Set{}

evens = {2,44,24,62,78}

Sets are very similar to dictionaries

, only that they are consisted of single keys, rather than pairs of key and value.

they are mutable(you can add items or remove items)

The set’s item must be immutable.

Since list is a mutable type, if we try to create a set that contains a list we will get an exception:

s = {list()}

>> TypeError: unhashable type: 'list'

s.update([11, 22], (55,), ('mimi',))

print(s)

update() unpacks the given arguments and adds them to the set.

Note that if ‘mimi’ would have been given outside a dedicated tuple,

‘m’ and ‘i’ letters would be added.

to delte

s.discard('Avi')

s.remove('Itzik')

s.clear() - delete every thing

s.pop -delete the first element

there are a few functions in set section ' for merging two sets ' find defferents ...

pytest

run test with -

pytest 'specific route '

pytest 'specific route::specific func'

Adding @pytest.mark.skip to the test means that Pytest

will skip it and the test will not be executed.

In case we want a test to be executed but not to be count as part of the failed or

passed tests we can use xfail marker by adding @pytest.mark.xfail.

@pytest.mark.skip to skip test or group tests

@pytest.mark.skip(reason="Test skipped during development")

@pytest.mark.skipif(condition=True, reason="Test skipped due to a specific condition")

@pytest.mark.sanity

pytest test\_sum.py -m sanity

run only selected tests

create a pytest.ini file in your root folder, and add your own markers.

how to create marks ?

in pytest.ini for example of adding low3

[pytest]

markers =

sanity: minimal amount of basic tests to run

low3: mark tests where the result of calling sum is lower than 3

add @pytest.mark.low3 before the your test

run it from terminal with pytest touteofthestfile -m low3

example of parametrized

@pytest.mark.parametrize("num1, num2", [(1, 2),(2,0)])

def test\_sum\_low\_values(num1,num2):

assert num1+num2 < 3 , f"Expected the sum to be lower than 3, but got {num1+num2}"

fixtures - init somthing that we have in our test

@pytest.fixture

def student():

return {

"name": "Shushu",

"grades": [70, 33, 93, 47]

}

Decorators

Decorators are functions that allows you run some code before and after a function.

def my\_decorator(some\_func):

def wrapper():

print("1. before executing some\_func()")

some\_func() # here we execute the function

print("2. after executing some\_func()")

return wrapper

def just\_print():

print("\*\* some function that uses the decorator")

my\_func = my\_decorator(just\_print)

my\_func()

--------------------------------------second way

@my\_decorator

def just\_print():

print("\*\* some function that uses the decorator")

just\_print()

\*args- for songal parameter

\*\*kwargs - for keyword parameter - name="ofer"

Default parameter values are evaluated when the function definition is executed,

not each time the function is called ( like the example of default list parameter)

def add\_to\_list(value, nums=[]):

nums.append(value)

return nums

print(add\_to\_list(1))

print(add\_to\_list(2))

[1]

[2]

But instead we get:

[1]

[1, 2]

how to fix ?

def add\_to\_list(value, nums=None):

nums = nums or []

nums.append(value)

return nums

\*args , \*\*kwargs

def more\_is\_better(\*args):

print(args)

more\_is\_better(1, 2, 3, 4)

every thing that we will send under \*args will get a Tuple() with the paras

more\_is\_better([1,2,3])

We will get a tuple of 1 item - the list.

if i send this :more\_is\_better(1, "hello", True,[1,2,3], 1,2,3)

i'll get : (1, 'hello', True, [1, 2, 3], 1, 2, 3)

We can also define unlimited keyword arguments for a function as a dictionary:

def print\_suitcase(\*\*suitcase):

for item,ammount in suitcase.items():

print(item, ammount)

for example :

print\_suitcase(shirts=3)

print\_suitcase(shirts=4, shoes=2, watermelon=16)

another example for \*\*kwargs

def print\_suitcase(\*\*suitcase):

for item, ammount in suitcase.items():

print(item, ammount)

items = {

"shirts": 2,

"shoes": 8,

"watermelon": 100

}

print\_suitcase(\*\*items)

example of combination :

def all\_arguments(num, \*args, \*\*kwargs):

print("regular argument", num)

print("args", args)

print("kwargs", kwargs)

all\_arguments(3, "arg1", "arg2", arg3=3, arg4=[1,2])

the output will be :

regular argument 3

args ('arg1', 'arg2')

kwargs {'arg3': 3, 'arg4': [1, 2]}

function is an object and we can know inforamtion about it here is an example :

def hello(name = "Margo"):

"""this is a greeting function"""

print(f'Hello {name.capitalize()}!')

