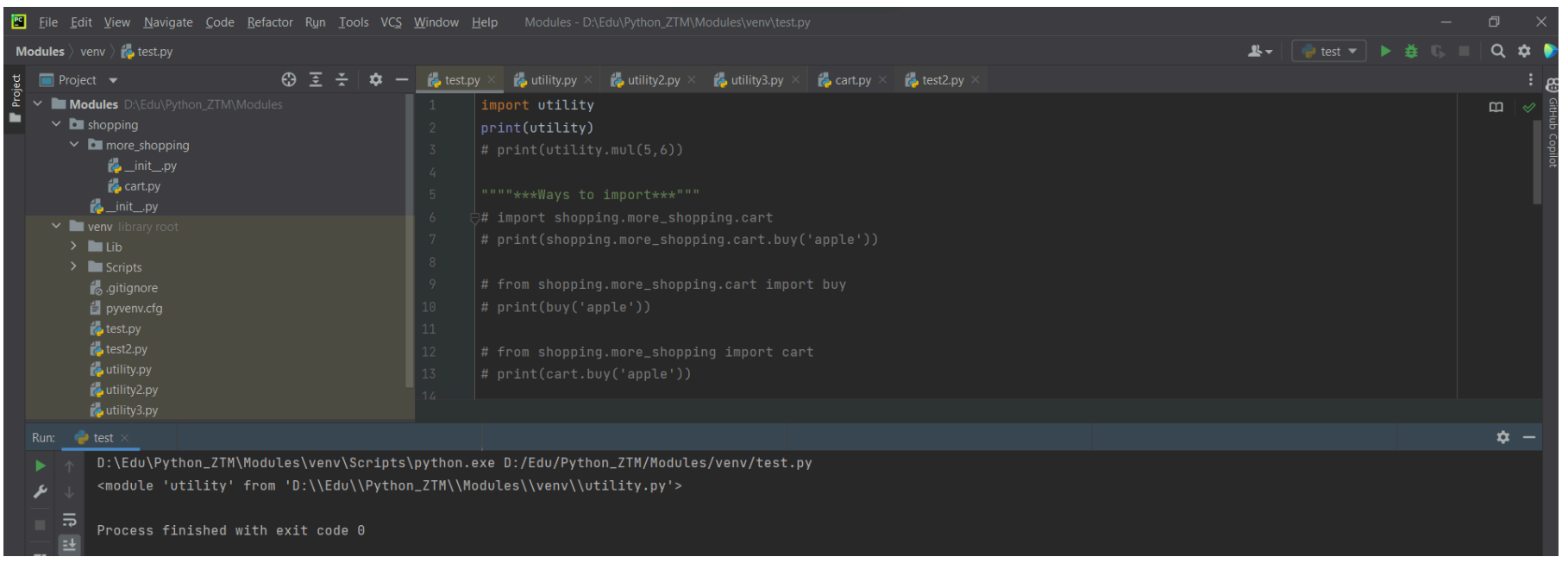
In [ ]:
```
import moduleexample
```

In [5]:
```
print (moduleexample)
```

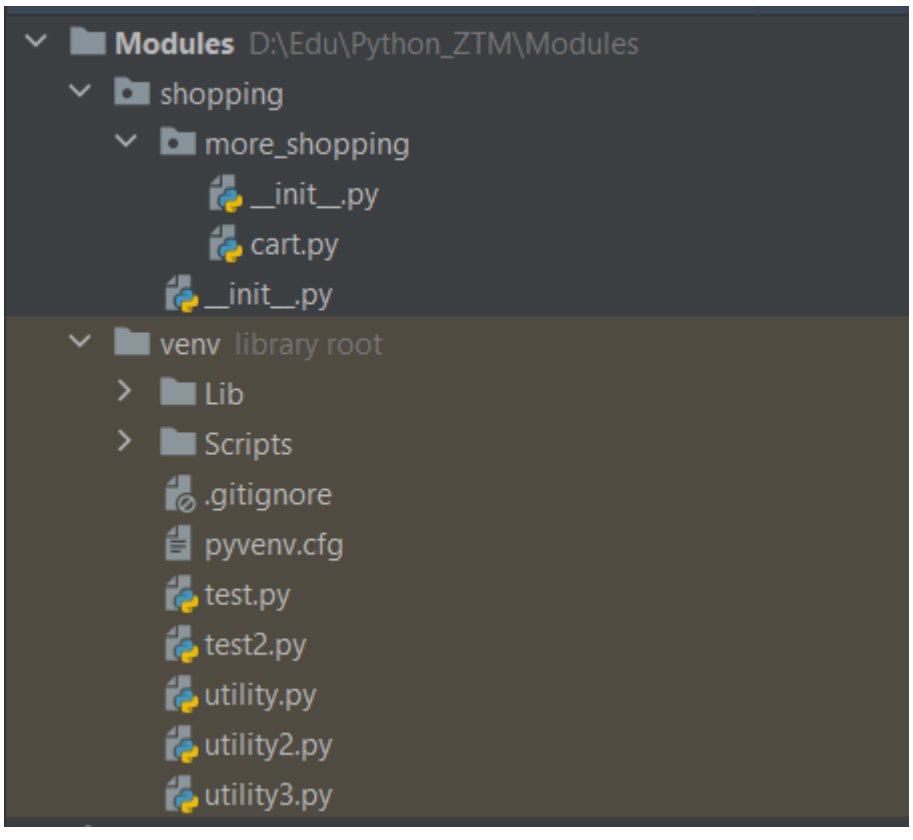
```
<module 'moduleexample' from 'D:\\Edu\\Python_ZTM\\moduleexample.py'>
```

In [7]:
```
moduleexample.mul(5,6)
```

Out[7]:
```
30
```



Files present in : "D:\Edu\Python_ZTM\Modules"



Packages:

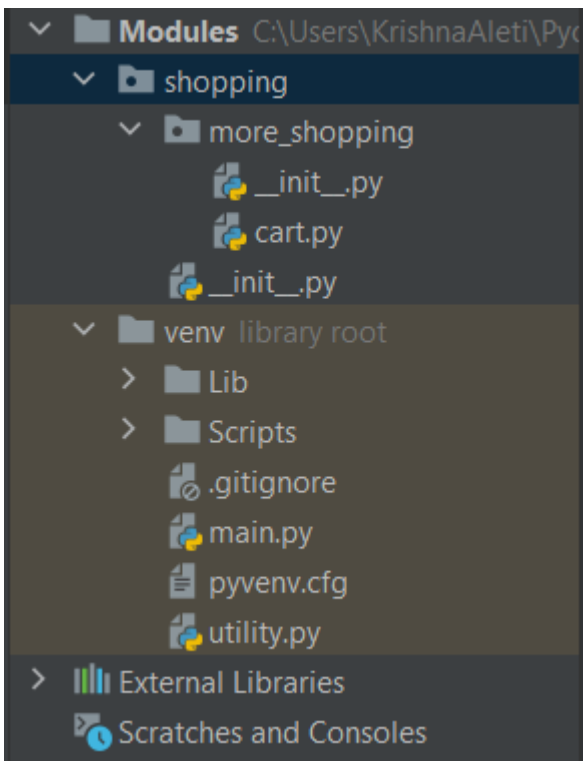
A package is simply a folder, and we learned about module, which are Python files,

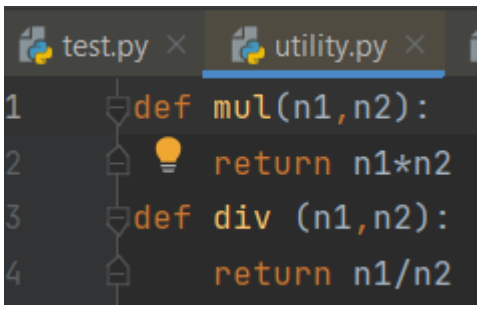
> A package is a level up, a package is a folder containing modules so you can have a package with multiple modules inside of them.

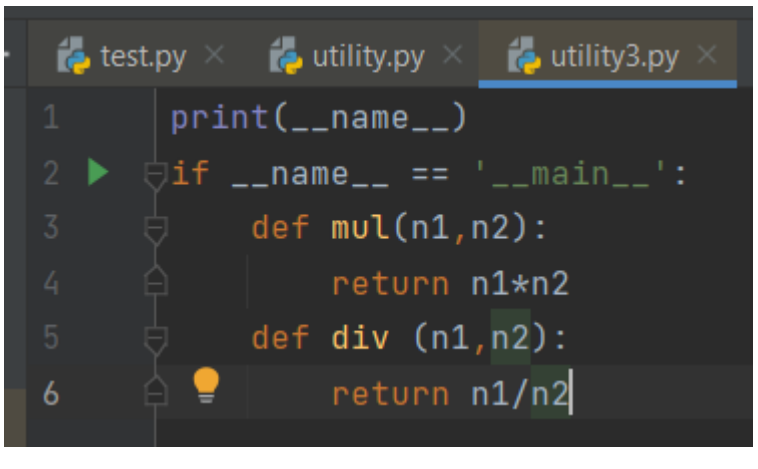
Note: to import some module from package: `import packagename.modulename`

> `__init__.py` indicates that the folder is an package

Any time you want to create a package, you'll see that we need to have an init file and it can just be completely empty.







test.py | utility.py | utility3.py | cart.py

```
1  print(__name__)
2
3  def buy(item):
4      cart = []
5      cart.append(item)
6      return cart
7
```

test.py | utility.py | utility3.py | cart.py | test2.py

```
1  print(__name__)
2  import utility2
3  if __name__ == '__main__':
4      print("This is main file")
5      print(utility2.mul(5,6))
```

test.py | utility.py | utility3.py | cart.py | test2.py | utility2.py

```
1  print(__name__)
2  def mul(n1,n2):
3      return n1*n2
4  def div (n1,n2):
5      return n1/n2
```

Executed test.py:

Modules D:\Edu\Python_ZTM\Modules
- shopping
  - more_shopping
    - __init__.py
    - cart.py
  - __init__.py

```
1  import utility
2  # print(utility)
3  print(utility.mul(5,6))
4
```

test

```
D:\Edu\Python_ZTM\Modules\venv\Scripts\python.exe D:/Edu/Python_ZTM/Modules/venv/test.py
30

Process finished with exit code 0
```

Project | test.py | utility3.py | utility.py | cart.py | test2.py | utility2.py

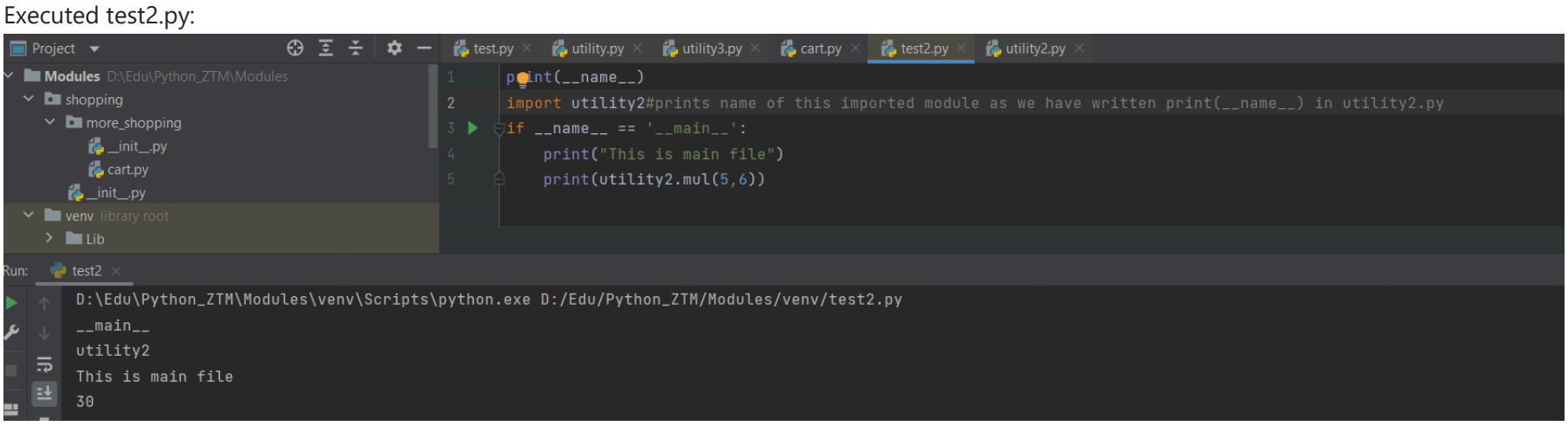
Modules D:\Edu\Python_ZTM\Modules
- shopping
  - more_shopping
    - __init__.py
    - cart.py
  - __init__.py
- venv library root
  - Lib
  - Scripts
  - .gitignore
  - pyvenv.cfg
  - test.py
  - test2.py
  - utility.py
  - utility2.py
  - utility3.py
- External Libraries

```
26
27 import shopping.more_shopping.cart
28 import utility3
29 print(__name__)
30 if __name__ == '__main__':
31     print("This is main file")
32
33 print(shopping.more_shopping.cart.buy('apple'))
34 print(utility3.mul(5,6))
35 '''will get error because we had written to def the functions only if the file name is main in utility3.py
36 but as we are importing here the name will be utility3 which is not equal to main hence the functions will not be
37 defined and we will get error, hence we will get error'''
38
39
```

Run: test

```
D:\Edu\Python_ZTM\Modules\venv\Scripts\python.exe D:/Edu/Python_ZTM/Modules/venv/test.py
Traceback (most recent call last):
  File "D:\Edu\Python_ZTM\Modules\venv\test.py", line 34, in <module>
    print(utility3.mul(5,6))
AttributeError: module 'utility3' has no attribute 'mul'
shopping.more_shopping.cart
utility3
__main__
This is main file
['apple']

Process finished with exit code 1
```

Executed test2.py:



Different ways of import:

In [ ]:
```
# import utility
# print(utility.mul(5,6))

# import shopping.more_shopping.cart
# print(shopping.more_shopping.cart.buy('apple'))

# from shopping.more_shopping.cart import buy
# print(buy('apple'))

# from shopping.more_shopping import cart
# print(cart.buy('apple'))


# from utility import div
# print(div(25,5))

# from utility import div,mul
# print(div(25,5))
# print(mul(25,5))

# from utility import *    #imports all the functions
# print(div(120,6))
# print(mul(20,5))
```

`__name__`:

In [2]:
```
#if __name__ == '__main__':
     #do something
```

> So this is a good way for us to run code only when it it the main file.

https://www.youtube.com/watch?v=Huz6bS0uLm4

`__name__` is a built-in variable which evaluates to the name of the current module. Thus it can be used to check whether the current script is being run on its own or being imported somewhere else by combining it with 'if statement', as shown below

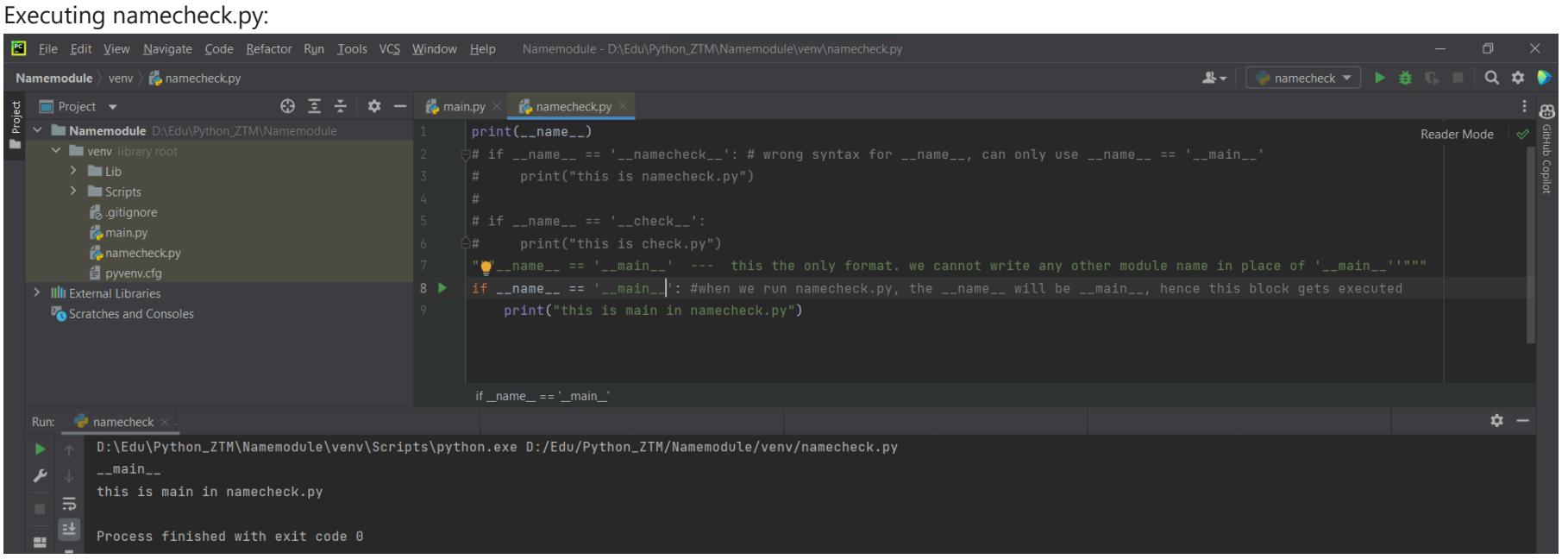
IMP:

if we run the file directly then `__name__` will equals to `__main__`, if we import some modules and execute then the `__name__` equals to the imported module name

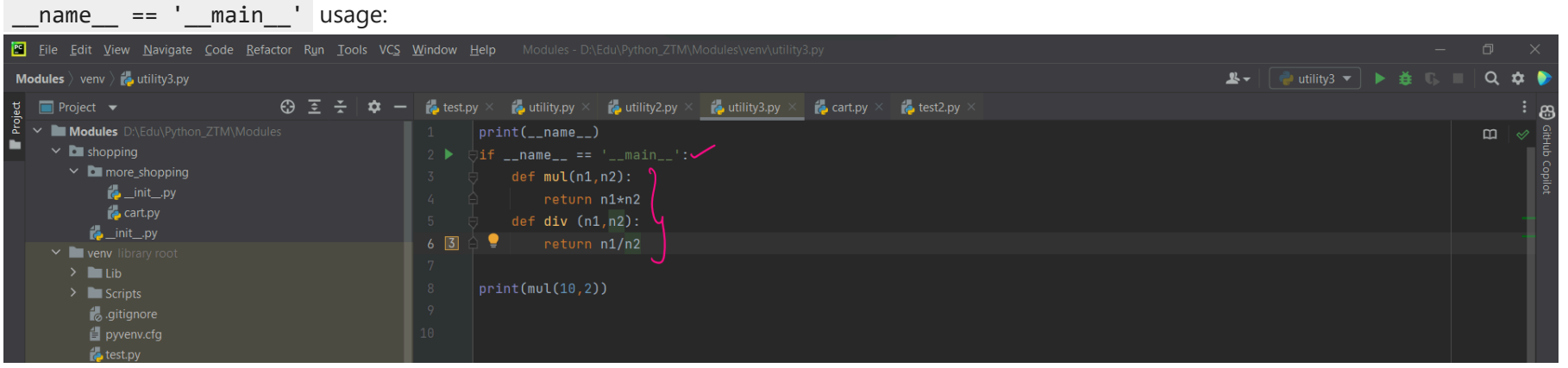
Files present in : "D:\Edu\Python_ZTM\Namemodule"

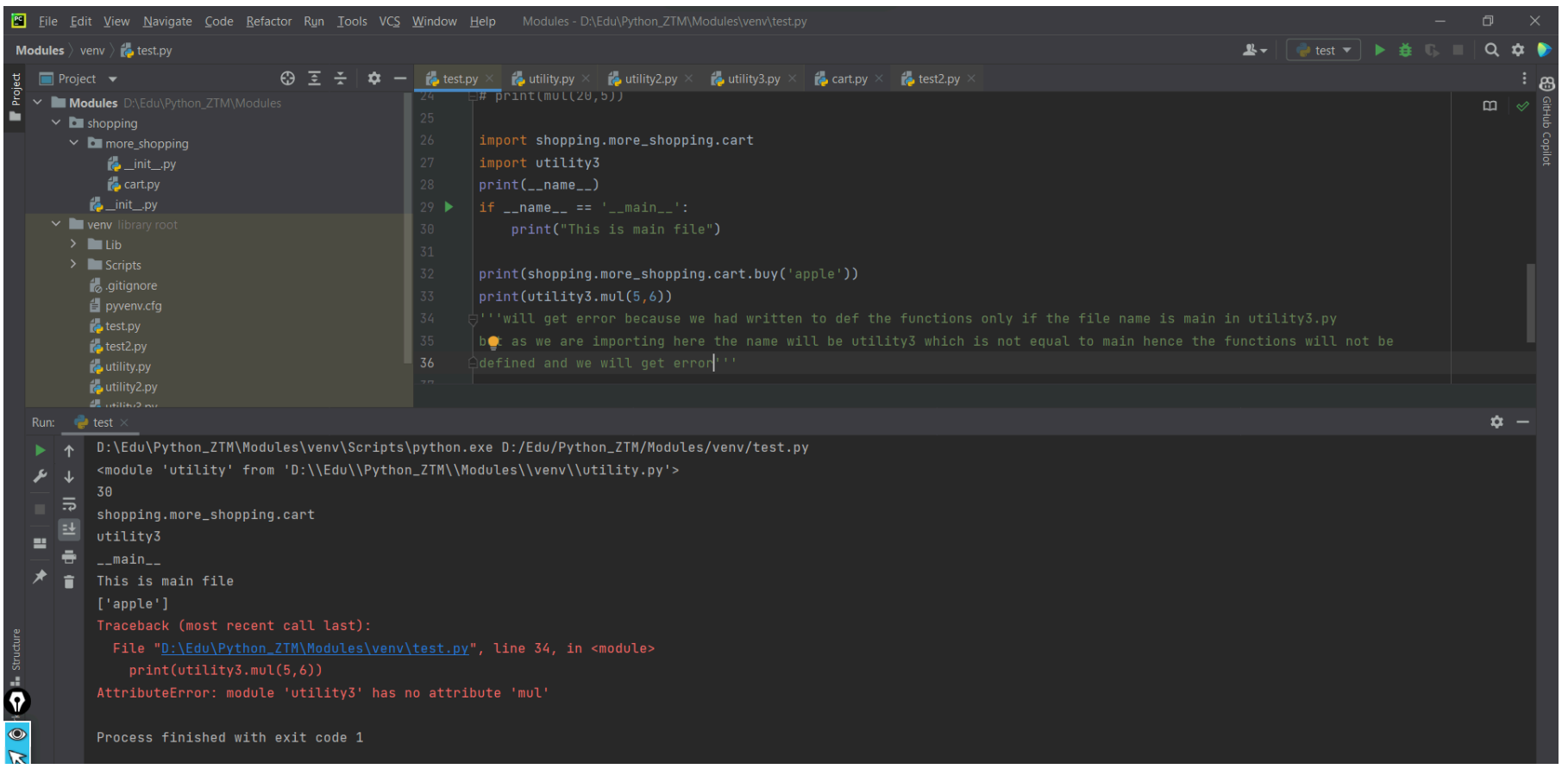
Executing main.py:

Executing namecheck.py:

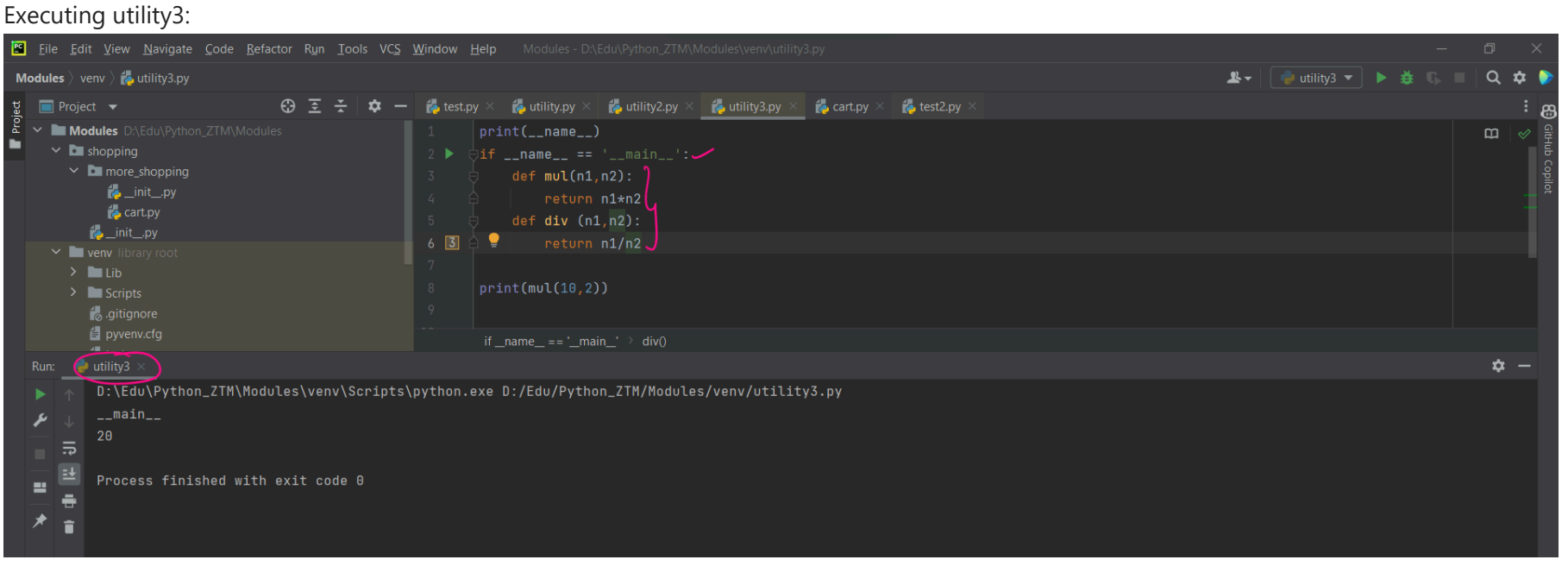


`__name__ == '__main__'` usage:





Executing utility3:



Built-in Modules:

https://docs.python.org/3/py-modindex.html

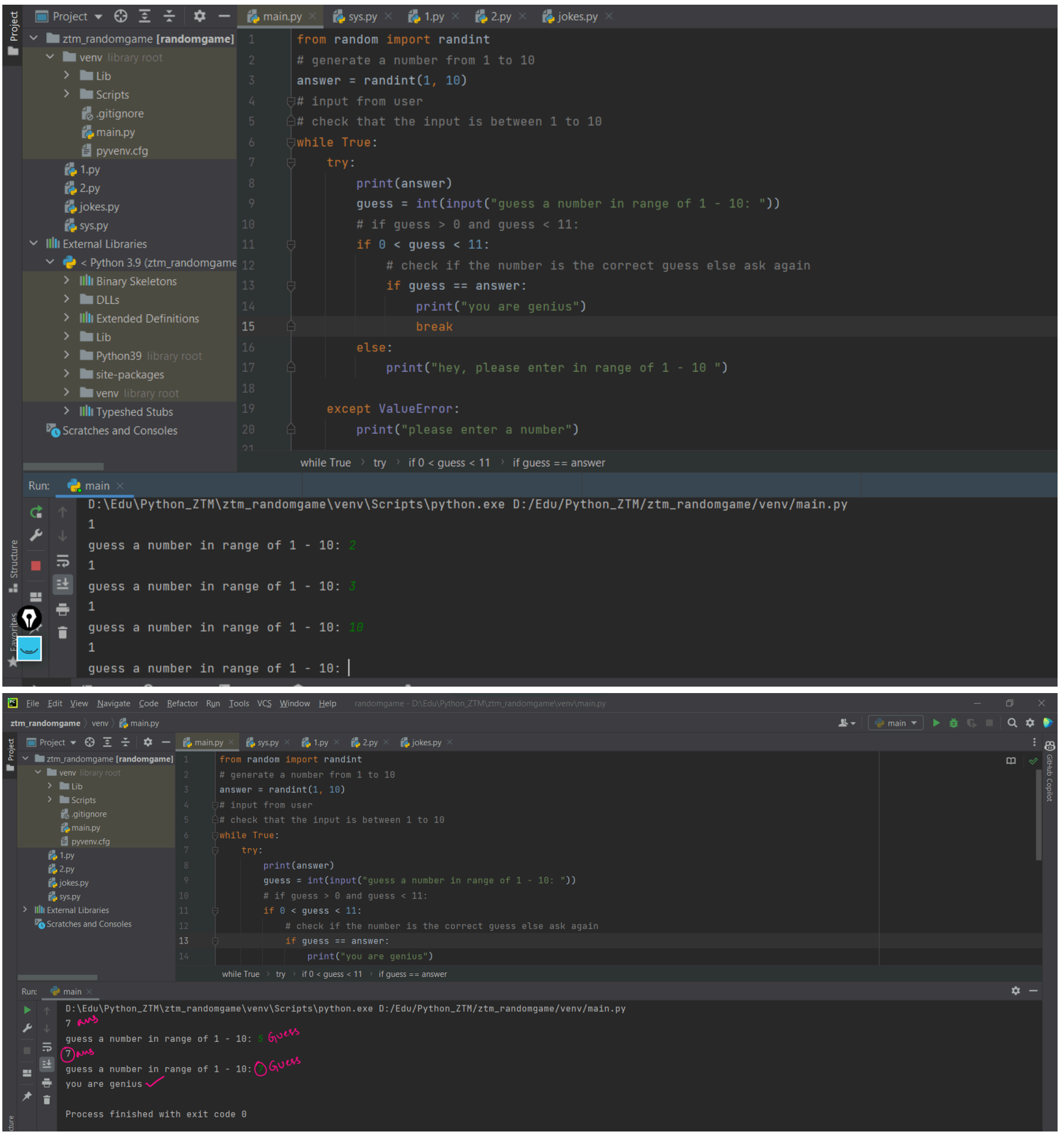
random

In [24]:
```
import random
print(random)
print(random.random()) # values b/w 0 & 1
print(random.randint(1,10)) # int values b/w mentioned start & stop
print(random.choice([1,2,3,4.4,5])) # picks some random values from the mentioned iterable
lst=[1,2,3,4,5,6]
random.shuffle(lst) #shuffles in place
print(lst)
```

```
<module 'random' from 'C:\\Users\\KrishnaAleti\\anaconda3\\lib\\random.py'>
0.7982464811398357
8
2
[2, 5, 4, 3, 6, 1]
```

In [21]:
```
from random import shuffle
lst=[1,2,3,4,5,6]
shuffle(lst) #shuffles in place
print(lst)
```

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



sys

In [23]:
```
import sys
print (sys)
```

```
<module 'sys' (built-in)>
```

This module provides access:

1. to some objects used or maintained by the interpreter and
2. to functions that interact strongly with the interpreter.