Let’s examine some cool properties:

print("Documentation: ", hello.\_\_doc\_\_)

print("Name: ", hello.\_\_name\_\_)

print("Code at: ", hello.\_\_code\_\_)

print("defaults : ", hello.\_\_defaults\_\_)

we will get this outhput :

Documentation: this is a greeting function

Name: hello

Code at: <code object hello at 0x013B5290, file "c:/Users/User/Desktop/DevOps/atech-devops-nov2023/python\_katas/kata\_1/file.py", line 15>

defaults : ('Margo',)

Global varibale

If we want to update a global variable from within a function we will have to specify the global keyword:

g = 'I am global'

def func():

global g

g = 'overrideing global'

print('func: ', g)

func()

print('outside: ', g)

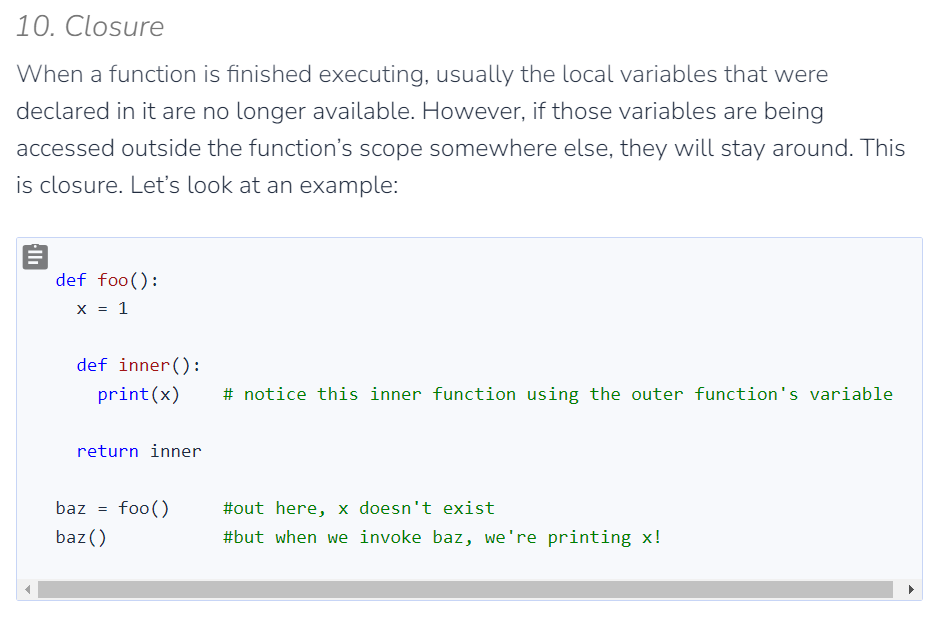
output will be :