> Need to run the script using terminal/commandline/interpreter

Dynamic objects:

argv -- command line arguments; argv[0] is the script pathname if known

path -- module search path; path[0] is the script directory, else ''

modules -- dictionary of loaded modules

displayhook -- called to show results in an interactive session

excepthook -- called to handle any uncaught exception other than SystemExit

To customize printing in an interactive session or to install a custom

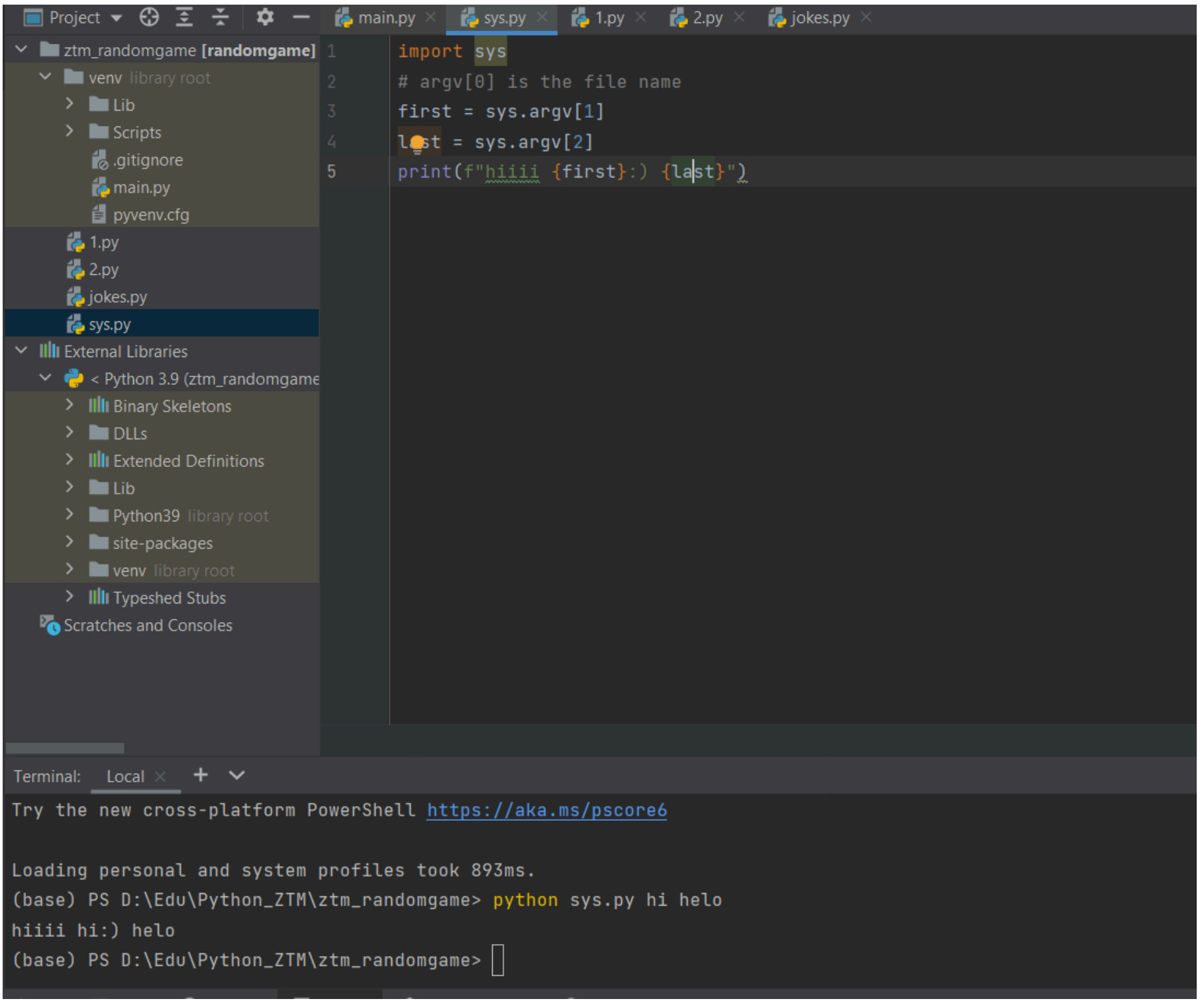
top-level exception handler, assign other functions to replace these.

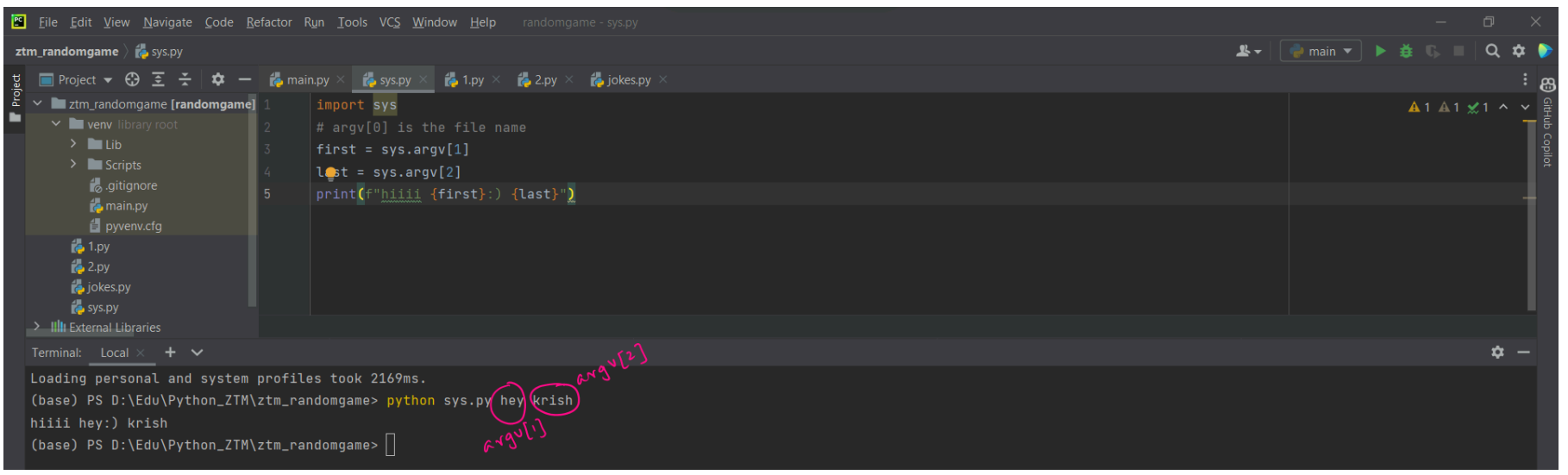
stdin -- standard input file object; used by input()

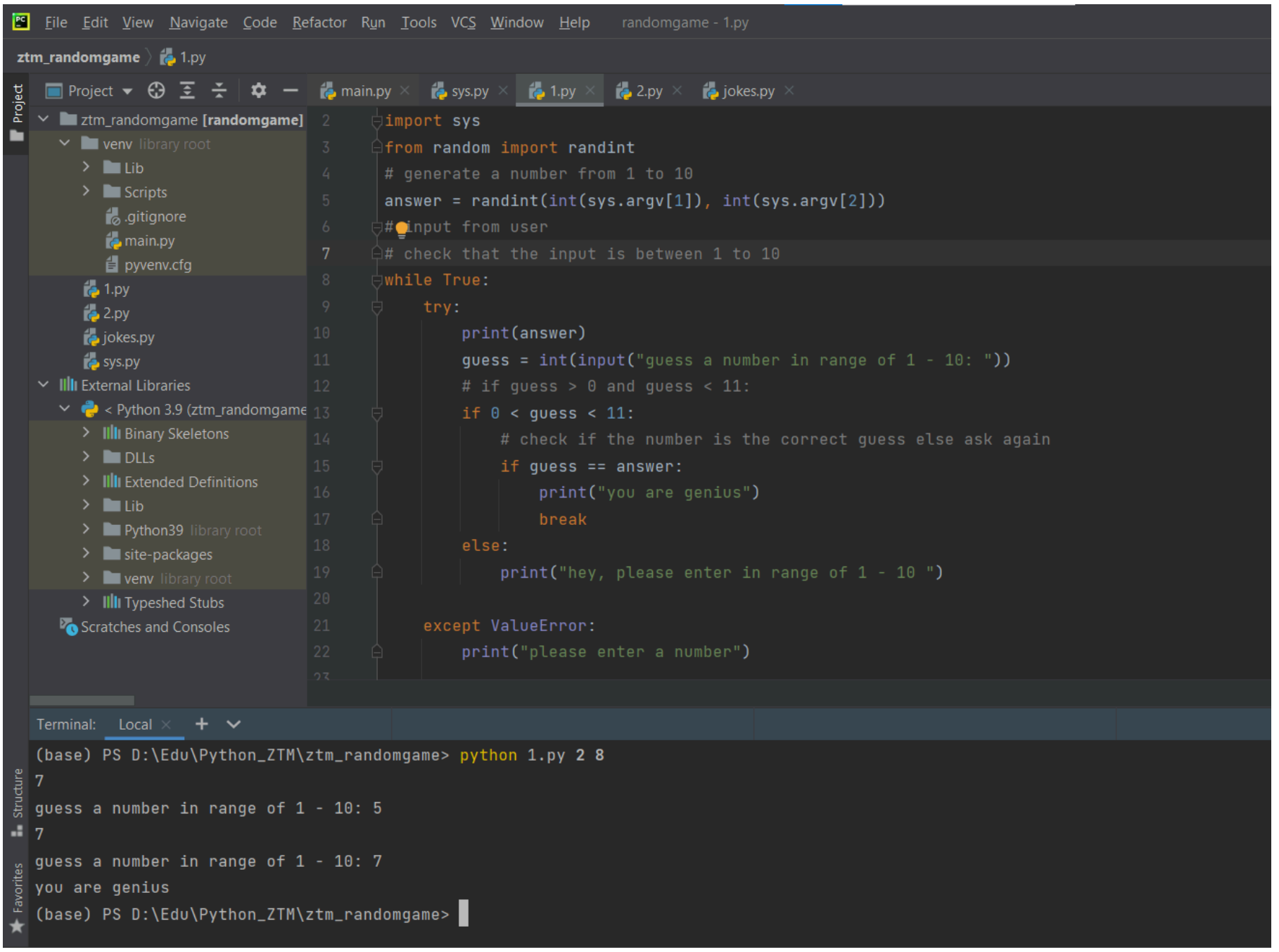
stdout -- standard output file object; used by print()

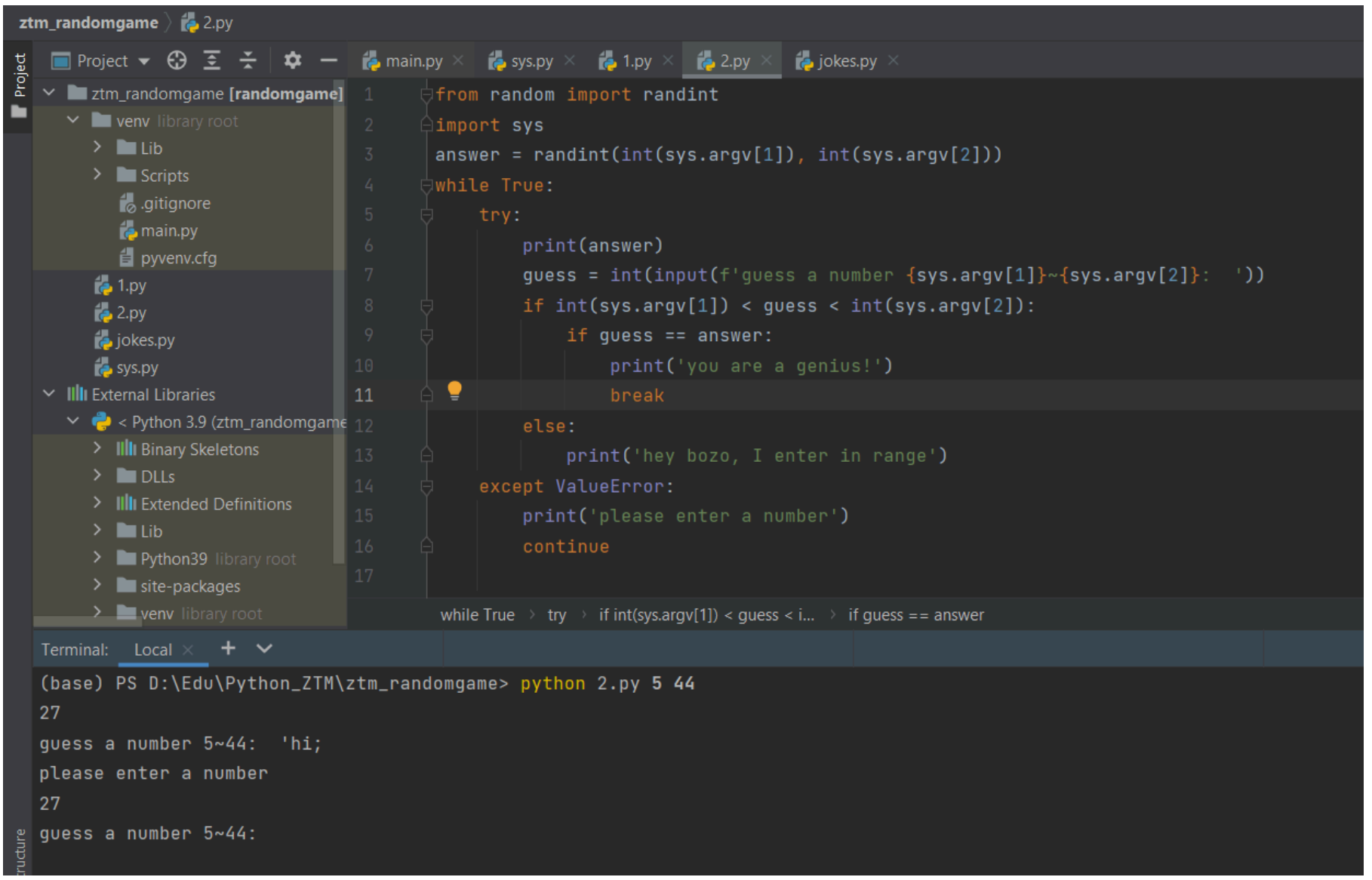
stderr -- standard error object; used for error messages

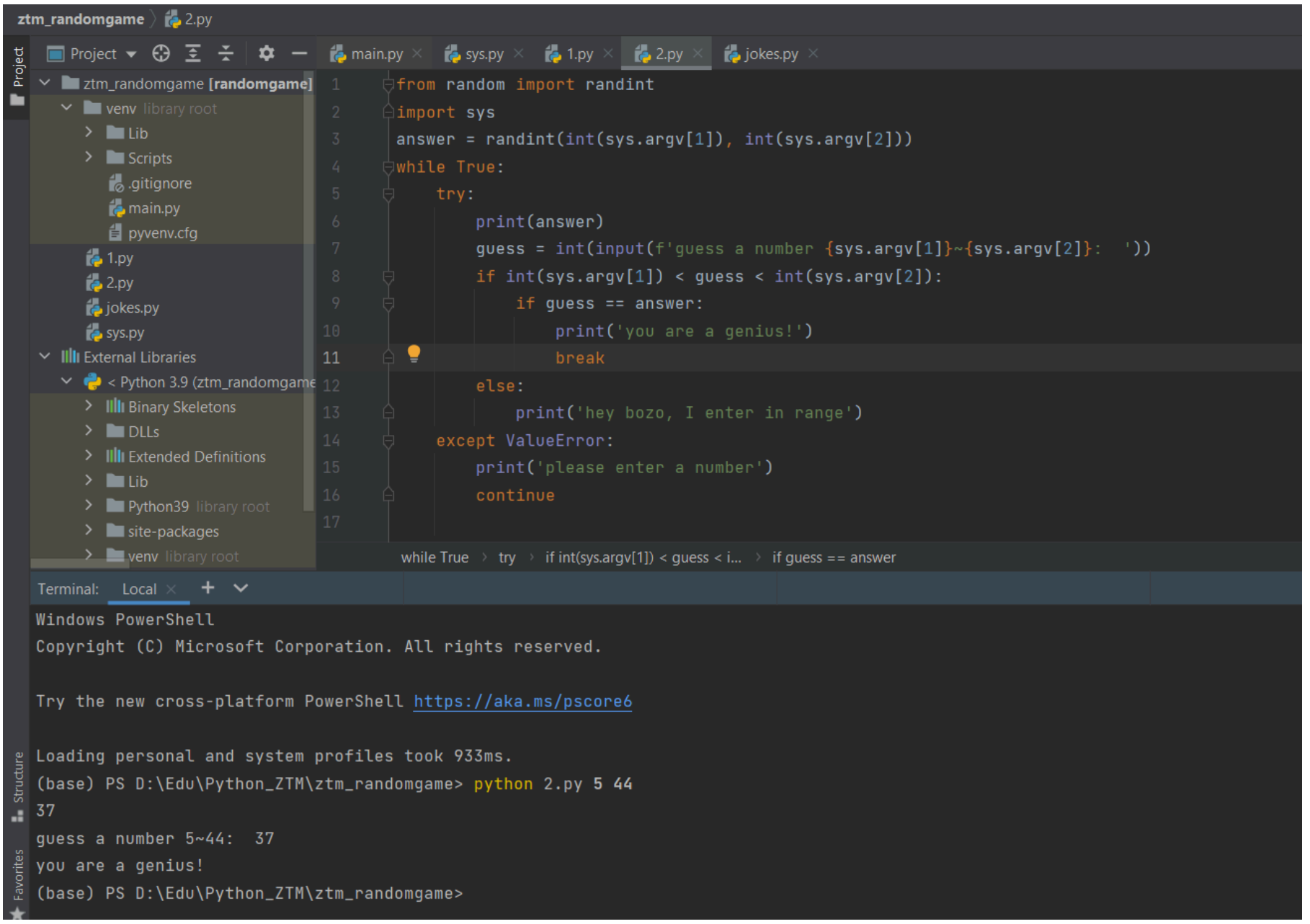
Exercise - "D:\Edu\Python_ZTM\ztm_randomgame"


Project
main.py
sys.py
1.py
2.py
jokes.py
ztm_randomgame [randomgame]
venv library root
Lib
Scripts
.gitignore
main.py
pyvenv.cfg
1.py
2.py
jokes.py
sys.py
External Libraries
< Python 3.9 (ztm_randomgame
Binary Skeletons
DLLs
Extended Definitions
Lib
Python39 library root
site-packages
venv library root
Typeshed Stubs
Scratches and Consoles
import sys
# argv[0] is the file name
first = sys.argv[1]
last = sys.argv[2]
print(f"hiiii {first}:) {last}")
Terminal: Local
Try the new cross-platform PowerShell https://aka.ms/pscore6
Loading personal and system profiles took 893ms.
(base) PS D:\Edu\Python_ZTM\ztm_randomgame> python sys.py hi helo
hiiii hi:) helo
(base) PS D:\Edu\Python_ZTM\ztm_randomgame>


File Edit View Navigate Code Refactor Run Tools VCS Window Help
randomgame - sys.py
ztm_randomgame sys.py
main
Project
main.py
sys.py
1.py
2.py
jokes.py
ztm_randomgame [randomgame]
venv library root
Lib
Scripts
.gitignore
main.py
pyvenv.cfg
1.py
2.py
jokes.py
sys.py
External Libraries
import sys
# argv[0] is the file name
first = sys.argv[1]
last = sys.argv[2]
print(f"hiiii {first}:) {last}")
GitHub Copilot
Terminal: Local
Loading personal and system profiles took 2169ms.
(base) PS D:\Edu\Python_ZTM\ztm_randomgame> python sys.py hey krish
hiiii hey:) krish
(base) PS D:\Edu\Python_ZTM\ztm_randomgame>
argv[1]
argv[2]


File Edit View Navigate Code Refactor Run Tools VCS Window Help randomgame - 1.py
ztm_randomgame 1.py
Project
main.py sys.py 1.py 2.py jokes.py
ztm_randomgame [randomgame]
venv library root
Lib
Scripts
.gitignore
main.py
pyvenv.cfg
1.py
2.py
jokes.py
sys.py
External Libraries
< Python 3.9 (ztm_randomgame
Binary Skeletons
DLLs
Extended Definitions
Lib
Python39 library root
site-packages
venv library root
Typeshed Stubs
Scratches and Consoles
import sys
from random import randint
# generate a number from 1 to 10
answer = randint(int(sys.argv[1]), int(sys.argv[2]))
#input from user
# check that the input is between 1 to 10
while True:
try:
print(answer)
guess = int(input("guess a number in range of 1 - 10: "))
# if guess > 0 and guess < 11:
if 0 < guess < 11:
# check if the number is the correct guess else ask again
if guess == answer:
print("you are genius")
break
else:
print("hey, please enter in range of 1 - 10 ")
except ValueError:
print("please enter a number")
Terminal: Local
(base) PS D:\Edu\Python_ZTM\ztm_randomgame> python 1.py 2 8
7
guess a number in range of 1 - 10: 5
7
guess a number in range of 1 - 10: 7
you are genius
(base) PS D:\Edu\Python_ZTM\ztm_randomgame>


ztm_randomgame 2.py
Project
main.py sys.py 1.py 2.py jokes.py
ztm_randomgame [randomgame]
venv library root
Lib
Scripts
.gitignore
main.py
pyvenv.cfg
1.py
2.py
jokes.py
sys.py
External Libraries
< Python 3.9 (ztm_randomgame
Binary Skeletons
DLLs
Extended Definitions
Lib
Python39 library root
site-packages
venv library root
from random import randint
import sys
answer = randint(int(sys.argv[1]), int(sys.argv[2]))
while True:
try:
print(answer)
guess = int(input(f'guess a number {sys.argv[1]}~{sys.argv[2]}: '))
if int(sys.argv[1]) < guess < int(sys.argv[2]):
if guess == answer:
print('you are a genius!')
break
else:
print('hey bozo, I enter in range')
except ValueError:
print('please enter a number')
continue
while True › try › if int(sys.argv[1]) < guess < i... › if guess == answer
Terminal: Local
(base) PS D:\Edu\Python_ZTM\ztm_randomgame> python 2.py 5 44
27
guess a number 5~44: 'hi;
please enter a number
27
guess a number 5~44:



Python Package index:

https://pypi.org/

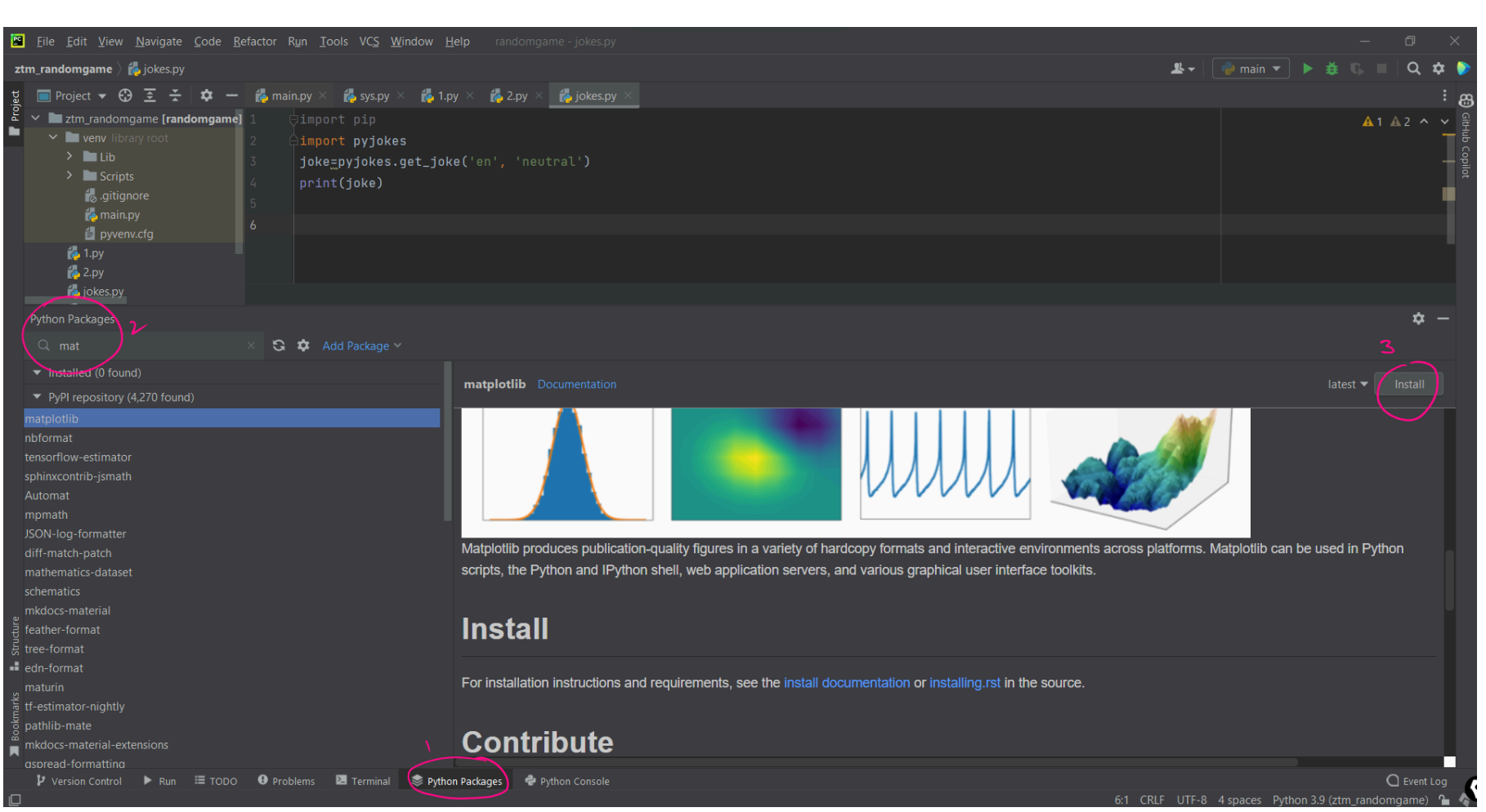
https://www.makeuseof.com/tag/install-pip-for-python/

Video - 176: How to install packages in pycharm

pyjokes: version - 0.5.0

1. the first (from right) number is usually a version where if you have bug fixes, you would go from zero to one to two so that you update
2. five will be a new release, maybe some added features.
3. And the last number (the major number) is usually breaking changes or major versions.

Installing Packages in Pycharm:



Virtual Environments:

Video - 177: How to use Virtual env

Useful Modules

In [9]:
```
from collections import Counter,defaultdict,OrderedDict
lst=[1,2,3,4,5,6,7,7,7]
sent="hey hi, i am learning python"
print(Counter(lst))
print(Counter(sent))
#It creates a dictionary that keeps track of how many times
#an item occurred in an iterable
```

```
Counter({7: 3, 1: 1, 2: 1, 3: 1, 4: 1, 5: 1, 6: 1})
Counter({' ': 5, 'h': 3, 'i': 3, 'n': 3, 'e': 2, 'y': 2, 'a': 2, ',': 1, 'm': 1, 'l': 1, 'r': 1, 'g': 1, 'p': 1, 't': 1, 'o':
1})
```

In [7]:
```
dict={'a':1,"b":2}
dict['a']
```

Out[7]:
```
1
```

In [37]:
```
print(int())
```

```
0
```

In [10]:
```
dict=defaultdict(int,{'a':1,"b":2})
print(dict['c'])
dict=defaultdict(lambda:8,{'a':1,"b":2})
print(dict['d'])
```

```
0
8
```

In [44]:
```
d=OrderedDict()
d['a']=1
d['b']=2

d2=OrderedDict()
d2['a']=1
d2['b']=2

print(d2==d)
```

```
True
```

In [45]:
```
d=OrderedDict()
d['a']=1
d['b']=2

d2=OrderedDict()
d2['b']=2
d2['a']=1

print(d2==d)
```

```
False
```

https://softwaremaniacs.org/blog/2020/02/05/dicts-ordered/en/

https://stackoverflow.com/questions/176011/python-list-vs-array-when-to-use

https://docs.python.org/3/py-modindex.html

In [16]:
```
import datetime
print(datetime.time(8,20,2))
print(datetime.date.today())
```

```
08:20:02
2022-07-19
```

In [19]:
```
from array import array
arr=array('i',[1,2,3])
print(arr)
print(arr[2])
```

```
array('i', [1, 2, 3])
3
```

Debugging:

https://docs.python.org/3/library/pdb.html

Video - 180: How to debug code

In [ ]:
```
#linting
#Use IDE or Editor
```

```
#able to read error
#pdb
```

In [20]:
```
import pdb
def add(n1,n2):
    pdb.set_trace()
    return n1+n2
add(4,'vefv')
```

```
> <ipython-input-20-178073f0c4da>(4)add()
      1 import pdb
      2 def add(n1,n2):
      3     pdb.set_trace()
----> 4     return n1+n2
      5 add(4,'vefv')

ipdb> 4
4
ipdb> 1
1
ipdb> 4
4
ipdb> knl
*** NameError: name 'knl' is not defined
ipdb> 4
4
ipdb> n1
4
ipdb> n2
'vefv'
ipdb> n1+n2
*** TypeError: unsupported operand type(s) for +: 'int' and 'str'
--KeyboardInterrupt--

KeyboardInterrupt: Interrupted by user
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-20-178073f0c4da> in <module>
      3     pdb.set_trace()
      4     return n1+n2
----> 5 add(4,'vefv')

<ipython-input-20-178073f0c4da> in add(n1, n2)
      2 def add(n1,n2):
      3     pdb.set_trace()
----> 4     return n1+n2
      5 add(4,'vefv')

TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

File I/O:

Opening a file and reading the entire file

In [14]:
```
myfile=open("test.txt")
print(myfile.read())
```

```
Hi my name is krish aleti
im fine.how are you?
```

Opening a file and reading the entire file multiple times

In [15]:
```
myfile=open("test.txt")
print(myfile.read())
print(myfile.read())
print(myfile.read())
```

```
Hi my name is krish aleti
im fine.how are you?
```

Read the file multiple times.

> I'm able to read the first time around, but these two times I'm not reading anything. Why is that?

Well, this open function has this idea of a cursor.That is, you can only read the file once and once you open, it returns a file object and the contents of the file you can read and the contents of the file are read with a cursor just like and printed onto the screen. But by the end of this first reading, the cursor is going to be at the end of the file. So now when it tries to read, it's going to be end of the file and nothing will be left there.

In [16]:
```
myfile=open("test.txt")
print(myfile.read())
myfile.seek(0)
#--- the value in seek is the index at which the cursor should be placed and read from
print(myfile.read())
myfile.seek(1)
print(myfile.read())
```