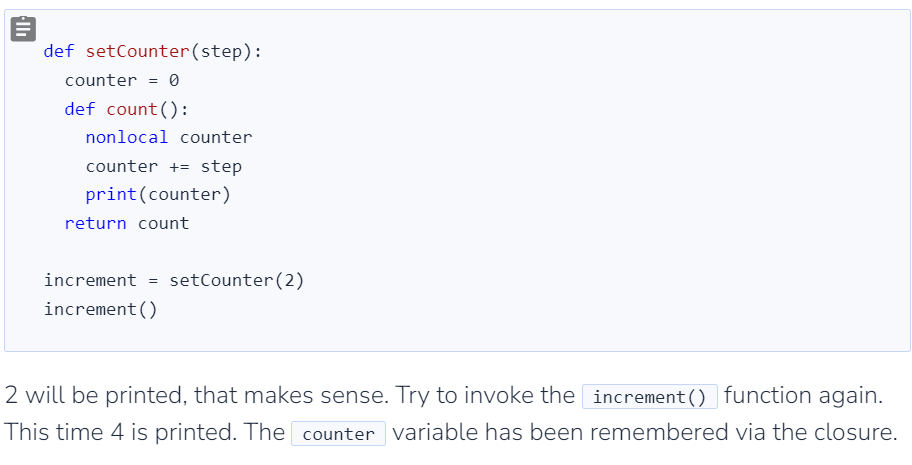
func: overrideing global

outside: overrideing global

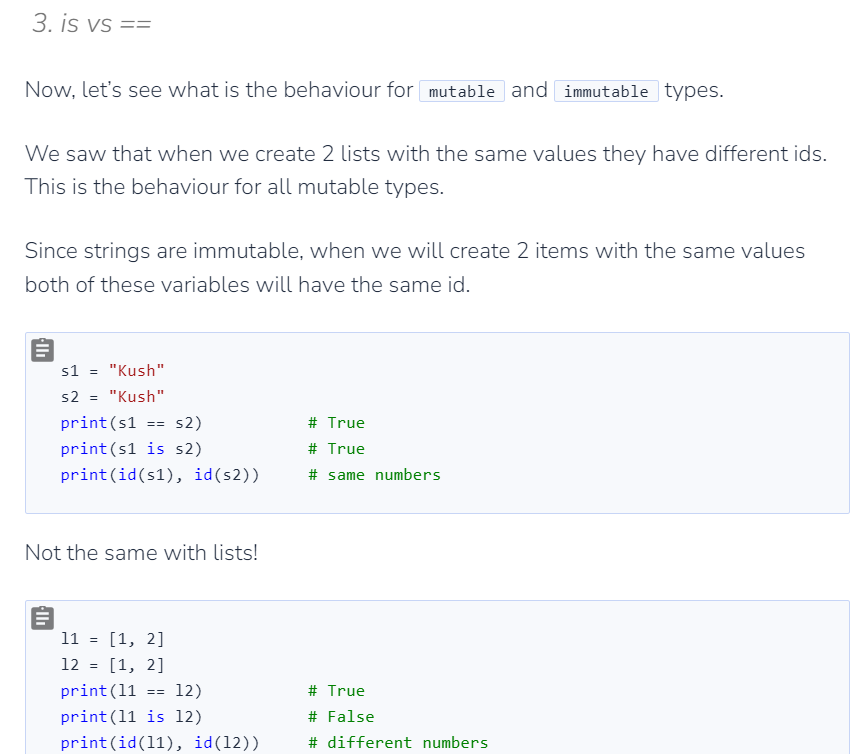
In case of function in function and we want to chnge the variable of the main function we should use nonlocal:

Closure

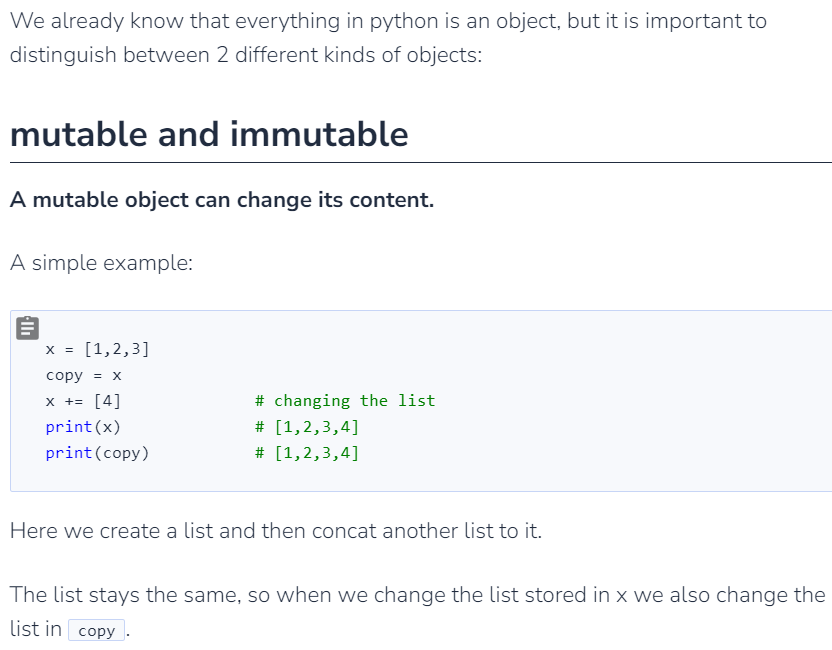
the output we will 1 , I can use the baz() func to get the value of x every time and every where after the baz=foo() excuted.