```
Hi my name is krish aleti
im fine.how are you?
Hi my name is krish aleti
im fine.how are you?
i my name is krish aleti
im fine.how are you?
```

Opening a file and reading line by line

In [17]:
```
myfile=open("test.txt")
print(myfile.readline())
```

```
Hi my name is krish aleti
```

In [18]:
```
myfile=open("test.txt")
print(myfile.readline())
print(myfile.readline())
```

```
Hi my name is krish aleti

im fine.how are you?
```

Opening a file and reading all lines

I get a list that contains the entire file, reads all the lines

In [22]:
```
myfile=open("test.txt")
print(myfile.readlines())
```

```
['Hi my name is krish aleti\n', 'im fine.how are you?']
```

closing the opened file

In [23]:
```
myfile=open("test.txt")
print(myfile.readlines())
myfile.close()
```

```
['Hi my name is krish aleti\n', 'im fine.how are you?']
```

Read, write, append

syntax: `with open("filename") as variable`

> So now 'with Open', you can actually don't have to worry about closing,

In [7]:
```
with open("test.txt") as file1:
#mode='r', file will be opened in read mode (bydeafult mode will be set to 'r')
    print(file1.readlines())
```

```
['Hi my name is krish aleti\n', ':)\n', 'im fine.how are you?']
```

read:

In [3]:
```
with open("test.txt",mode='r') as file1:
#mode='r', file will be opened in read mode
    print(file1.readlines())
```

```
['Hi my name is krish aleti\n', ':)\n', 'im fine.how are you?']
```

write:

In [4]:
```
with open("test.txt",mode='w') as file1:
    #mode='w', file will be opened in write mode
    print(file1.readlines())
```

```
---------------------------------------------------------------------------
UnsupportedOperation                      Traceback (most recent call last)
<ipython-input-4-350017c8089d> in <module>
      1 with open("test.txt",mode='w') as file1: #mode='r', file will be opened in read mode
----> 2     print(file1.readlines())

UnsupportedOperation: not readable
```

In [5]:
```
with open("test.txt",mode='rw') as file1:
    #mode='rw', file will be opened in read & write mode
    print(file1.readlines())
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-5-208a91a97de1> in <module>
----> 1 with open("test.txt",mode='rw') as file1: #mode='rw', file will be opened in read & write mode
      2     print(file1.readlines())

ValueError: must have exactly one of create/read/write/append mode
```

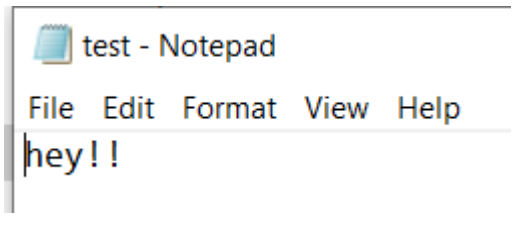
In [2]:
```
#In order to read and write, we do read plus.
with open("test.txt",mode='r+') as file1:
    #mode='r+', file will be opened in read & write mode
    print(file1.readlines())
```

```
['Hi my name is krish aleti\n', ':)\n', 'im fine.how are you?']
```

In [3]:
```
with open("test.txt",mode='r+') as file1:
    text=file1.write(":)")
    # now, the smiley will be written again at the starting of the file.
    print(text)
#by the way, this text file just gave us that we wrote 2 characters, The Smiley Face.
```

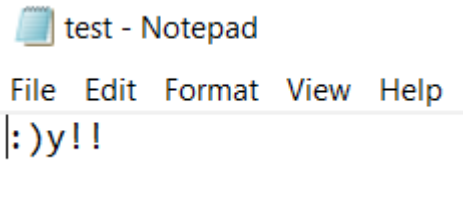
```
2
```

```
with open("test.txt",mode='r+') as file1:
    text=file1.write("hey!!")
    print(text)
```



read & write:

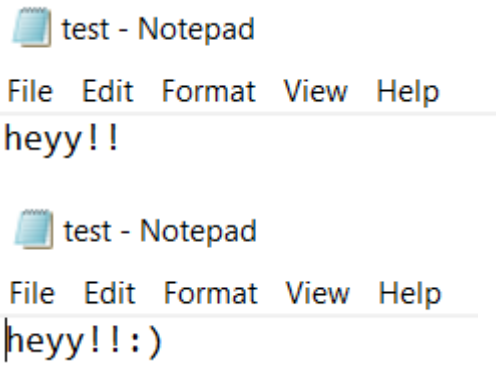
```
with open("test.txt",mode='r+') as file1:
    text=file1.write(":)")
    print(text)
```



Note: Remember, when we open a file, the cursor goes to zero and we write to the file, and if there's something already existing in there while we overwrite it.

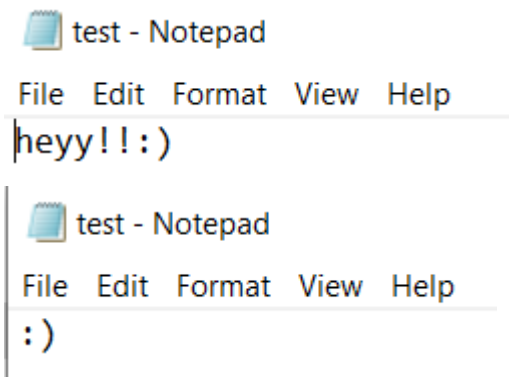
> To avoid the overwriting and write something at the end of the line we use 'append'

append:




```
with open("test.txt",mode='a') as file1:
    text=file1.write(":)")
    print(text)
```

write:




```
with open("test.txt",mode='w') as file1:
    text=file1.write(":)")
    print(text)
```

Imp:

1. mode = r+ (read and write), file.write() --- The cursor was set to zero and overwrote it
2. mode = w (write), file.write() --- It will assume this is a new file and we just want to tell, just add the smiley face.
3. mode = a (append), file.write() ---

Remember, when we open a file, the cursor goes to zero and we write to the file, and if there's something already existing in there while we overwrite it. To avoid the overwriting and write something at the end of the line we use 'append'

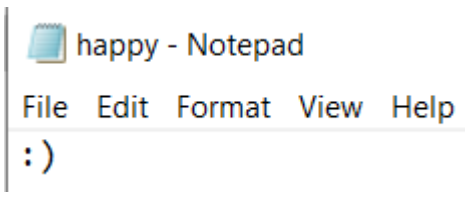
Creating a file:

In [4]:
```
#happy.txt file is not present
with open("happy.txt",mode='r+') as file1:
    text=file1.write(":)")
    print(text)
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-4-1c9f35d6819f> in <module>
----> 1 with open("happy.txt",mode='r+') as file1:
      2     text=file1.write(":)")
      3     print(text)

FileNotFoundError: [Errno 2] No such file or directory: 'happy.txt'
```

In [5]:
```
with open("happy.txt",mode='w') as file1:
    text=file1.write(":)")
    print(text)

#happy.txt file was not present but as we gave mode=w,
#a new file is created and written happy face into it
```

2



IMP:

> So write also creates a new file, if it doesn't exist already or overwrites the existing one.

File paths:

In [6]:
```
ls
```

```
 Volume in drive D is Data
 Volume Serial Number is B49E-AE1F

 Directory of D:\Edu\Python_ZTM

07-09-2021  04:18 PM    <DIR>          .
07-09-2021  04:18 PM    <DIR>          ..
01-09-2021  11:11 AM    <DIR>          .ipynb_checkpoints
06-09-2021  09:51 AM    <DIR>          __pycache__
25-06-2021  01:23 PM    <DIR>          All-Python-codes-of-ZTM-course-by-Andrei-Neagoie-master
21-12-2020  11:06 AM         1,324,179 aneagoie_ztm-python-cheat-sheet.pdf
26-08-2021  11:52 AM         1,917,593 Complete Python Developer in 2020_Zero to Mastery_1.ipynb
07-09-2021  04:18 PM           309,616 Complete Python Developer in 2021_Zero to Mastery_2.ipynb
07-09-2021  04:14 PM                 2 happy.txt
06-09-2021  06:39 PM    <DIR>          Modules
06-09-2021  06:39 PM    <DIR>          modules2
06-09-2021  06:39 PM    <DIR>          name
07-09-2021  02:50 PM               182 Python_ZTM.txt
07-09-2021  04:10 PM                86 sub.py
07-09-2021  04:10 PM                 2 test.txt
06-09-2021  06:58 PM    <DIR>          ztm_randomgame
07-09-2021  03:20 PM    <DIR>          ZTMPYCodes
               7 File(s)      3,551,660 bytes
              10 Dir(s)  964,049,620,992 bytes free
```

access files which is in another folder of the same directory:

In [7]:
```
with open("newfile.txt",mode='r') as file1:
    print(file1.read())
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-7-0abe3cefbb80> in <module>
----> 1 with open("newfile.txt",mode='r') as file1:
      2     print(file1.read())

FileNotFoundError: [Errno 2] No such file or directory: 'newfile.txt'
```

In [13]:
```
#opened a file which is in ZTMPYcodes folder
with open("ZTMPYCodes/newfile.txt",mode='r') as file1:
    print(file1.read())
```

```
this is new file in other folder
```

Accessing the file using the full path:

In [18]:
```
with open("D:\Edu\Krishna\check.txt", mode='r') as file1:
    print(file1.read())
```

```
checkkk
```

Imp:

`./` means current folder

`..` means back a folder

File IO Errors:

In [19]:
```
ls
```

```
 Volume in drive D is Data
 Volume Serial Number is B49E-AE1F

 Directory of D:\Edu\Python_ZTM

07-09-2021  05:28 PM    <DIR>          .
07-09-2021  05:28 PM    <DIR>          ..
01-09-2021  11:11 AM    <DIR>          .ipynb_checkpoints
06-09-2021  09:51 AM    <DIR>          __pycache__
25-06-2021  01:23 PM    <DIR>          All-Python-codes-of-ZTM-course-by-Andrei-Neagoie-master
21-12-2020  11:06 AM         1,324,179 aneagoie_ztm-python-cheat-sheet.pdf
26-08-2021  11:52 AM         1,917,593 Complete Python Developer in 2020_Zero to Mastery_1.ipynb
07-09-2021  05:28 PM           314,264 Complete Python Developer in 2021_Zero to Mastery_2.ipynb
07-09-2021  04:14 PM                 2 happy.txt
06-09-2021  06:39 PM    <DIR>          Modules
06-09-2021  06:39 PM    <DIR>          modules2
06-09-2021  06:39 PM    <DIR>          name
07-09-2021  02:50 PM               182 Python_ZTM.txt
07-09-2021  04:26 PM                86 sub.py
07-09-2021  04:10 PM                 2 test.txt
06-09-2021  06:58 PM    <DIR>          ztm_randomgame
07-09-2021  04:27 PM    <DIR>          ZTMPYCodes
               7 File(s)      3,556,308 bytes
              10 Dir(s)  964,049,612,800 bytes free
```

In [22]:
```
#sad.txt is not present in the current directory
with open("sad.txt", mode='r') as file1:
        print(file1.read())
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-22-af9b77a7f6df> in <module>
      1 #sad.txt is not present in the current directory
----> 2 with open("sad.txt", mode='r') as file1:
      3         print(file1.read())

FileNotFoundError: [Errno 2] No such file or directory: 'sad.txt'
```

In [20]:
```
#sad.txt is not present in the current directory
try:
    with open("sad.txt", mode='r') as file1:
        print(file1.read())
except FileNotFoundError as err:
    print("sad.txt is not present in the current directory")
```

```
sad.txt is not present in the current directory
```

In [21]:
```
#sad.txt is not present in the current directory
try:
    with open("sad.txt", mode='r') as file1:
        print(file1.read())
except FileNotFoundError as err:
    print("sad.txt is not present in the current directory")
    raise err
```

```
sad.txt is not present in the current directory
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-21-a7db6bea683a> in <module>
      5 except FileNotFoundError as err:
      6     print("sad.txt is not present in the current directory")
----> 7     raise err

<ipython-input-21-a7db6bea683a> in <module>
      1 #sad.txt is not present in the current directory
      2 try:
----> 3     with open("sad.txt", mode='r') as file1:
      4         print(file1.read())
      5 except FileNotFoundError as err:

FileNotFoundError: [Errno 2] No such file or directory: 'sad.txt'
```

> IO error usually happens when the computer or the machine you're on has some issue reading or writing or doing any sort of IO operation.

Translator:

https://pypi.org/project/translate/

text in the file - test.txt is:- Hi my name is krishna. I am good, how are you?

In [1]:
```
try:
    with open("test.txt",mode='r') as file3:
        print(file3.read())
except FileNotFoundError:
    print("test.txt file not found")
```

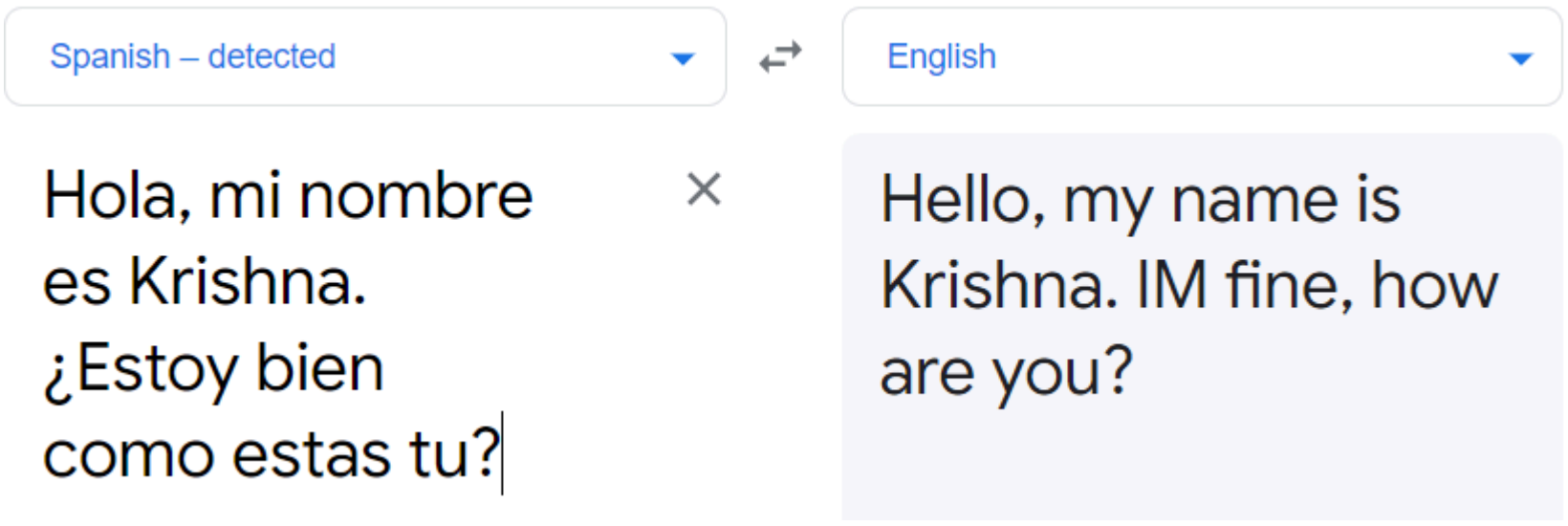
```
Hi my name is krishna. I am good, how are you?
```

In [24]:
```
pip install translate
```

```
Collecting translate
  Downloading translate-3.6.1-py2.py3-none-any.whl (12 kB)
Requirement already satisfied: lxml in c:\users\krishnaaleti\anaconda3\lib\site-packages (from translate) (4.6.1)
Requirement already satisfied: requests in c:\users\krishnaaleti\anaconda3\lib\site-packages (from translate) (2.24.0)
Collecting libretranslatepy==2.1.1
  Downloading libretranslatepy-2.1.1-py3-none-any.whl (3.2 kB)
Requirement already satisfied: click in c:\users\krishnaaleti\anaconda3\lib\site-packages (from translate) (7.1.2)
Requirement already satisfied: idna<3,>=2.5 in c:\users\krishnaaleti\anaconda3\lib\site-packages (from requests->translate) (2.1
0)
Requirement already satisfied: chardet<4,>=3.0.2 in c:\users\krishnaaleti\anaconda3\lib\site-packages (from requests->translate)
(3.0.4)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in c:\users\krishnaaleti\anaconda3\lib\site-packages (fro
m requests->translate) (1.25.11)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\krishnaaleti\anaconda3\lib\site-packages (from requests->translat
e) (2020.6.20)
Installing collected packages: libretranslatepy, translate
Successfully installed libretranslatepy-2.1.1 translate-3.6.1
Note: you may need to restart the kernel to use updated packages.
```

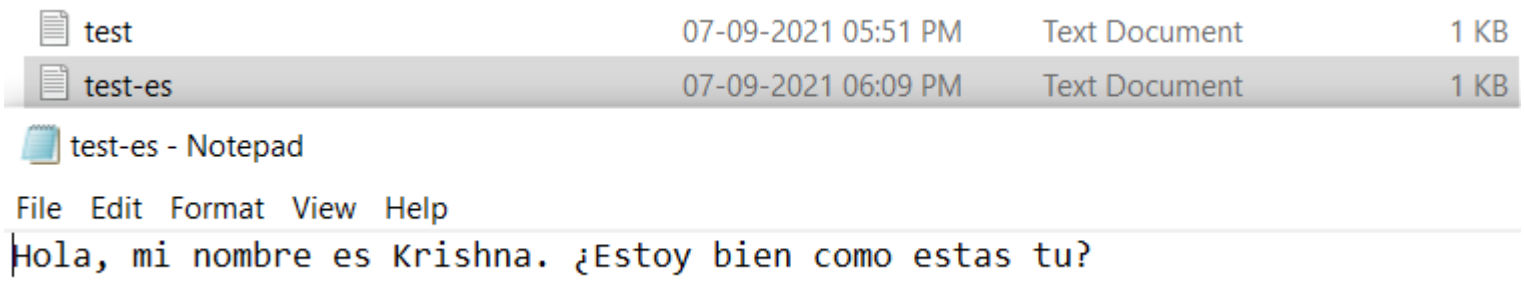
In [5]:
```
from translate import Translator
translator= Translator(to_lang="es")
# es:Spanish #https://en.wikipedia.org/wiki/ISO_639-1
try:
    with open("test.txt",mode='r') as file3:
        text=file3.read()
        translation = translator.translate(text)
        print(translation)
except FileNotFoundError:
    print("test.txt file not found")
# text in the file is:- Hi my name is krishna. I am good, how are you?
```

```
Hola, mi nombre es Krishna. ¿Estoy bien como estas tu?
```



In [6]:
```
from translate import Translator
translator= Translator(to_lang="es")
#es:Spanish #https://en.wikipedia.org/wiki/ISO_639-1
try:
    with open("test.txt",mode='r') as file3:
        text=file3.read()
        translation = translator.translate(text)
    with open("test-es.txt", mode='w') as file2:
        file2.write(translation)
except FileNotFoundError:
    print("test.txt file not found")
#text in the file is:- Hi my name is krishna. I am good, how are you?
```

New file: 'test-es.txt' was created with the translated text



Regular Expressions:

https://www.w3schools.com/python/python_regex.asp

In [8]:
```
import re
st="search inside of this text please"
print('search' in st)
```

```
True
```

In [9]:
```
import re
st="search inside of this text please"
```

```
re.search("this", st)
```

Out[9]:
```
<re.Match object; span=(17, 21), match='this'>
```

> regular expression gives us an output, not just true or false, but an actual object that tells us a little bit more information about 'this'. We can see over here one of the information is the 'span', so where it occurs in the string, it 'occurs at index of 17 and ends at index of 21'.

In [23]:
```
import re
st="search inside of this text please"
a=re.search("this",st)
print(a.span())
print(a.start())
print(a.end())
print(a.group())
```

```
(17, 21)
17
21
this
```

In [24]:
```
import re
st="search inside of this text please"
a=re.search("this",st)
x=a.span()
print(x)
print(a.start())
print(a.end())
print(a.group())


print(x[0])
```

```
(17, 21)
17
21
this
17
```

> I can also do group and `group returns the part of the string where there was the match` . Well, it only gives me that piece of or part of the string that there was the match.

group is really useful when we're trying to do multiple searches. So in our case, we are just doing one search. So we have a pattern that we're trying to match, but there's way that we can do multiple searches within the one line.

In [21]:
```
import re
st="search inside of this text please"
a=re.search("thIs",st)
print(a.group())
```

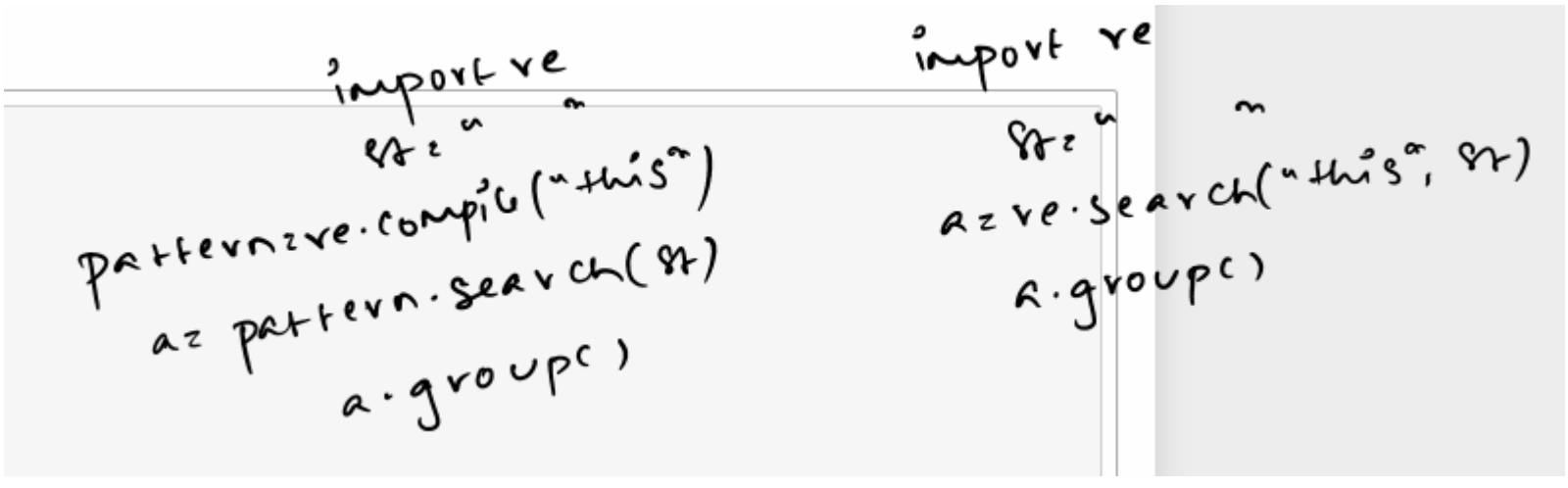
```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-21-7f0774c97daf> in <module>
      2 st="search inside of this text please"
      3 a=re.search("thIs",st)
----> 4 print(a.group())

AttributeError: 'NoneType' object has no attribute 'group'
```

Imp:

if the regular expression doesn't find anything, it's going to return none so it's either going to return a match, object or none.

In [1]:
```
import re
st="search inside of this text please"
pattern=re.compile("this")
a=pattern.search(st)
print(a.group())
```

```
this
```



In [6]:
```
import re
st="search this inside of this text please"
pattern=re.compile("this")
a=pattern.search(st)
b=pattern.findall(st)
c=pattern.fullmatch(st) # Matches against all of the string.
print(a)
print(b)
print(c)
```

```
<re.Match object; span=(7, 11), match='this'>
['this', 'this']
None
```

In [5]:
```
import re
st="search this inside of this text please"
pattern=re.compile("search this inside of this text please")
c=pattern.fullmatch(st) #exact match
print(c)
```

```
<re.Match object; span=(0, 38), match='search this inside of this text please'>
```

In [8]:
```
# match - matches the string and then doesn't care what comes afterwards.
import re
st="search this inside of this text please"
pattern=re.compile("search this inside of this text please!@#")
c=pattern.match(st)
print(c)
```

```
None
```

In [7]:
```
# match - matches the string and then doesn't care what comes afterwards.
import re
st="search this inside of this text please!!!"
pattern=re.compile("search this inside of this text please")
# whatever we are trying to  match should be a part of original string
c=pattern.match(st)
print(c)
```

```
<re.Match object; span=(0, 38), match='search this inside of this text please'>
```

https://regex101.com/

In [5]:
```
#Search the string to see if it starts with "The" and ends with "Spain":
import re
txt = "The rain in Spain Spain Spain"
print(len(txt))
x = re.search("^The.*Spain$", txt)
print(x)
```

```
29
<re.Match object; span=(0, 29), match='The rain in Spain Spain Spain'>
```

In [6]:
```
#Print a list of all matches:

import re

txt = "The rain in Spain"
x = re.findall("ai", txt)
print(x)
```

```
['ai', 'ai']
```

> The list contains the matches in the order they are found. If no matches are found, an empty list is returned:

In [7]:
```
#Return an empty list if no match was found:

import re

txt = "The rain in Spain"
x = re.findall("Portugal", txt)
print(x)
```

```
[]
```

The search() function

The search() function searches the string for a match, and returns a Match object if there is a match.

If there is more than one match, only the first occurrence of the match will be returned:

In [8]:
```
#Search for the first white-space character in the string:

import re

txt = "The rain in Spain"
x = re.search("\s", txt)

print("The first white-space character is located in position:", x.start())
```

```
The first white-space character is located in position: 3
```

In [9]:
```
#If no matches are found, the value None is returned:
#Make a search that returns no match:

import re

txt = "The rain in Spain"
x = re.search("Portugal", txt)
print(x)
```

```
None
```

The split() Function

In [10]:
```
#The split() function returns a list where the string has been split at each match
#Split at each white-space character:

import re

txt = "The rain in Spain"
x = re.split("\s", txt)
print(x)
```

```
['The', 'rain', 'in', 'Spain']
```

In [11]:
```
#You can control the number of occurrences by specifying the maxsplit parameter:
#Split the string only at the first occurrence:

import re

txt = "The rain in Spain"
x = re.split("\s", txt, 1)
print(x)
```

```
['The', 'rain in Spain']
```

The sub() Function

In [12]:
```
#The sub() function replaces the matches with the text of your choice:
#Replace every white-space character with the number 9:

import re

txt = "The rain in Spain"
x = re.sub("\s", "9", txt)
print(x)
```

```
The9rain9in9Spain
```

In [13]:
```
#You can control the number of replacements by specifying the count parameter:
#Replace the first 2 occurrences:

import re

txt = "The rain in Spain"
x = re.sub("\s", "9", txt, 2)
print(x)
```

```
The9rain9in Spain
```

Match Object

A Match Object is an object containing information about the search and the result.

Note: If there is no match, the value None will be returned, instead of the Match Object.

In [14]:
```
#Do a search that will return a Match Object:

import re

txt = "The rain in Spain"
x = re.search("ai", txt)
print(x) #this will print an object
```

```
<re.Match object; span=(5, 7), match='ai'>
```

IMP:`

The Match object has properties and methods used to retrieve information about the search, and the result:

1. .span() returns a tuple containing the start, and end positions of the match.
2. .string returns the string passed into the function
3. .group() returns the part of the string where there was a match

In [37]:
```
#Print the position (start- and end-position) of the first match occurrence.
#The regular expression looks for any words that starts with an upper case "S":

import re

txt = "The rain in Spain"
```

```
x = re.search(r"\bS\w+", txt)
print(x.span())
```

```
(12, 17)
```

In [17]:
```
#Print the string passed into the function:

import re

txt = "The rain in Spain"
x = re.search(r"\bS\w+", txt)
print(x.string)
```

```
The rain in Spain
```

In [18]:
```
#Print the part of the string where there was a match.
#The regular expression looks for any words that starts with an upper case "S":

import re

txt = "The rain in Spain"
x = re.search(r"\bS\w+", txt)
print(x.group())
```

```
Spain
```

Note:

'r' stands for a raw string because in Python something like a slash, backslash has special meaning because remember with back slashes, we can do special things like New Line or tab and 'r' says, hey, this is a raw string Ignore everything that python interpreter might interpret.

I want this to just be a pure string. So you'll see this along with regular expressions.

> the "r" in the beginning is making sure that the string is being treated as a "raw string"

In [25]:
```
import re

pattern=re.compile(r'[a-zA-Z].[a]')
#it's because I am searching for Letter followed by anything followed by 'a'.
txt = "The rain in Spain"
a=pattern.search(txt)
print(a)
print(a.group())
```