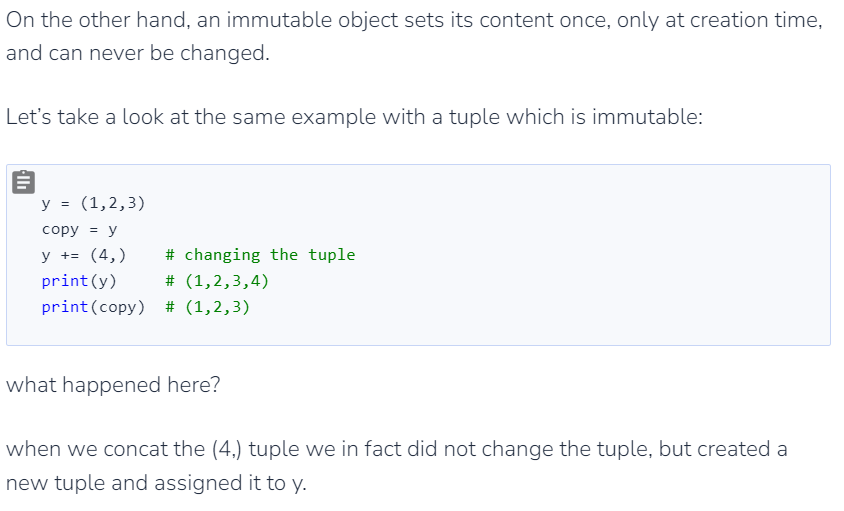


Is vs ==



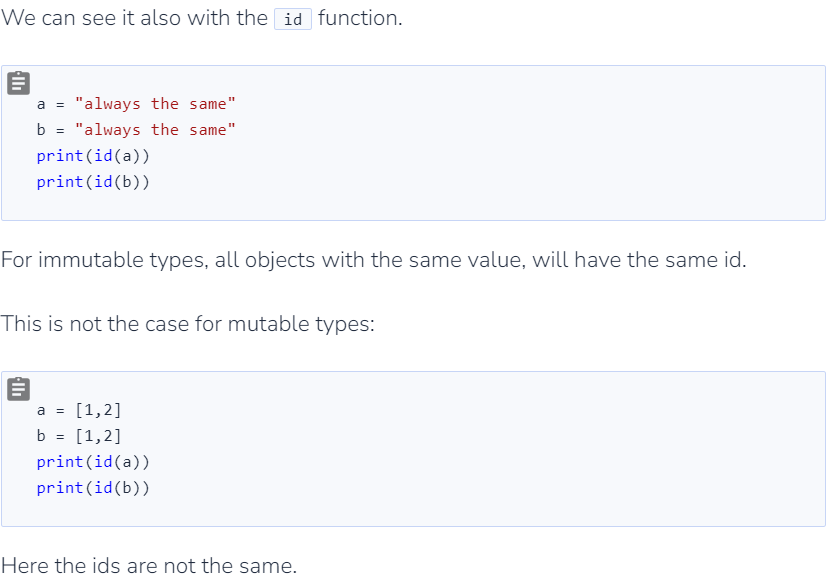
Mutable vs Immutable





For Summarize:

If it is a mutable object it will changed in both variables, if the object is immutable it will only change in the same object only .



reading with files

# Open the file for reading

with open('test.txt', 'r') as my\_file:

# Initialize an empty list to store the lines

lines\_list = []

# Read and append each line to the list

for line in my\_file:

lines\_list.append(line.strip()) # strip() is used to remove newline characters

# Print the list of lines

print(lines\_list)

import json

# Open the file for reading

with open('students.json', 'r') as file:

# Load data from the JSON file

data = json.load(file)

# Display the data

print(data)

------------------------------------------------------------------------------------

import json

with open('students.json', 'r') as file:

data = json.load(file)

# Get the list of students

students\_list = data["students"]

# Print the names of students

for student in students\_list:

print(student["name"])

import xml.etree.ElementTree as ET

# Replace 'students.xml' with the name of your XML file

xml\_file\_path = 'students.xml'

# Parse XML data from the file

tree = ET.parse(xml\_file\_path)

root = tree.getroot()

# Get the list of students

students\_list = root.findall('student')

# Print the names of students

for student in students\_list:

name = student.find('name').text

print(name)