```
<re.Match object; span=(12, 15), match='Spa'>
Spa
```

Exercises: Interactive RegEx:- https://regexone.com/

https://emailregex.com/

https://regex101.com/

https://www.w3schools.com/python/python_regex.asp

(^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+.[a-zA-Z0-9-.]+$)

^ ---> Starts with

[a-zA-Z] ---> Returns a match for any character alphabetically between a and z, lower case OR upper case

[0-9] ---> Returns a match for any digit between 0 and 9

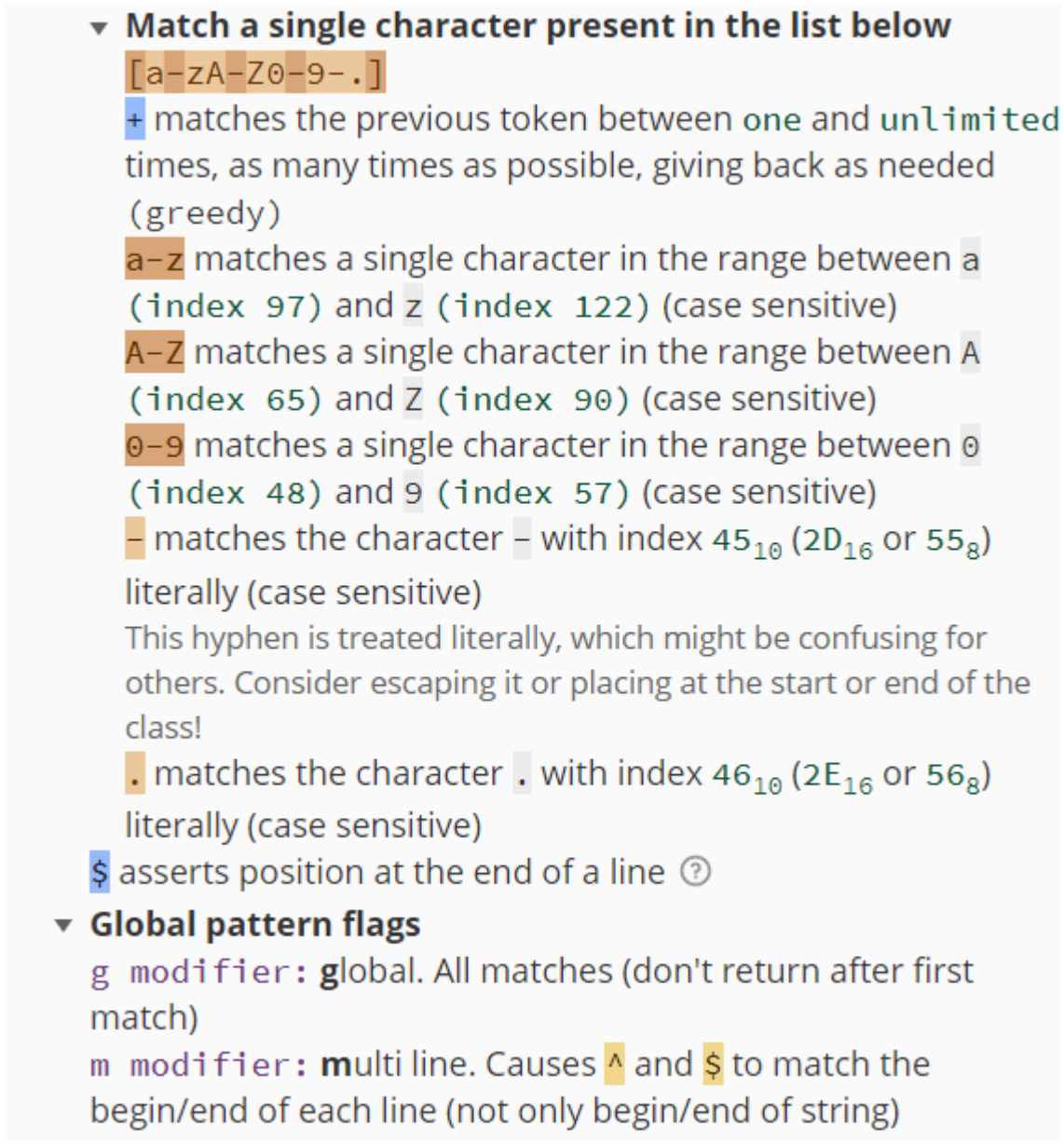
$ ---> Ends with

EXPLANATION

- " (^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0- " gm
  9-.]+$)
  - **1st Capturing Group**
    (^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+
    $)
    ^ asserts position at start of a line
    - **Match a single character present in the list below**
      [a-zA-Z0-9_.+-]
      + matches the previous token between one and unlimited times, as many times as possible, giving back as needed (greedy)
      a-z matches a single character in the range between a (index 97) and z (index 122) (case sensitive)
      A-Z matches a single character in the range between A (index 65) and Z (index 90) (case sensitive)
      0-9 matches a single character in the range between 0 (index 48) and 9 (index 57) (case sensitive)
      - _.+- matches a single character in the list _.+- (case sensitive)

    @ matches the character @ with index 64_{10} (40_{16} or 100_{8}) literally (case sensitive)
    - **Match a single character present in the list below**
      [a-zA-Z0-9-]
      + matches the previous token between one and unlimited times, as many times as possible, giving back as needed (greedy)
      a-z matches a single character in the range between a (index 97) and z (index 122) (case sensitive)
      A-Z matches a single character in the range between A (index 65) and Z (index 90) (case sensitive)
      0-9 matches a single character in the range between 0 (index 48) and 9 (index 57) (case sensitive)
      - matches the character - with index 45_{10} ($2D_{16}$ or 55_{8}) literally (case sensitive)

    \. matches the character . with index 46_{10} ($2E_{16}$ or 56_{8}) literally (case sensitive)



`(^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$)`

1. `^[a-zA-Z0-9_.+-]` --- I see that it's asserting a position '^' at the start of the line, so the start of the line,

we want to make sure that it captures this [a-zA-Z0-9_.+-] So it matches a single character present in a set, so this email can be anything, it could be letters, it could be numbers, and then finally it could be underscore, it could have a dot in there. It could have plus, it could have minus.

2. `+` --- then plus is a quantifier. So it matches between one and unlimited times. So this string can be as long as I want and it's still going to work. It's going to accept plus minus.It's going to accept underscore but it's not going to accept an exclamation mark, because it's not in here.
3. Well, the next part is @.It's literally the @. We have to make sure we have it and then after the @, again,
4. once again, we have the characters that we can use with a plus at the end to say, hey, you can have as many of these as you want.
5. `\.` --- Next, we have a special one, matches the character Dot. If instead I didn't have this slash, this could be any character. This is still a matching string without a dot.So, again, special character.
6. After the dot, once again, we have anything.

That we want within these square brackets, then as long as we want and 7. `$` --- then finally this dollar sign says, hey, it has to end with this, this is the end of the line.

In [27]:
```
import re

pattern=re.compile(r'(^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$)')
txt='krish@yopmail.com'
a=pattern.search(txt)
print(a)
print(a.group())
txt1='krish.com'
b=pattern.search(txt1)
print(b)
print(b.group())
```

```
<re.Match object; span=(0, 17), match='krish@yopmail.com'>
krish@yopmail.com
None
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-27-a58e7b6f4b79> in <module>
      9 b=pattern.search(txt1)
     10 print(b)
---> 11 print(b.group())

AttributeError: 'NoneType' object has no attribute 'group'
```

Password Validation:

Atleast 9 characters long

Contains any sort of letters, numbers, $@#%

Has to end with number --- https://regex101.com/

In [37]:
```
import re

pattern=re.compile(r'(^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$)')
txt='krish@yopmail.com'
passpattern=re.compile(r"[A-Za-z0-9 $@#%]{8,}\d")
password='krish5%$%7'
a=pattern.search(txt)
b=passpattern.search(password)
print(a)
print(a.group())
print(b)
print(b.group())
```

```
<re.Match object; span=(0, 17), match='krish@yopmail.com'>
krish@yopmail.com
<re.Match object; span=(0, 10), match='krish5%$%7'>
krish5%$%7
```

In [14]:
```
"""Atleast 9 characters long
Contains any sort of letters, numbers, $@#%
Has to end with number"""
import re
passpattern=re.compile(r'[A-Za-z0-9 $@#%]{8,}\d')
#---[A-Za-z0-9 $@#%]: should be atleast 8
password='krishna5'
b=passpattern.search(password)
print(b)
```

```
None
```

In [28]:
```
"""Atleast 9 characters long
Contains any sort of letters, numbers, $@#%
Has to end with number"""
import re
passpattern=re.compile(r'[A-Za-z0-9 $@#%]{8,}\d')
#---[A-Za-z0-9 $@#%]: should be atleast 8
password='krishnaa5'
b=passpattern.search(password)
print(b)
```

```
<re.Match object; span=(0, 9), match='krishnaa5'>
```

In [7]:
```
"""Atleast 8 characters long
Contains any sort of letters, numbers, $@#%
Has to end with number"""
import re
passpattern=re.compile(r'([A-Za-z0-9$%#@]{7,}[0-9])')
password='kris5%$%4'
b=passpattern.search(password)
print(b)
```

```
<re.Match object; span=(0, 9), match='kris5%$%4'>
```

In [9]:
```
"""Atleast 9 characters long
Should start with Capital letter
Contains any sort of letters, numbers, $@#%
Has to end with number"""
import re
passpattern=re.compile(r'[A-Z][A-Za-z0-9 $@#%]{8,}\d')
#---[A-Za-z0-9 $@#%]: should be atleast 8
password='Kkrishnaa5'
b=passpattern.search(password)
print(b)
```

```
<re.Match object; span=(0, 10), match='Kkrishnaa5'>
```

To maintain the privacy of individuals, especially when dealing with Personal Health Infor1mation (PHI) or Personal Identifiable Information (PII), it's important to identify and mask these types of data in text. Here's a simple Python script that uses regular expressions to identify PHI and PII.

In this example, I'm considering PHI as "Name", "Age" and "Phone Number" and PII as "Username" and "Password". You can add more patterns to the regular expressions to match more complex PHI and PII.

In [1]:
```
import re

def mask_phi_pii(text):
    # Define patterns for PHI and PII
    phi_patterns = {
        'name': r'\b[A-Z][a-z]+\s[A-Z][a-z]+\b',
        'age': r'\b\d{1,3}(?:-?\d{1,2})?\b',
        'phone': r'\b\d{10}\b'
    }

    pii_patterns = {
        'username': r'\b[a-z]{1,10}\b',
        'password': r'\b[A-Za-z0-9]{1,15}\b'
    }
```

```
    # Mask PHI
    for pattern in phi_patterns.values():
        text = re.sub(pattern, 'XXXX', text)

    # Mask PII
    for pattern in pii_patterns.values():
        text = re.sub(pattern, 'XXXX', text)

    return text

text = "John Doe is 30 years old. His phone number is 1234567890. His username is jdoe and password is Password123."
print(mask_phi_pii(text))
```

```
XXXX XXXX XXXX XXXX XXXX. XXXX XXXX XXXX XXXX XXXX. XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX.
```

Testing in Python:

"D:\Edu\Python_ZTM\TestingPython"

https://docs.python.org/3/library/unittest.html

test.py | main.py | test2.py | test3.py | test4.py | Test5.py | test6.py | test7.py | test

```
def do_stuff(num=0):
    try:
        if num: #checking that the number is other than None because if num is None it me
            return int(num)+5
        else:
            return 'Please enter number'
    except ValueError as err:
        return err
    #we're returning an error. It's actually an instance of the ValueError class.

#print(bool(None)) ---> False
```

test.py | main.py | test2.py | test3.py | test4.py | Test5.py | test6.py | test7.py | test8.py | test9.py

```
import unittest
import main #file/script which needs to be tested

"""we inherit inside of this class what unit test gives us, which is a test.
So this is just a standard way to work with unit test, so we're inheriting whatever
unittest testcases and now we have that functionality available in our test main class."""
"""assert equal is something that we get when we inherit from unit test and assert equal says, hey,
assert Or make sure that these two parameters are equal in our case, I want to test that the result is equal
to, well, in our case, 10+5."""

class TestMain(unittest.TestCase):
    def test_do_stuff(self):
        test_param=10
        result=main.do_stuff(test_param)
        self.assertEqual(result,15)
unittest.main()
```

test.py | main.py | test2.py | test3.py | test4.py | T

```
import unittest
import main #file/script which needs to be tested
class TestMain(unittest.TestCase):
    def test_do_stuff(self):
        test_param=10
        result=main.do_stuff(test_param)
        self.assertEqual(result,10)

unittest.main()
```

test.py | main.py | test2.py | test3.py | test4.py | Test5.py | test6.py | test7.py | test8.py | test9.py

```python
import unittest
import main  # file/script which needs to be tested
class TestMain(unittest.TestCase):
    def test_do_stuff(self):
        test_param = 10
        result = main.do_stuff(test_param)
        self.assertEqual(result, 15)
# I'm asserting that hey I want to make sure that the result and the value are equal
    def test_do_stuff2(self):
        test_param = 'csdv'
        result = main.do_stuff(test_param)
        self.assertEqual(result, ValueError)
#I'm asserting that hey I want to make sure that the result and the value error are equal, which will fail.

unittest.main()
#AssertionError: ValueError("invalid literal for int() with base 10: 'csdv'") != <class 'ValueError'>
"""we get this AssertionError (ValueError!= <class 'ValueError'>) because in main.py - we're returning an err.
and err is actually an instance of the ValueError class."""
```

test.py | main.py | test2.py | test3.py | test4.py | Test5.py | test6.py | test7.py | test8.py | test9.py

```python
import unittest
import main  # file/script which needs to be tested


class TestMain(unittest.TestCase):
    def test_do_stuff(self):
        test_param = 10
        result = main.do_stuff(test_param)
        self.assertEqual(result, 15)
    def test_do_stuff2(self):
        test_param = 'csdv'
        result = main.do_stuff(test_param)
        self.assertTrue(isinstance(result, ValueError))
#I'm asserting that hey I want to make sure that the result (err) is an instance of ValueError class.

unittest.main()

```

test.py | main.py | test2.py | test3.py | test4.py | Test5.py | test6.py | test7.py | test8.py | test9.py

```python
#https://docs.python.org/3/library/unittest.html
import unittest
import main  # file/script which needs to be tested


class TestMain(unittest.TestCase):
    def test_do_stuff(self):
        test_param = 10
        result = main.do_stuff(test_param)
        self.assertEqual(result, 15)
    def test_do_stuff2(self):
        test_param = 'csdv'
        result = main.do_stuff(test_param)
        self.assertIsInstance(result, ValueError)
#I'm asserting that hey I want to make sure that the result (err) is an instance of ValueError class.

unittest.main()

```

test.py | main.py | test2.py | test3.py | test4.py | Test5.py | test6.py | test7.py | test8

```python
#https://docs.python.org/3/library/unittest.html
import ...


class TestMain(unittest.TestCase):
    def test_do_stuff(self):
        test_param = 10
        result = main.do_stuff(test_param)
        self.assertEqual(result, 15)
    def test_do_stuff2(self):
        test_param = 'csdv'
        result = main.do_stuff(test_param)
        self.assertIsInstance(result, ValueError)
    def test_do_stuff3(self):
        test_param = None
        result = main.do_stuff(test_param)
        self.assertIsInstance(result, ValueError)
#AssertionError: 'Please enter number' is not an instance of <class 'ValueError'>
unittest.main()
```

test.py | main.py | test2.py | test3.py | test4.py | Test5.py | test6.py | test7.py | test8.py

```python
#https://docs.python.org/3/library/unittest.html
import ...


class TestMain(unittest.TestCase):
    def test_do_stuff(self):
        test_param = 10
        result = main.do_stuff(test_param)
        self.assertEqual(result, 15)
    def test_do_stuff2(self):
        test_param = 'csdv'
        result = main.do_stuff(test_param)
        self.assertIsInstance(result, ValueError)
    def test_do_stuff3(self):
        test_param = None
        result = main.do_stuff(test_param)
        self.assertEqual(result, 'Please enter number')
    def test_do_stuff4(self):
        test_param = ''
        result = main.do_stuff(test_param)
        self.assertEqual(result, 'Please enter number')

unittest.main()
#unit test mean simply says, hey, just run all the tests over here.
```

test.py | main.py | test2.py | test3.py | test4.py | Test5.py | test6.py | test7.py | test8.py | test9.py

```python
#https://docs.python.org/3/library/unittest.html
import unittest
import main  # file/script which needs to be tested

class TestMain(unittest.TestCase):
    def test_do_stuff(self):
        test_param = 10
        result = main.do_stuff(test_param)
        self.assertEqual(result, 15)
    def test_do_stuff2(self):
        test_param = 'csdv'
        result = main.do_stuff(test_param)
        self.assertIsInstance(result, ValueError)
    def test_do_stuff3(self):
        test_param = None
        result = main.do_stuff(test_param)
        self.assertEqual(result, 'Please enter number')
    def test_do_stuff4(self):
        test_param = ''
        result = main.do_stuff(test_param)
        self.assertEqual(result, 'Please enter number')

#Now I make sure that this is a test file and I'm  going to run this code only if it's the main file being run.
if __name__ == '__main__':
    unittest.main()
```

Here we are using script instead of main:

test.py | test3.py | test4.py | Test5.py | test6.py | script.py

```python
def do_stuff(num=0):
    try:
        if num:
            return int(num)+5
        else:
            return 'Please enter number'
    except ValueError as err:
        return err
```

test.py | test3.py | test4.py | Test5.py | test6.py | script.py | test7.py | script2test.py | test8.py | scripttest.py

```python
#https://docs.python.org/3/library/unittest.html
import unittest
import script #file/script which needs to be tested

class TestMain(unittest.TestCase):
    def test_do_stuff(self):
        test_param = 10
        result = script.do_stuff(test_param)
        self.assertEqual(result, 15)
    def test_do_stuff2(self):
        test_param = 'csdv'
        result = script.do_stuff(test_param)
        self.assertIsInstance(result, ValueError)
    def test_do_stuff3(self):
        test_param = None
        result = script.do_stuff(test_param)
        self.assertEqual(result, 'Please enter number')
    def test_do_stuff4(self):
        test_param = ''
        result = script.do_stuff(test_param)
        self.assertEqual(result, 'Please enter number')

#Now I make sure that this is a test file and I'm  going to run this code only if it's the main file being run.
if __name__ == '__main__':
    unittest.main()
#unit test mean simply says, hey, just run all the tests over here.
```

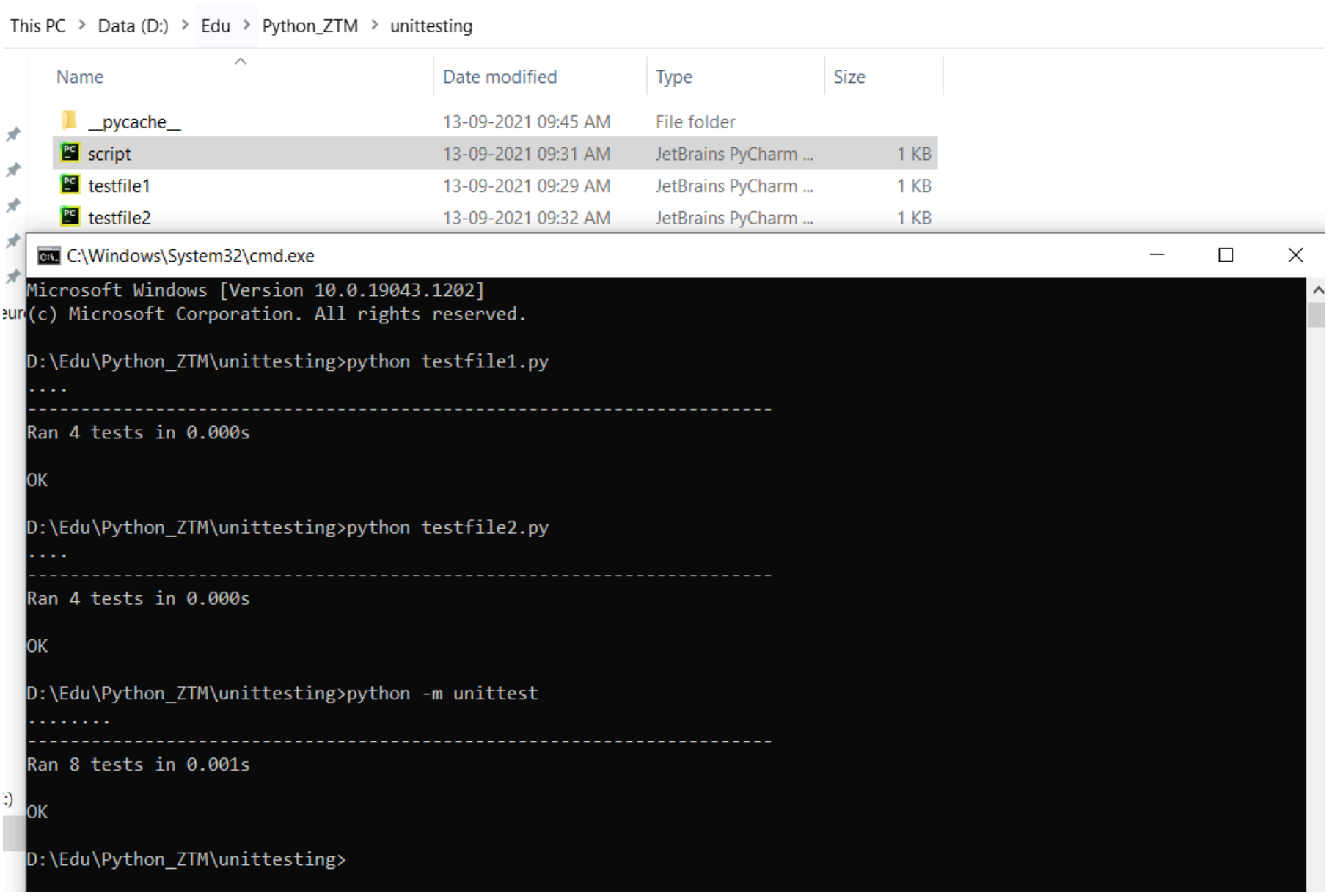
So far we've just ran it from the command line as a single file, and that has been working.

C:\Windows\System32\cmd.exe

```
Microsoft Windows [Version 10.0.19043.1202]
(c) Microsoft Corporation. All rights reserved.

D:\Edu\Python_ZTM\TestingPython>python scriptt.py
....
----------------------------------------------------------------------
Ran 4 tests in 0.001s

OK
```

But usually you have more than one file and you have each module. Tested with different functions, right? So ideally, you want to run all these tests together.

D:\Edu\Python_ZTM\unittesting



To get more details:You see which tests we've run and which ones are OK, which ones have failed.

```
D:\Edu\Python_ZTM\unittesting>python -m unittest -v
test_do_stuff (testfile1.TestMain) ... ok
test_do_stuff2 (testfile1.TestMain) ... ok
test_do_stuff3 (testfile1.TestMain) ... ok
test_do_stuff4 (testfile1.TestMain) ... ok
test_do_stuff (testfile2.TestMain) ... ok
test_do_stuff2 (testfile2.TestMain) ... ok
test_do_stuff3 (testfile2.TestMain) ... ok
test_do_stuff4 (testfile2.TestMain) ... ok

----------------------------------------------------------------------
Ran 8 tests in 0.002s

OK
```

Scriptfile:

```python
def do_stuff(num=0):
    try:
        if num:
            return int(num)+5
        else:
            return 'Please enter number'
    except ValueError as err:
        return err
```

TestFile:

rndgame.py | rndgametest.py | script.py | testfile1.py | testfile2.py

```python
import unittest
import script  # file/script which needs to be tested
class TestMain(unittest.TestCase):
    def setUp(self):
        print("hey!!! going to start the testing - And setup allows us to run a piece of code. That sets up before "
              "each call of the test")
    def test_do_stuff(self):
        """function comments"""
        test_param = 10
        result = script.do_stuff(test_param)
        self.assertEqual(result, 15)
    def test_do_stuff2(self):
        test_param = 'csdv'
        result = script.do_stuff(test_param)
        self.assertIsInstance(result, ValueError)
    def test_do_stuff3(self):
        test_param = None
        result = script.do_stuff(test_param)
        self.assertEqual(result, 'Please enter number')
    def test_do_stuff4(self):
        test_param = ''
        result = script.do_stuff(test_param)
        self.assertEqual(result, 'Please enter number')
    def tearDown(self):
        print("cleaning up - As the name suggests, we run it at the end of each method that we call so in here, "
              "let's say cleaning")
# Now I make sure that this is a test file and I'm  going to run this code only if it's the main file being run.
if __name__ == '__main__':
    unittest.main()
# unit test mean simply says, hey, just run all the tests over here.
```

Output:

```
D:\Edu\Python_ZTM\unittesting>python -m unittest -v
test_do_stuff (testfile1.TestMain)
function comments ... hey!!! going to start the testing - And setup allows us to run a piece of code. That sets up before each call of the test
cleaning up - As the name suggests, we run it at the end of each method that we call so in here, let's say cleaning
ok
test_do_stuff2 (testfile1.TestMain) ... hey!!! going to start the testing - And setup allows us to run a piece of code. That sets up before each call of the test
cleaning up - As the name suggests, we run it at the end of each method that we call so in here, let's say cleaning
ok
test_do_stuff3 (testfile1.TestMain) ... hey!!! going to start the testing - And setup allows us to run a piece of code. That sets up before each call of the test
cleaning up - As the name suggests, we run it at the end of each method that we call so in here, let's say cleaning
ok
test_do_stuff4 (testfile1.TestMain) ... hey!!! going to start the testing - And setup allows us to run a piece of code. That sets up before each call of the test
cleaning up - As the name suggests, we run it at the end of each method that we call so in here, let's say cleaning
ok
test_do_stuff (testfile2.TestMain) ... ok
test_do_stuff2 (testfile2.TestMain) ... ok
test_do_stuff3 (testfile2.TestMain) ... ok
test_do_stuff4 (testfile2.TestMain) ... ok

----------------------------------------------------------------------
Ran 8 tests in 0.008s

OK

D:\Edu\Python_ZTM\unittesting>
```

UnitTesting Exercise:

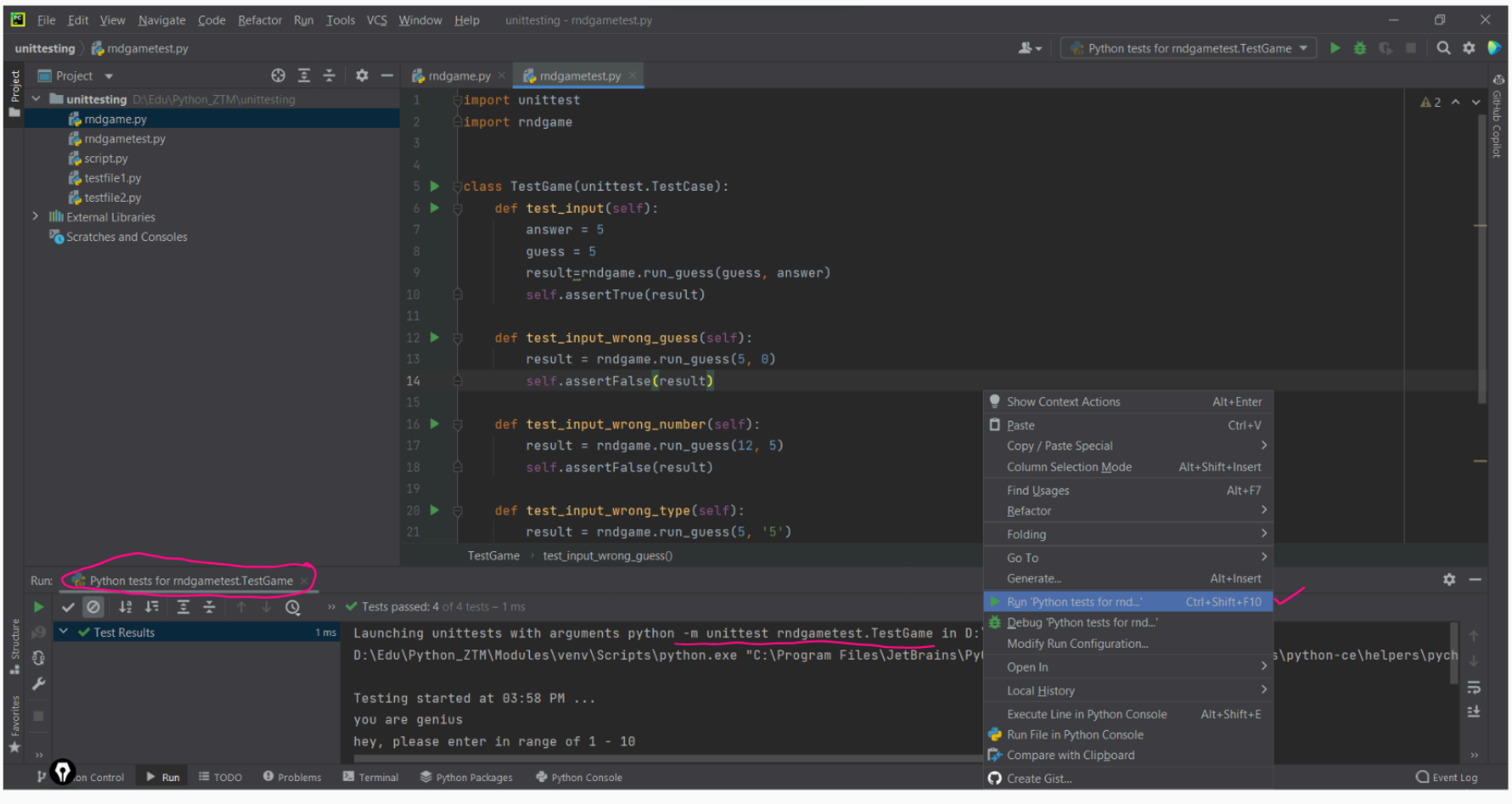
rndgame.py rndgametest.py

```python
from random import randint
# generate a number from 1 to 10
def run_guess(guess, answer):
    if 0 < guess < 11:
        if guess == answer:
            print("you are genius")
            return True
    else:
        print("hey, please enter in range of 1 - 10 ")


if __name__ == '__main__':
    answer = randint(1, 10)
    # input from user
    # check that the input is between 1 to 10
    while True:
        try:
            print(answer)
            guess = int(input("guess a number in range of 1 - 10: "))
            # if guess > 0 and guess < 11:
            if run_guess(guess, answer):
                break
        except ValueError:
            print("please enter a number")
```

rndgame.py rndgametest.py

```python
import unittest
import rndgame


class TestGame(unittest.TestCase):
    def test_input(self):
        answer = 5
        guess = 5
        result=rndgame.run_guess(guess, answer)
        self.assertTrue(result)

    def test_input_wrong_guess(self):
        result = rndgame.run_guess(5, 0)
        self.assertFalse(result)

    def test_input_wrong_number(self):
        result = rndgame.run_guess(12, 5)
        self.assertFalse(result)

    def test_input_wrong_type(self):
        result = rndgame.run_guess(5, '5')
        self.assertFalse(result)


if __name__ == '__main__':
    unittest.main()
```

Performing unittest from the pycharm IDE:


File Edit View Navigate Code Refactor Run Tools VCS Window Help unittesting - rndgametest.py
unittesting rndgametest.py
Python tests for rndgametest.TestGame
Project
unittesting D:\Edu\Python_ZTM\unittesting
rndgame.py
rndgametest.py
script.py
testfile1.py
testfile2.py
External Libraries
Scratches and Consoles
rndgame.py
rndgametest.py
import unittest
import rndgame
class TestGame(unittest.TestCase):
def test_input(self):
answer = 5
guess = 5
result=rndgame.run_guess(guess, answer)
self.assertTrue(result)
def test_input_wrong_guess(self):
result = rndgame.run_guess(5, 0)
self.assertFalse(result)
def test_input_wrong_number(self):
result = rndgame.run_guess(12, 5)
self.assertFalse(result)
def test_input_wrong_type(self):
result = rndgame.run_guess(5, '5')
TestGame test_input_wrong_guess()
Show Context Actions Alt+Enter
Paste Ctrl+V
Copy / Paste Special
Column Selection Mode Alt+Shift+Insert
Find Usages Alt+F7
Refactor
Folding
Go To
Generate... Alt+Insert
Run 'Python tests for rnd...' Ctrl+Shift+F10
Debug 'Python tests for rnd...'
Modify Run Configuration...
Open In
Local History
Execute Line in Python Console Alt+Shift+E
Run File in Python Console
Compare with Clipboard
Create Gist...
Run: Python tests for rndgametest.TestGame
Tests passed: 4 of 4 tests – 1 ms
Test Results 1 ms
Launching unittests with arguments python -m unittest rndgametest.TestGame in D:
D:\Edu\Python_ZTM\Modules\venv\Scripts\python.exe "C:\Program Files\JetBrains\Py
Testing started at 03:58 PM ...
you are genius
hey, please enter in range of 1 - 10
Run TODO Problems Terminal Python Packages Python Console Event Log

In [ ]: