# 1. A

Гордият човек е като локва - хвърли в нея камък и ще опръска всичко наоколо с мръсотия. А смиреният е като море - ще погълне безследно всеки камък и даже кръгове по водата няма да се образуват.

Дядо Добри

```
 self.is_on = not self.is_on
# change "on" to "off" or vice versa

elif idx == int(len(word) // 2) # int can be omitted, because len(word)
is integer // 2 will return the type of len(word)

result = [
    f"You have {len(self.workers)} workers",
    f"----- {len(info['Keeper'])} Keepers:",
    *info["Keeper"],
    f"----- {len(info['Caretaker'])} Caretakers:",
    *info["Caretaker"],
    f"----- {len(info['Vet'])} Vets:",
    *info["Vet"]
]
```

## Referenced list, ????

```
biggest_sum = -float("inf")
```

Dunder - double underscore ???

**round_half_correctly.py**

**float problems**

```
a = 7.55
b = 240 - 232.45
print(b)  # 7.550000000000011
print(f"{a:.1f}")  # 7.5
print(f"{b:.1f}")  # 7.6
print(f"{240 - 232.45:.1f}")  # 7.6
```

snake_case
**P**ascal**C**ase
camel**C**ase
Mangling
**or** has a lower priority than **and**
**and** has a lower priority than **not**

Parameters
Arguments
Attributes

```
print(f"Milk: {', '.join(str(x) for x in cups) or 'empty'}")

@staticmethod
def find_object(collection: list, attribute: str, value: str):
    for obj in collection:
        if str(getattr(obj, attribute)) == value:
            return obj
```

```
print(isinstance('a', int))  # False
print(isinstance(5, int))  # True

from functools import reduce
map_functions = {
    '*': lambda x: reduce(lambda a, b: a * b, x),
    '/': lambda x: reduce(lambda a, b: a / b, x),
# '/': lambda x: reduce(lambda a, b: a + b if a == 0 or b == 0 else a / b, x),
    '+': lambda x: reduce(lambda a, b: a + b, x),
    '-': lambda x: reduce(lambda a, b: a - b, x),
}  02_expression_evaluator_a.py in 03_Stacks_Queues_Tuples_and_Sets_Exercise
```

summation_pairs.py
W:\1_Python\1-Training\1_Projects\1st_Project\03_Advanced
\02_Tuples_and_Sets\Lab\6_summation_pairs.py

```
command = "Replace-{file_name}-{old_string}-{new_string}"
action, *info, last = command.split('-')

a, b, c = 2, '*', 3
print(eval(f"{a}{b}{c}")) # 6
```
eval is slow and info inside eval could be stolen from hacker

```
if ("Doll" and "Wooden") in crafted: # Wrong!!!
if "Doll" in crafted and "Wooden" in crafted: # Correct!!!

for i in range(0, 2, 0.5):
    print(i)
```
**TypeError: 'float' object cannot be interpreted as an integer**
```
for i in range(0, 5, int(0.5)):
    print(i, end='   ')
```
**ValueError: range() arg 3** (int(0.5)) **must not be zero**

Python is a **dynamic** language
Variables are **not** directly associated with
any particular value type
Any variable can be **assigned** (and **re-assigned**)
values of all types
```
x = 2.45         # float
y = 5            # int
w = x // 2       # float - take the class of x
print(type(w))   # class 'float'
w = y // 2       # int - changing to the class of y
print(type(w))   # class 'int'
```
Python **integers** are **immutable**
Python **floats** are **immutable**
Python **strings** are **immutable**
This means that once a string is created,

```
it is not possible to modify it
name = 'George'
name[0] = 'P' # Error не може да променим G
print(name)    # George
name = 'Ime'   # заделя друго място в паметта
различно  от мястото за George
print(name)    # Ime
name = 4       # заделя трето място в паметта
print(name)    # 4
string interpolation are string literals(буквален)
that allow embedded(вградени) expressions

result = first_number // second_number # integer division
result = first_number % second_number   # modular division
result = first_number / second_number   # result is always  float
```

“Prime number”  Просто число
“Complex number”

```
# TODO: Add logic here
# TODO: Check the other cases…
```

## 2. Abbreviations

```
ABC - Abstract base classes (ABCs) enforce derived classes to
implement particular methods from the base class
from abc import ABC, abstractmethod

CRUD – Create, Read, Update, Delete

DRY - Don't Repeat Yourself (DRY) principle

Dunder – double underscore ???

MRO - Method Resolution Order - mro() -> list ; __mro__ -> tuple
class Teacher(Person, Employee):
print(Teacher.mro())  # [<class '__main__.Teacher'>, <class '__main__.Person'>, <class '__main__.Employee'>, <class 'object'>]
print(Teacher.__mro_) # (<class '__main__.Teacher'>, <class '__main__.Person'>, <class '__main__.Employee'>, <class 'object'>)

SOLID
    SRP – Single Responsibility Principle
    OCP – Open/Closed Principle
    LSP – Liskov Substitution Principle
    ISP – Interface Segregation Principle
    DIP - Dependency Inversion Principle





Mocking - A way to simulate a third party service in our app
Mocking - simulate the real behavior, we mock the services
and methods from other classes and simulate the real behavior
```

## 3. Algorithms

Mario – OOP-February 2023-Iterators and Generators – 30min to the end -

## 4. Booleans

```
self.is_on = not self.is_on
# change "on" to "off" or vice versa

print(bool(0))      # False
print(bool(-0))     # False
print(bool(""))     # False
print(bool(" "))    # True
print(bool(False))  # False
print(bool(None))   # False
print(bool(True))   # True
print(bool(1))      # True
print(bool("a"))    # True
```

## 5. Comprehensions

```
action, *info, last = command.split('-')
```

```
if x.isdigit() else x vs just if x.isdigit() - position in comprehension
in the middle                      at the end
action, way, *info = [int(x) if x.isdigit() else x for x in input().split()]
# Input - a b 1 2 3 12
print(action, way, info)  # a b [1, 2, 3, 12]
action, way, *info = [int(x) for x in input().split() if x.isdigit()]
# Input - a b 1 2 3 12
print(action, way, info)  # 1 2 [3, 12]
```

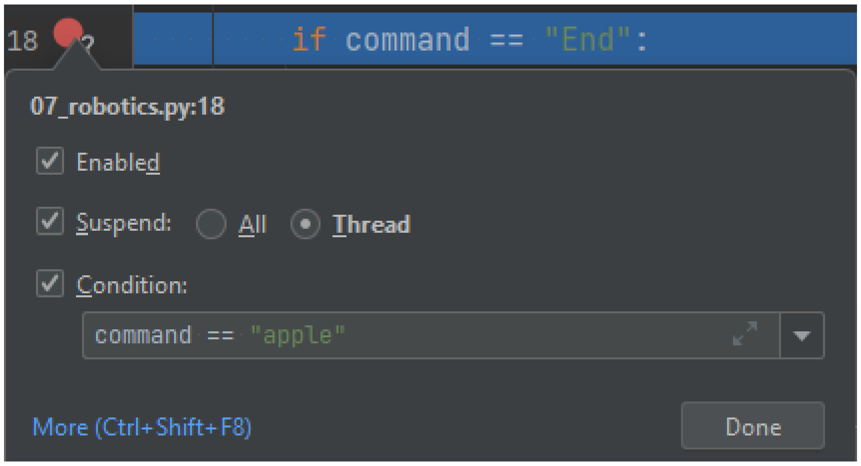
```
return {r1, c1}.issubset(range(rows)) and {r2, c2}.issubset(range(cols))

available_cargos = {'a': 10, 'b': 11, 'c': 3}
cargo_location = max(available_cargos, key=available_cargos.get)
print(cargo_location)  # b

03_Advanced\04_Multidimensional_Lists\Recapitulate\Exercises_2\03_knight_game.py
knight_attacks=len({(i+di,j+dj)for di,dj in positions if (i+di,j+dj) in knights})
knight_attacks=len({(i+di,j+dj)for di,dj in positions}.intersection(knights))
# row using intersection is faster than row with if

"\n".join(str(x) for x in [*self.customers, *self.dvds])

set1 = {input() for _ in range(n)}

data1, data2 = [list(map(int, el.split(','))) for el in input().split('-')]

materials.reverse() <=> materials[:: -1]

[print(f"{toy}: {crafted.count(toy)}") for toy in sorted(set(crafted))]

print(f"Milk: {', '.join(str(x) for x in cups) or 'empty'}")


# b = [1, 2, 3]
b = []
a = ["a", *b, "c"]
print(a)  # ['a', 'c']
```

```
res = [
    f"Username: {self.username}, Age: {self.age}",
    "Liked movies:" if self.movies_liked else "Liked movies:\nNo movies liked.",
    *[m.details() for m in self.movies_liked],
    "Owned movies:" if self.movies_owned else "Owned movies:\nNo movies owned.",
    *[m.details() for m in self.movies_owned],
]
```

```python
matrix = [[int(x) for x in input().split(", ")] for _ in
range(int(input().split(", ")[0]))]

email = "avs@gmail.com"  # correct
email = "avs@gmail.come"  # wrong
if any(email.endswith(x) for x in (".com", ".bg", ".net", ".org")):
    print("correct")
else:
    print("wrong")
```

## 6. Debugger

https://softuni.bg/trainings/resources/video/86023/video-28-june-2023-ines-kenova-python-oop-june-2023/4108 - 11 minute



```
18        if command == "End":
```

07_robotics.py:18

✓ Enabled

✓ Suspend: All [✓ Thread]

✓ Condition:

```
command == "apple"
```

right click over the existing break point (brake)

stops if command = = "apple" - this is new statement different from the one in file

## 7. Decimal

```
from decimal import localcontext, Decimal, ROUND_HALF_UP, ROUND_HALF_DOWN
round_half_correctly.py

import decimal - in decimal.py
ERROR скапах всичко

a = Decimal('0.1')
b = Decimal('0.1')
c = Decimal("0.1")
result = a + b + c   # 0.3
a = 0.1
b = 0.1
c = 0.1
result = a + b + c   # 0.30000000000000004
a = Decimal(0.1)      # without apostrophe
b = Decimal(0.1)      # without apostrophe
c = Decimal(0.1)      # without apostrophe
result = a + b + c  # 0.3000000000000000166533453694
price = Decimal("3 * 1.2")  Error
price = Decimal("3 + 1.2")  Error
no operations allowed, just one number
```

## 8. Dictionaries

```
symbols[ch] = symbols.get(ch, 0) + 1

d_test = {'a': [1, 2], 'b': [5, 6]} # key renaming
d_test['c'] = d_test.pop('a') # {'b': [5, 6], 'c': [1, 2]}

available_cargos = {'a': 10, 'b': 11, 'c': 3}
cargo_location = max(available_cargos, key=available_cargos.get)
print(cargo_location)  # b

available_cargos = {'a': 10, 'b': 11, 'c': 3}
products = {'z': 100, 'a': 100}
new = {}
new.update(**available_cargos)
print(new)  # {'a': 10, 'b': 11, 'c': 3}
new.update(**products)
print(new)  # {'a': 100, 'b': 11, 'c': 3, 'z': 100}


resources = {}
if key not in resources:
    resources[key] = 0

dict_test = {3: 4, 4: 5, 5: 5, 7: 2, 11: 2}
print(len(dict_test))
sorted_dict = dict(sorted(dict_test.items(), key=lambda x: (-x[1], -
x[0]))) # -x[0] error if x[0] is str!!!
print(sorted_dict) # {5: 5, 4: 5, 3: 4, 11: 2, 7: 2}
sorted_dict = dict(sorted(dict_test.items(), key=lambda x: (-x[1],
x[0]))) # -x[0] error if x[0] is str!!!
print(sorted_dict) # {4: 5, 5: 5, 3: 4, 7: 2, 11: 2}

# dict_test1 = {"k3": 4, "k4": 5, "k5": 5, "k7": 2}
# sorted_dict = dict(sorted(dict_test1.items(), key=lambda x: (x[1],
x[0]))) # -x[0] error if x[0] is str!!!
# print(sorted_dict)

# race_info = sorted(race_info, key=lambda x: -race_info[x]) #
returns list with keys sorted by values

# sorted(symbols.items())  # returns list of tuples
# dict_test = dict(sorted(symbols.items()))
# for ch, count in dict_test.items():
#     print(f"{ch}: {count} time/s")
# for ch, count in sorted(dict_test.items()):
#     print(f"{ch}: {count} time/s")
```

```
from collections import defaultdict
# from collections import OrderedDict

# student_info = defaultdict(list)
# # student_info = defaultdict(lambda: [0.0])
# for _ in range(int(input())):
#     name, grade = input().split()
#     # if name not in student_info: this check can be omitted with
defaultdict
#     #     student_info[name] = []
#     student_info[name].append(float(grade))

# x = ('key1', 'key2', 'key3')
# y = 0, 1, 2
# this_dict = dict.fromkeys(x)
# # this_dict = dict.fromkeys(x, y)
# print(this_dict)
# this_dict = dict(zip(x, y))
# print(this_dict)
#
# txt = "Hello, welcome to my world."
# print(txt.find("q"))  # -1 or index if q in txt
# print(txt.index("q"))  # Error or index if q in txt
#
car = {
    "brand": "Ford",
    "model": "Mustang",
    "year": 1964
}
# x = car.items()
# print(car)
# print(type(car))
# print(x)
# print(type(x))
# for key, value in car.items():
#     print(key, value)

# x = car.setdefault("model", "Bronco")  # return Mustang if key
exists
# print(x)
# print(car)
# y = car.setdefault("mod", "Bronco")  # add it and return Bronco if
key does not exist
# print(y)
# print(car)
#
# car.update({"model": "laguna"})  # change value if key exists
# print(car)
# car.update({"test": "New_mod"})  # add key, value if key does not
exist
# print(car)
# car["li"] = 5  # act as update
# print(car)
# car["model"] = "lag"  # act as update
# print(car)

# bus = {
#     "br": "Fo",
#     "model": "Mus",
#     "ye": 19
# }
# # car.setdefault(bus)  # Error - requires (key, value)
# # car.update("model", "Bronco")  # Error - requires dict
# car.update(bus)  # requires dict {key, value}
# print(car)


# x = car.get("br", )  # None
# print(x)
# x = car.get("br", 47)  # 47
# print(x)
# y = car["br"]  # Error
# print(y)

# x = car.keys()  # Returns a list containing the dictionary's keys
# x = car.values()  # Returns a list of all the values in the
dictionary

# for el in car.items(): # !!!! tuple is the answer
#     print(el)

# car.popitem()  # Removes the last inserted key-value pair
# car.pop("br")  # Removes key-value pair or Error
# car.pop("br", defaultvalue) returns defaultvalue and no Error

#
# a = ("a", "b", "c", "d")
# a = ("a", "b")
# b = ("1", "2", "3")
# x = zip(a, b)
# # print(tuple(x))
# print(x)
# print(dict(x))


# print({ch: ord(ch) for ch in input().split(',')})
# data = [("Peter", 22), ("Amy", 18), ("George", 35)]
# dict_data = {key: value for (key, value) in data}
# print(dict_data)
# print(f"{key}: {value} for (key, value) in data}") # do not work

# x = "012"
# y = "01234567"
# for i in range(len(y)):
#     j = i % len(x)
#     print(i, j, sep='->')

# print(list(car.items()))
# print(car['model'])

sponsors = {  # {sponsor: {position: reward}}
    "Petronas": {1: 1_000_000,
                 3: 500_000},
    "TeamViewer": {5: 100_000,
                   7: 50_000},
}
race_pos = 1
expenses = 200_000
revenue = - expenses

for sponsor in sponsors:
    for position in sponsors[sponsor]:
        if position >= race_pos:
            revenue += sponsors[sponsor][position]
            break

print(revenue)
```

## 9. Error-Handling

```
methods are faster than try except!!!

Syntax errors(parsing errors) and Exceptions

times = "asd"
print(7 / times)  # TypeError: unsupported operand type(s) for /: 'int' and 'str'
print("7" / times)  # TypeError: unsupported operand type(s) for /: 'str' and 'str'
print(7 / int(times))  # ValueError: invalid literal for int() with base 10: 'asd'
print(int("asd"))  # ValueError: invalid literal for int() with base 10: 'asd'
print(int([11]))  # TypeError: int() argument must be a string, a bytes-like object or a real number, not 'list'

try:
    times = int(input())
    # times = float(input())
except ValueError as ex:
    print(f"ValueError: {ex}")
    print("blabla")
except KeyError:
    print()
except (NameError, TypeError, IndexError) as ex:
    print(ex)

# custom exceptions
class SmallValueException(Exception):
    pass


class HighValueException(Exception):
    pass


amount = float(input())  # you cannot transfer negative money

if amount < 1:
    raise SmallValueException("Amount can not be less than 1lv.")
elif amount > 1000:
    raise HighValueException("Transaction limit max 1000")
# custom exceptions

try:
    print("try")
    a = 7
    b = int(input())  # if b = 0 print("End") would not be executed, 
but print("finally")
    c = a / b
except ValueError as text:
    print("ValueError")  # ValueError
    print(text)  # invalid literal for int() with base 10: 'dhhfd'
else:
    print("from else")  # Not very useful. will be executed if 
successful try.
finally:
    print("finally")  # will always be executed

print("End")  # if b = 0, code could not reach that line, because of 
error. if b = 'str' will print End.
# if b = 0 -> ZeroDivisionError. if b = 'str' ValueError.
```

## 10. File Handling

**io** (in / out) module is the default module for accessing files - Built-in

```
file = open('W:/1_Python/1-Training/1_Projects/1st_Project/text.py') correct
file = open('W:\1_Python\1-Training\1_Projects\1st_Project\text.py') wrong
```

We should always make sure that an open file is properly **closed**

To avoid **unwanted behaviour always close** the files

Files opened with **"with"** statement will be **closed automatically** once it leaves the **with** block

```
with open("file.txt", "w") as f:
    f.write("Hello World!!!")
     print(f.read())  # Error: io.UnsupportedOperation:
f is not readable if the file is open for writing, adding …
modes 'w', 'a' ….etc
```

- **w** - open for writing, truncating the file first. Truncating(съкращавам) - If the file exists, its **overwritten**
- **x** - create a new file and open it for writing
- **r** – open in reading mode. ‘r’ is by default. No diff, If ‘r’ or mode is empty.
- **a** - open for writing, appending to the end of the file. Or create a file, if it doesn’t exists.
- **t** - text mode (default)
- **b** - binary mode
- **+** - open a disk file for updating (reading and writing)
-

```
▪ try:
      file = open('zzz_text.py', 'r')
      print(file.read())
  except FileNotFoundError:
      print("File not found or path is incorrect")
  finally:
      print("exit")
```

```
file = open('text.txt')  # => open('python.txt', 'r')
print(file.read())
print(file.read(7))  # will print nothing if file has been read already
print(file.readline())
print(file.readline(7))
for line in file:  # line is str + \n
    # print(line)  # adds additional empty line after printing each line of file
    print(line, end="")  # will print nothing if file has been read already
    print(line.split())
print(file.read())  # will print nothing if file has been read in
"for line in file" already
file.close()
```

```
Delete File
import os

file_path = "text.txt"
if os.path.exists(file_path):
    os.remove(file_path)

try:
    os.remove('text.txt')
except FileNotFoundError:
    print('File already deleted!')
```

```
# region Directory manipulation
import os

os.path.isfile(path) # method that returns True if the path is a file or a
symlink(symbolic link) to a file.
os.path.exists(path) # method that returns True if the path is a file, directory,
or a symlink(symbolic link)  to a file.


# print(os.mkdir('W:/1_Python/1-
Training/1_Projects/1st_Project/Lessons_Notes/File_Handling_Notes/Tes
t_Folder'))
print(os.getcwd())  # Return a string representing the current
working directory.
# os.mkdir('Test')
# os.rmdir('W:/1_Python/1-
Training/1_Projects/1st_Project/Lessons_Notes/File_Handling_Notes/Tes
t_Folder')
# os.chdir('Test_Folder')
print(os.listdir('W:/1_Python/1-Training/1_Projects/1st_Project'))

# endregion
```

# 11. Formatting, Printing

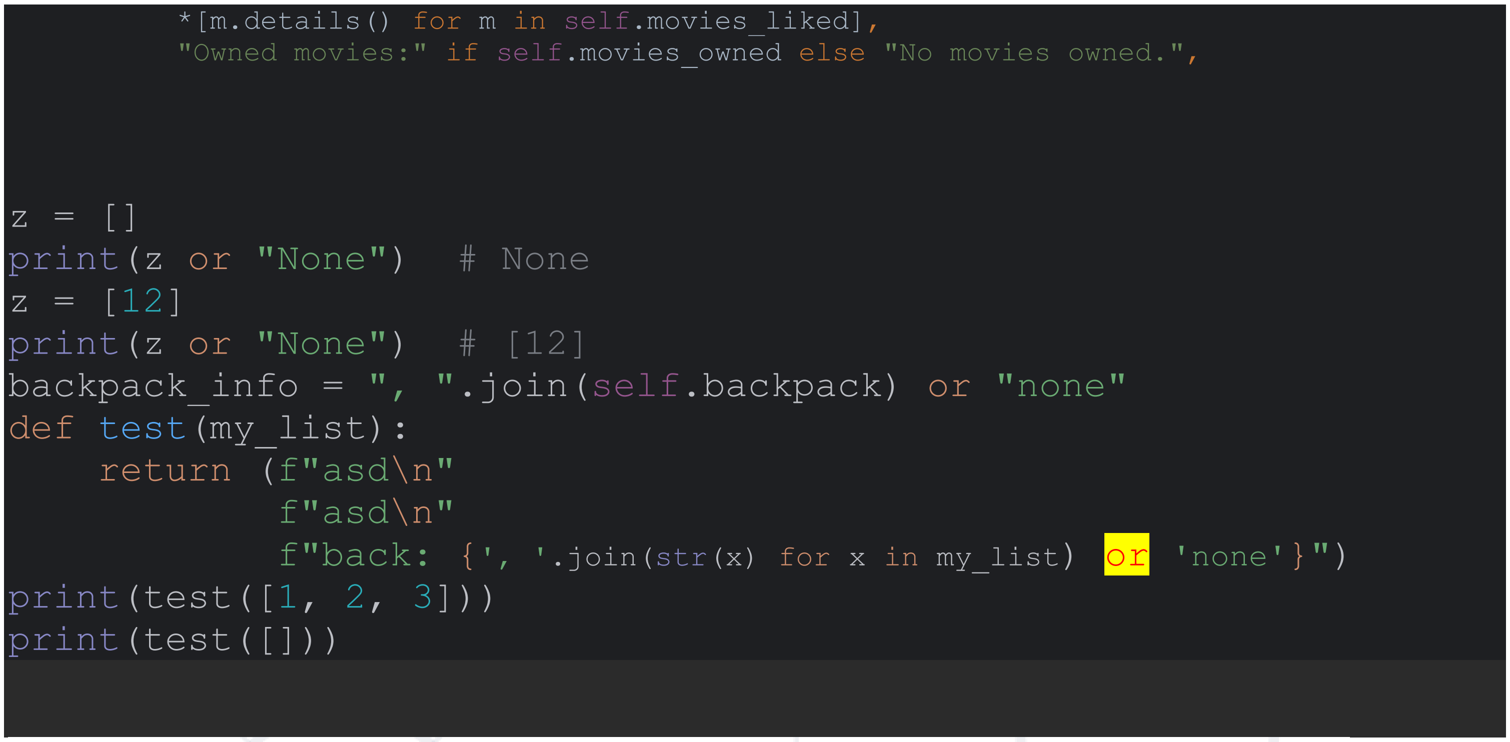
```
int(5 / 2) < = > 5 // 2
```

```
f"""{self.name} Library does not have {book_author}'s "{book_title}"."""

f"back: \n{', '.join(str(x) for x in my_list) or 'none'}" -> OK
f"back: \n{'\n'.join(str(x) for x in my_list) or 'none'}" -> not woring with \n
       OK  not OK
Expression fragments inside f-strings cannot include backslashes
string interpolation
```

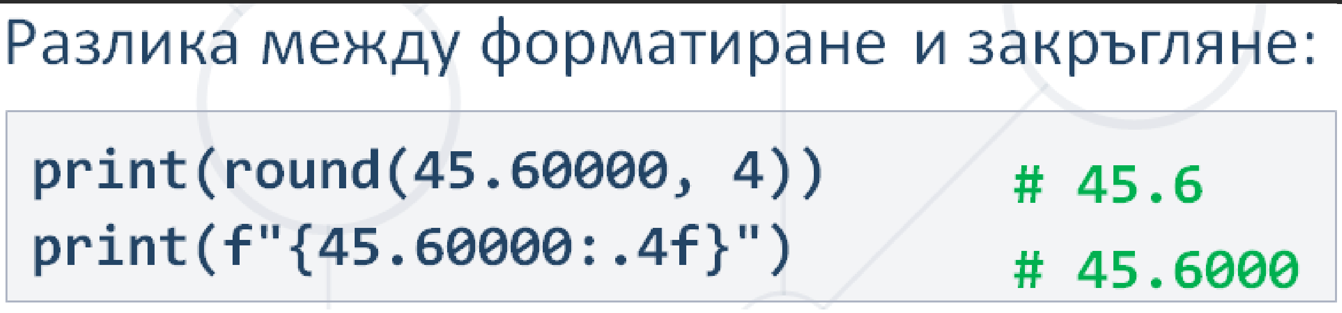
```
res = [
    f"Username: {self.username}, Age: {self.age}",
    "Liked movies:" if self.movies_liked else "Liked movies:\nNo movies liked.",
    *[m.details() for m in self.movies_liked],
    "Owned movies:" if self.movies_owned else "Owned movies:\nNo movies owned.",
    *[m.details() for m in self.movies_owned],
]
```

```
"\n".join(str(x) for x in [*self.customers, *self.dvds])

result = [
    f"You have {len(self.workers)} workers",
    f"----- {len(info['Keeper'])} Keepers:",
    *info["Keeper"],
    f"----- {len(info['Caretaker'])} Caretakers:",
    *info["Caretaker"],
    f"----- {len(info['Vet'])} Vets:",
    *info["Vet"]
]

orders = list("abcdef")
print("Orders left: ", end='')
print(*orders, sep=', ') # * splat operator
print("Orders left:", *orders, "text", '.')
print(int(1.5))        #  1

print(f"{minutes}:{seconds:02d}")  #  5:07
print(f"{num:.1f}")  #  1 -> 1.0 ; 1.333 -> 1.3

print(round(4.5))       #-> 4 round to nearest even number
print(round(5.5))       #-> 6 banker's number
x = 4.5
print(f'{x:.0f}')    #-> 4 round to nearest even number
x = 5.5
print(f'{x:.0f}')    #-> 6 banker's number
```

```
result = [f"Username: {self.username}, Age: {self.age}",
          "Liked movies:" if self.movies_liked else "No movies liked.",
```



```
        *[m.details() for m in self.movies_liked],
        "Owned movies:" if self.movies_owned else "No movies owned.",



z = []
print(z or "None")   # None
z = [12]
print(z or "None")   # [12]
backpack_info = ", ".join(self.backpack) or "none"
def test(my_list):
    return (f"asd\n"
            f"asd\n"
            f"back: {', '.join(str(x) for x in my_list) or 'none'}")
print(test([1, 2, 3]))
print(test([]))
```



Разлика между форматиране и закръгляне:

```
print(round(45.60000, 4))         # 45.6
print(f"{45.60000:.4f}")          # 45.6000
```

# 12. Functions

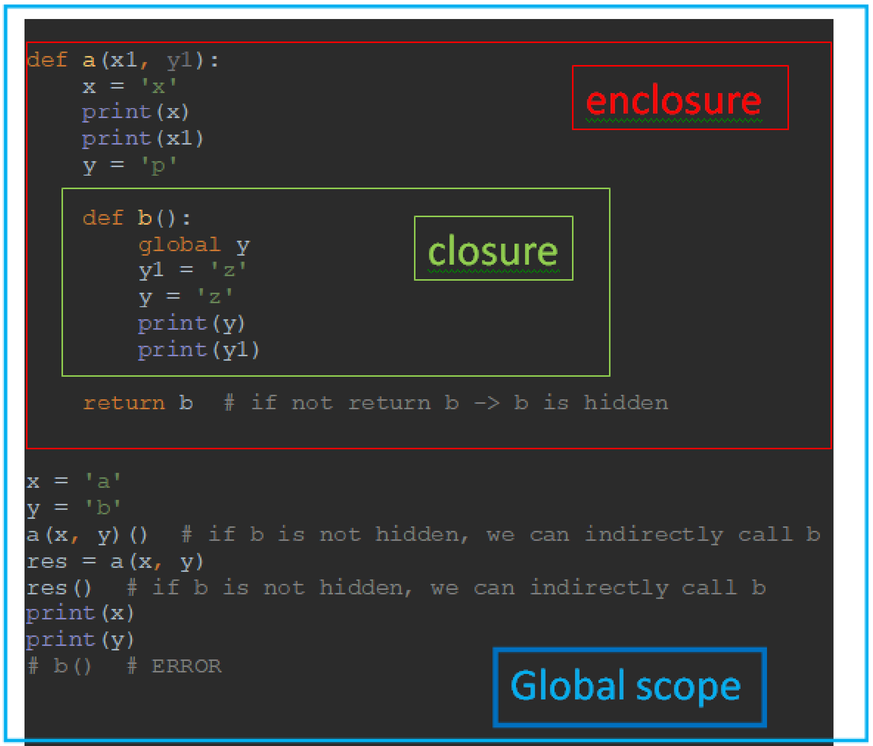
## 12.1. General

```
def sum_nums(a, c=5, action, *args): # is OK
def even_odd(*args): # is OK
def even_odd(*args, action): # not OK action to be at the end
print(even_odd(1, 2, 3, 4, 5, 6, "even"))

def test_kwargs(a, **kwargs):  # **kwargs -> dict with 0 or more ele-> a: 3  kwargs:
{'b': 7, 'c': 12}
    print(a)
    print(kwargs)

my_dic = {'b': 7, 'c': 12}
test_kwargs(3, **my_dic) ⇔ test_kwargs(3, b=7, c=12)
**my_dic ⇔ b=7, c=12
test_kwargs(a=3)


*args (packing)-> tuple with 0 or more ele-> a: 3 c: 7 args: (12, 19)
**kwargs -> dict with 0 or more ele-> a: 3  kwargs: {'b': 7, 'c': 12}

def add_number_12(num_seq):
    num_seq.append(12)
# no return, but list nums is modified
nums = [1, 2, 3]
print(nums)  # [1, 2, 3]
add_number_12(nums)  # no return, but list nums is modified, because lists
are referenced. num_seq and nums are pointing to one and the same place in
memory
print(nums)  # [1, 2, 3, 12]

print(**{"name": "George", "town": "Sofia", "age": 20})  # error
print(*[1, 2, 3])  # OK

def get_info(name, age, town):
    return f"This is {name} from {town} and he is {age} years old"
print(get_info(**{"name": "George", "town": "Sofia", "age": 20}))  #
**(unpacking) transforms dict to next row
print(get_info(name="George", town="Sofia", age=20))  # kwargs can read this tipe
of info
print(get_info("George", "Sofia", 20))  # for correct result needs correct
sequence
```

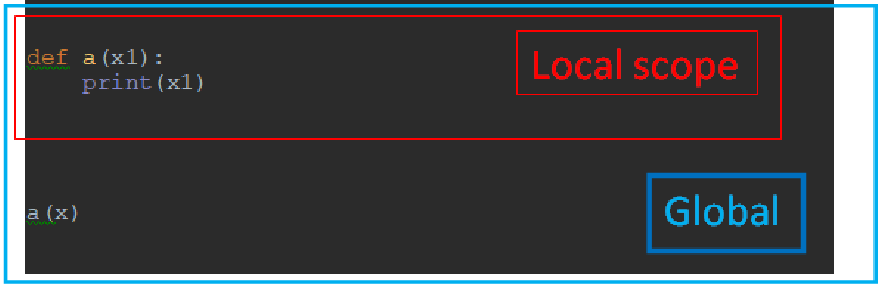
## 12.2. Function executor

```
03_Advanced\05_Functions_Advanced\Recapitulate\Exercises\06_function_executor.py

def func_executor(*args):
    return "\n".join(f"{el[0].__name__} - {el[0](*el[1])}" for el in args)

Test One !!!
def sum_numbers(num1, num2):
    return num1 + num2


def multiply_numbers(num1, num2):
    return num1 * num2


print(func_executor(
    (sum_numbers, (1, 2)),
    (multiply_numbers, (2, 4))
))

Test 2 !!!
def make_upper(*strings):
    result = tuple(s.upper() for s in strings)
    return result

def make_lower(*strings):
    result = tuple(s.lower() for s in strings)
    return result

print(func_executor(
    (make_upper, ("Python", "softUni")),
    (make_lower, ("PyThOn",)),
))
```

## 12.3. Scopes



```
def a(x1, y1):
    x = 'x'
    print(x)
    print(x1)
    y = 'p'

    def b():
        global y
        y1 = 'z'
        y = 'z'
        print(y)
        print(y1)

    return b  # if not return b -> b is hidden

x = 'a'
y = 'b'
a(x, y)()  # if b is not hidden, we can indirectly call b
res = a(x, y)
res()  # if b is not hidden, we can indirectly call b
print(x)
print(y)
# b()  # ERROR
```

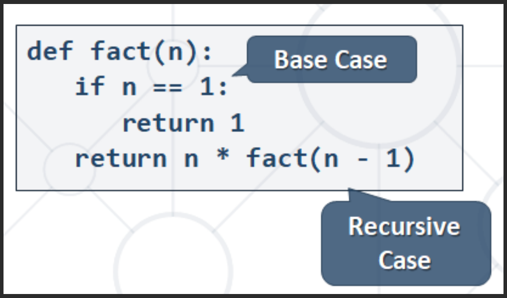


```
def a(x1):
    print(x1)


a(x)
```

```
def a(x1, y1): # no task in judge for global and non local - don’t use them
    x = 'xa'  # not changed on global scope
    print(x)  # xa  => changed on local scope
    print(x1)  # x  => not changed on global scope

    def b():
        global y  # y  => changed on global scope
        nonlocal y1
        y1 = 'y1b'
        y = 'yb'
        print(y)  # yb  => changed on global scope
        print(y1)  # y1b  => changed on local scope

    return b  # if not return b -> b is hidden
x = 'x'
y = 'y'
a(x, y)()  # if b is not hidden, we can indirectly call b
# res = a(x, y)
# res()  # if b is not hidden, we can indirectly call b
print(x)
print(y)
# # b()  # ERROR
```

## 12.4. Recursion

The process in which a function calls itself is called **recursion**
A recursive function has the following structure:
base case and recursive case


```
def fact(n):
    if n == 1:
        return 1
    return n * fact(n - 1)
```

1-Training\1_Projects\1st_Project\Lessons_Notes\recursive_funcs.py

```
def not_recursion():
    def not_recursion():
        def not_recursion():
            print(3)
        print(2)
        not_recursion()
    print(1)
    not_recursion()
not_recursion()

def a():  # infinite recursion
    a()

a()  # [Previous line repeated 996 more times]
# RecursionError: maximum recursion depth exceeded


def recursive_power(num, power): # short but not good for debugging
    if power == 0:
        return 1
    return num * recursive_power(num, power - 1)
def recursive_power(number, power): # longer but in debug you can see how recursion works
    result = 1
    if power == 0:
        return result
    result = number * recursive_power(number, power - 1)
    return result
    # return number ** power
print(recursive_power(2, 3))
```

## 13. Imports

```
from string import punctuation  # !"#$%&'()*+,-./:;<=>?@[\]^_`{|}~

import math
x = 5.98
print(math.floor(x))    -> not floor(x)
print(int(x))           => floor(x)
from math import ceil, floor
x = 5.98
print(floor(x))     -> not math.floor(x)

import decimal - in decimal.py
ERROR скапах всичко

import random
number = random.randint(1, 100)
print(number)

from functools import reduce
map_functions = {
    '*': lambda x: reduce(lambda a, b: a * b, x),
    '/': lambda x: reduce(lambda a, b: a / b, x),
# '/': lambda x: reduce(lambda a, b: a + b if a == 0 or b == 0 else a / b, x),
    '+': lambda x: reduce(lambda a, b: a + b, x),
    '-': lambda x: reduce(lambda a, b: a - b, x),
} 02_expression_evaluator_a.py in 03_Stacks_Queues_Tuples_and_Sets_Exercise

from string import ascii_lowercase
chars = list(ascii_lowercase)
```

```
py -m pip install PyQt5
py -m pip install pyfiglet   or keep the cursor over the library and click install
py -m pip install opencv-python
```

## 14. Lists

```
Be very careful with remove in for cycle!!!

"\n".join(str(x) for x in [*self.customers, *self.dvds])
```

```
if x.isdigit() else x vs just if x.isdigit() - position in comprehension
in the middle                 at the end
action, way, *info = [int(x) if x.isdigit() else x for x in input().split()]
# Input - a b 1 2 3 12
print(action, way, info)  # a b [1, 2, 3, 12]
action, way, *info = [int(x) for x in input().split() if x.isdigit()]
# Input - a b 1 2 3 12
print(action, way, info)  # 1 2 [3, 12]
```

```
fruits = ['apple', 'banana', 'cherry']
x = fruits.pop(1)
print(x)  # banana
```

```
nums = [1, 2, 3]
nums2 = nums  # referenced
nums3 = nums.copy() ⇔ list(nums) # not referenced

bottles = list(map(int, input().split()))
my_list = list(range(5))  # [0, 1, 2, 3, 4]


enumerate <class 'enumerate'>
print(list(enumerate(list("123"))))  # [(0, '1'), (1, '2'), (2, '3')]
print(list(enumerate(list(range(3)))))  # [(0, 0), (1, 1), (2, 2)]

indexes = [idx for idx, el in enumerate(test_tuple) if el == "asd"]
return list with idx for all el == "asd"

x = [[]] * 3  # [[], [], []]
x[1].append(5)  # [[5], [5], [5]] !!!
y = [[] for _ in range(3)]  # [[], [], []]
y[1].append(5)  # [[], [5], []]
a = [0] * 3  # [0, 0, 0]
a[2] += 7
print(a)  # [0, 0, 7]

my_list = [1, 2, 3, 1, 2, 2,  2, 2, 4, 5, 'a']
result = list(filter(lambda x: x == 2, my_list))  # [2, 2, 2, 2, 2]
result1 = next(filter(lambda x: x == 2, my_list))  # 2
result2 = next(filter(lambda x: x == 7, my_list), "Not in list")  # Not in list
# result3 = next(filter(lambda x: x == 7, my_list))  # StopIteration (error)
```

```
a = [1, 2, 3]
b = ['w', 'f']
d = [*a, *b]
print(d)  # [1, 2, 3 'w', 'f']
print(*d)  # 1 2 3 w f

a = "12345"
b = list(a)  # ['1', '2', '3', '4', '5']

# removing elements in the middle of the list
a_nums = a_nums[:left_idx] + a_nums[right_idx + 1:]
print(a_nums)
# =>
for i in range(idx + value, idx - value - 1, -1):
    b_nums.pop(i)
print(b_nums)
# =>
del c_nums[left_idx:right_idx + 1]
print(c_nums)
```

## 15. Matrix

```
matrix = [[0 for j in range(2)] for i in range(3)] # [[0, 0], [0, 0], [0, 0]]
matrix = [[0 for _ in range(2)] for _ in range(3)] # [[0, 0], [0, 0], [0, 0]]

matrix = [[int(j) for j in input().split(", ") if int(x) % 2 == 0] for i in
range(int(input()))]
matrix = [[int(x) for x in input().split(", ") if int(x) % 2 == 0] for _ in
range(int(input()))]

# flattening matrix 2d
matrix = [[int(j) for j in input().split(", ")] for i in range(int(input()))]
flatten_matrix = [el for list_i in matrix for el in list_i]
# flattening matrix 3d
m3d = [[[k for k in range(3)] for j in range(3)] for i in range(3)]
print(m3d)  # [[[0, 1, 2], [0, 1, 2], [0, 1, 2]], [[0, 1, 2], [0, 1, 2], [0, 1,
2]], [[0, 1, 2], [0, 1, 2], [0, 1, 2]]]
flatten_m3d = [k for m2d in m3d for list_i in m2d for k in list_i]
print(flatten_m3d)  # [0, 1, 2, 0, 1, 2, 0, 1, 2, 0, 1, 2, 0, 1, 2, 0, 1, 2, 0,
1, 2, 0, 1, 2, 0, 1, 2]

# sum of primary or secondary diagonal
n = int(input())
matrix = [[int(x) for x in input().split()] for _ in range(n)]
primary_diagonal = sum([matrix[i][i] for i in range(n)])
secondary_diagonal = sum([matrix[i][n - i - 1] for i in range(n)])
print(primary_diagonal)
print(secondary_diagonal)

# faster solution
primary_diagonal_sum = 0
```

```
secondary_diagonal_sum = 0
for i in range(n):
    row = [int(x) for x in input().split()]
    primary_diagonal_sum += row[i]
    secondary_diagonal_sum += row[n - i - 1]
print(primary_diagonal_sum)
print(secondary_diagonal_sum)
```

03 Advanced\04 Multidimensional Lists\Recapitulate\Exercises 2\03 knight game.py

```
possible_moves = {(i + di, j + dj)
                  for v, h in [[1, 2], [2, 1]]
                  for di, dj in [[v, h], [v, -h], [-v, h], [-v, -h]]
                  if i + di in range(n) and j + dj in range(n)}
```

03 Advanced\04 Multidimensional Lists\Exercises 2\04 easter bunny.py

```
directions = {
    "up": (-1, 0),
    "down": (1, 0),
    "left": (0, -1),
    "right": (0, 1)
}
```

# 16. OOP

```
from sys import path
print(*path, sep="\n") # prints Source Root Directories

from typing import List, Dict  ….

          four 4 central principles of OOP
Inheritance, Encapsulation, Abstraction, Polymorphism

Mangling
class is a blueprint that defines the nature of a future
object

self.fuel_consumption = self.DEFAULT_FUEL_CONSUMPTION
self.fuel_consumption = Vehicle.DEFAULT_FUEL_CONSUMPTION
if we want subclass to have own DEFAULT_FUEL_CONSUMPTION we must use
self but not Vehicle.
04_OOP\03_Inheritance\Exercises\04_Need for Speed

@property - calling instance.expensed - no braces ()
def expenses(self) -> int:
    return 200_000 is it possible to change it??????
it's purpose is to return values:
 int, dict, list, str ……  self.__name….

c.__class__.__name__

def __getitem__(self, item):
    return self.people
def __getitem__(self, idx: int):
    return self.people[idx]
or  return f"Person {idx}: {self.people[idx]}"
04_OOP/06_Polymorphism_and_Abstraction/Exercises/02_groups.py
```

## 16.1. First-Steps-in-OOP

```
Object is a data abstraction that captures an internal
representation and an interface
The interface defines behaviors but hides implementation
State(Data) attributes - Instance variables and Class variables
behavior attributes - methods are like functions,
that work only within a class
```

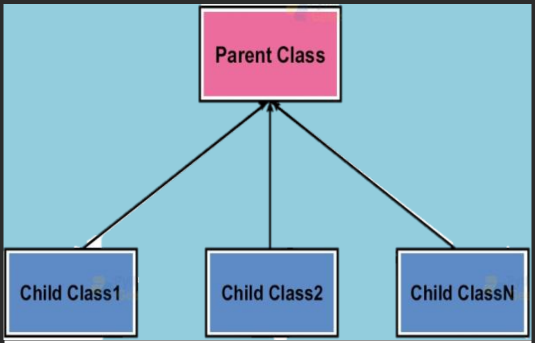
## 16.2. Classes-and-Objects

```
def __init__(self, mileage, max_speed: int = 150)
def __init__(self, mileage, max_speed=150)
Example.text         # attribute reference – state(data attribute)
Example.print_text   # attribute reference – behavior(method)
x = Example()        # instantiation – uses function notations
There are two kinds of attribute references: Data and Methods
Data attributes - Instance variables and Class variables
Instance variables - unique to each instance
It is not a good practice to declare or remove data attributes
outside the class
Class variables - shared by all instances of the class
class Customer:
    id = 1
    def __init__(self):
        self.id = Customer.id
c1 = Customer()
print(c1.id)  # 1
Customer.id = 2  # instances created before this row will have
one class variable value(c1.id = 1) (data attribute), and after this row –
other class variablevalue(c2.id = 2), (c1.id = 1)
c2 = Customer()
print(c1.id)  # 1
print(c2.id)  # 2
print(Customer.id)  # 2

Built-in methods "magic" or "dunder"
Surrounded by double underscores __dict__
Dunder – double underscore ???
__str__() – returns a printable string representation
__repr__() – returns a machine-readable representation
__doc__() – Provides a documentation of the object as a string
class MyClass:
    """This is MyClass."""
    def example(self):
        """This is the example module of MyClass."""
print(MyClass.__doc__) # This is MyClass.
print(MyClass.example.__doc__) # This is the example module of MyClass.


__dict__() is a dictionary containing a module's symbol table
class Dog:
    def __init__(self, name):
        self.name = name
x = Dog("Max")
print(x.__dict__) # {"name": "Max"}
```

## 16.3. Inheritance

```
class Person(object): ⇔ class Person:

class Bird(Animal):
    def __init__cursor(self) press Alt + Enter and result is:
class Bird(Animal):
    def __init__(self, name: str, weight: float):
        super().__init__(name, weight)

self.fuel_consumption = self.DEFAULT_FUEL_CONSUMPTION
self.fuel_consumption = Vehicle.DEFAULT_FUEL_CONSUMPTION
if we want subclass to have own DEFAULT_FUEL_CONSUMPTION we must use
self but not Vehicle.
04_OOP\03_Inheritance\Exercises\04_Need for Speed

Single, Multiple, Multilevel, Hierarchical, Hybrid Inheritance

class Student(Person):
    def __init__(self, name, age, student_id):
        super().__init__(name, age) if we need to add student_id
        self.student_id = student_id # Data attribute
class Student(Person):
# will not inherit any Data attribute
# will inherit superclass methods only
      pass or def some_method

class Daughter(Father, Mother): # Multiple Inheritance
    def __init__(self):
        # super().__init__() will inherit Father only
        Father.__init__(self)
        Mother.__init__(self)
class Person:
    def sleep(self):
        return "sleeping..."
class Employee:
    def get_fired(self):
        return "fired..."
class Teacher(Person, Employee): # Multiple Inheritance
    def teach(self):
        return "teaching..."
teacher = Teacher()
print(teacher.__class__.__bases__[0].__name__)  # Person
print(teacher.__class__.__bases__[1].__name__)  # Employee
MRO - Method Resolution Order - mro() -> list ; __mro__ -> tuple
print(Teacher.mro()) # [<class '__main__.Teacher'>, <class '__main__.Person'>, <class '__main__.Employee'>, <class 'object'>]
print(Teacher.__mro__) # (<class '__main__.Teacher'>, <class '__main__.Person'>, <class '__main__.Employee'>, <class 'object'>)
```

```
Hierarchical Inheritance
class Parent:
    def init(self, name):
        self.name = name
class Daughter(Parent):
    def __init__(self, name):
        super().__init__(name)
class Son(Parent):
    def __init__(self, name):
        super().__init__(name)
Hierarchical Inheritance
```



Hierarchical Inheritance 1

```
mixin is a class that has no data, only methods
mixin cannot be instantiated by themselves
mixin is needed in many different classes
04_OOP\05_Static_and_Class_Methods\Exercises\04_Gym_with_mixin
class NextIdMixin:
    id = 0
    @classmethod
    def get_next_id(cls):
        cls.id += 1
        return cls.id
class Customer(NextIdMixin):
    id = 0
    def __init__(self, name: str, address: str, email: str):
        self.id = self.get_next_id()
```

## 16.4. Encapsulation

```
prop + Tab @property (getter) it's designed to return values
int, dict, list, str ……
props + Tab @property and @???.setter (getter + setter)
Encapsulation is Packing of data and methods into a single
component
Encapsulation put restrictions and can prevent the accidental
modification of data
To do that, an object's variable can only be changed by an
object's method
Everything written within the Python class (methods and variables)
are public by default
Python implements weak encapsulation. This means it is performed
by convention rather than being enforced by the language
It is a matter of convention to differentiate them into three
terms - public, protected and private
Using a single leading underscore is just a convention
naming an attribute with two leading underscores invokes name
mangling (red_car._Car.__max_speed) is used for attributes that one
class does not want subclasses to use, but it is still possible to
access or modify a variable that is considered "private" from
outside the class
def get_id(self, pin) -> (str, int):
    if pin == self.__pin:
        return self.__id
def change_pin(self, old_pin, new_pin) -> str:
    if old_pin == self.__pin:
        self.__pin = new_pin
class Person:
    def __init__(self, name: str, age: int):
        self.__name = name
        self.__age = age
    def get_name(self): # or @property def name(self): return self.__name
        return self.__name
    def get_age(self):
        return self.__age
person = Person("George", 32)
print(person.get_name())
print(person.get_age())  # 32
person.age = 37  # person.age is variable type(int) in that case
print(person.get_age())  # 32
print(person.age)  # <class 'int'>
print(type(person.age))  # 37 - person.age is variable in that
case
print(person.age())  # TypeError: 'int' object is not callable
```

__get_fuel_and_speed(self) - "private" class method that should only be called from inside the class where it is defined

```
class Car:
    def __init__(self, fuel: int):
        self.fuel = fuel
        self.__max_speed = 200
    def drive(self): # car = Car(12) -> car.drive() - calling
        print('driving max speed ' + str(self.__max_speed))
    @property # property method calling vs method calling()
    def fuel(self): car.fuel - no ()calling, because @property
        return self.__fuel
    @fuel.setter
    def fuel(self, value):
        if value < 100:
            self.__fuel = value
    def __get_fuel_and_speed(self):  # "private"  class method
        return f"{self.fuel} - {self.__max_speed}"
    def get_info(self):
        return self.__get_fuel_and_speed()
red_car = Car(47)
# print(red_car.__max_speed) # AttributeError: 'Car' object has no
attribute '__max_speed'
print(red_car._Car__max_speed)  # 200 name mangling
red_car.drive()  # driving max speed 200
red_car.__max_speed = 10  # won't change because it is name
mangled
red_car.drive()  # driving max speed 200
print(red_car.fuel)  # 47
red_car.fuel = 120  # 47 because 120 > 100 - AttributeError if no @fuel.setter
red_car.fuel = 83  # 83 - AttributeError if no @fuel.setter
print(red_car.__get_fuel_and_speed())  # AttributeError: 'Car'
object has no attribute '__get_fuel_and_speed'
print(red_car.get_info())  # 83 - 200
```

hasattr()

method takes two parameters - **Object and Name**

```
class Person:
    def __init__(self, name):
        self.name = name
person = Person('Peter')
print(hasattr(person, 'name')) # True
print(hasattr(person, 'age'))  # False
```

getattr()

```
class Person:
    def __init__(self, name):
        self.name = name
person = Person('Peter')
print(getattr(person, 'name'))         # True
print(getattr(person, 'age'))          # AttributeError
print(getattr(person, 'age', 'None'))  # None
```

__getattr__()

```
class Phone:
    def __getattr__(self, attr):
        return None
phone = Phone()
print(phone.color)            # None
print(getattr(phone, 'size')) # None
"""__getattribute__ gets called “first”(the highest priority),
 whether or not there's the attribute.
  __getattr__ gets called “last”(the lowest priority),
   if Python cannot find the attribute"""
```

setattr()

method takes three parameters - **Object, Name and Value**

```
class Person:
    def __init__(self, name):
        self.name = name
person = Person('Peter')
print(setattr(person, 'name', 'George'))  # None - returns None
print(person.name)                        # George
print(setattr(person, 'age', 21))         # None - returns None
print(person.age)                         # 21
```

__setattr__()

method takes 2 parameters -**Name and Value**

```
class Phone:
    def __setattr__(self, attr, value):
        self.__dict__[attr] = value.upper()
phone = Phone()
phone.color = 'black'
print(phone.color)  # BLACK
```

```
class Person:
    def __init__(self, name: str, age: int):
        self.__name = name
        self.__age = age  # _Person__age
p = Person("Tom", 23)
print(p.__age)  # AttributeError: 'Person' object has no attribute '__age'
print(p._Person__age)  # 23
```

```
delattr()
method takes two parameters - Object and Name
class Person:
    def __init__(self, name):
        self.name = name
person = Person('Peter')
print(person.name)                       # Peter
print(delattr(person, 'name'))           # None
print(person.name)                       # AttributeError
__delattr__()
method takes 1 parameter - Name
class Phone:
    def __delattr__(self, attr):
        del self.__dict__[attr]
        print(f"'{str(attr)}' was deleted")
phone = Phone()
phone.color = 'black'
del phone.color  # 'color' was deleted
```

## 16.5. Static-and-Class-Methods

staticmethod knows nothing about the class or instance it is called on
**cannot modify** object state or class state

```
class Book:
    def __init__(self, name):
        self.name = name
b1, b2, b3 = Book("a"),  Book("b"),  Book("c")
class Customer:
    def __init__(self):
        self.books: List[Book] = [b1, b2, b3]
    @staticmethod
    def find_object(collection: list, attribute: str, value: str):
        for obj in collection:
            if str(getattr(obj, attribute)) == value:
                return obj
    def find_book(self, book_name):
        # b = self.find_object(self.books, "name", book_name)
        # if b:
        #     return b
        # return "no book"
        try:
            return [b for b in self.books if b.name == book_name][0]
        except IndexError:
            return "no book"
```

@classmethod can modify a class state that would apply across all the instances of the class
provide a shortcut for creating new instance objects
Ensures **correct instance creation** of the derived class
**easily follow** the Don't Repeat Yourself (DRY) principle

```
class Pizza:
    def __init__(self, ingredients):
        self.ingredients = ingredients
    @classmethod
    def pepperoni(cls):
        return cls(["tomato sauce", "parmesan", "pepperoni"])
first_pizza = Pizza.pepperoni()
print(first_pizza.ingredients)  # ['tomato sauce', 'parmesan', 'pepperoni']
```

## 16.6. Polymorphism and Abstraction

```
Methods are interface

Polymorphism is based on the Greek words "poly" (many) and
"morphism" (forms) vs duck typing
Polymorphism is ability to take different forms

Polymorphism is overriding method of superclass
Polymorphism is connected with inheritance, while duck typing not.
duck typing doesn’t care about objects' types, but whether they
have the methods we need
def robot_sensors(robot): # object must be Robot type - Polymorphism
def start_playing(obj): # it can be any type of obj
    return obj.play() # but must have play method - duck typing
04_OOP\06_Polymorphism_and_Abstraction\polymorphism_and_abstraction.py


Python does not support compile-time polymorphism or method
overload. If a class has multiple methods with the same name, the
method defined in the last will override the earlier one
class Person:
    def say_hello():
        return "Hi!"
    def say_hello():
        return "Hello"
print(Person.say_hello())  # Hello

def number_of_robot_sensors(robot):
    try:
        print(robot.sensors_amount())
    except AttributeError:
        print("unknown robot")

Abstraction is a process of handling complexity by hiding
unnecessary information from the user
```

## Operator Overloading

| Magic Methods | Get Called Using |
|---|---|
| __add__(self, other) | + |
| __sub__(self, other) | - |
| __mul__(self, other) | * |
| __floordiv__(self, other) | // |
| __truediv__(self, other) | / |
| __pow__(self, other[, modulo]) | ** |
| __lt__(self, other) | < |
| __le__(self, other) | <= |
| __eq__(self, other) | == |
| __ne__(self, other) | != |
| __gt__(self, other) | > |
| __ge__(self, other) | >= |

```
04_OOP\06_Polymorphism_and_Abstraction\operator_overloading.py
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y
    def __add__(self, other):
        return Point(self.x + other.x, self.y + other.y)
    def __str__(self):
        return f"({self.x}, {self.y})"
p1 = Point(3, 7)
p2 = Point(1, 2)
p3 = p1 + p2 # error if no def __add__(self, other):
print(p3.x, p3.y)  # 4 9
print(p3)  # (4, 9)

class Purchase:  # sofa, table; 800
    def __init__(self, product_name, cost):
        self.product_name = product_name
        self.cost = cost
    def __add__(self, other):
        name = f'{self.product_name}, {other.product_name}'
        cost = self.cost + other.cost
        return Purchase(name, cost)
first_purchase = Purchase('sofa', 650)
second_purchase = Purchase('table', 150)
print(first_purchase + second_purchase)  # sofa, table; 800
```

```
class Person:
    def __init__(self, name, salary):
        self.name = name
        self.salary = salary
    def __gt__(self, other):
        return self.salary > other.salary
person_one = Person('John', 20)
person_two = Person('Natasha', 36)
print(person_one > person_two)  # False
04_OOP\06_Polymorphism_and_Abstraction\operator_overloading.py
```

## Abstraction

**Abstraction is a process of handling complexity by hiding unnecessary information from the user**

Abstraction can be achieved by:
Abstract classes - MUST contain one or more abstract methods
or Functions and methods - declared but contain no implementation
Abstract classes - may not have @abstractmethod if superclass is abstract class and have @abstractmethod, but must inherit superclass and ABC
Abstract classes may not be instantiated and require subclasses to provide implementations for the abstract methods

Abstract base classes (ABCs) enforce derived classes to implement particular methods from the base class

```
from abc import ABC, abstractmethod
class Animal(ABC):
    def __init__(self, name):
        self.name = name
    @abstractmethod
    def sound(self):
        # raise NotImplementedError("Subclass must implement")
        pass
class Mammal(Animal, ABC(if no ABC must have def sound(self))):
It’s abstract class but no @abstractmethod,
because the superclass is abstract class and have @abstractmethod
    def __init__(self, name: str, weight: float, living_region: str):
        Animal.__init__(self, name, weight)
        self.living_region = living_region
class Dog(Animal):
    def __init__(self, name):
        super().__init__(name)
    def sound(self):  # TypeError: if def sound not implemented
        print("Bark!")
class Cat(Animal):
    def __init__(self, name):
        super().__init__(name)
    def sound(self): # TypeError: if def sound not implemented
        print("Meow!")
cat = Cat("Willy")
cat.sound()
dog = Dog("Willy")
dog.sound()
# animal = Animal("Willy")  # TypeError: Can't instantiate abstract class Animal with abstract method sound
04_OOP\06_Polymorphism_and_Abstraction\Exercises\04_Wild_Farm\project\animals\animal.py
```

```
Abstraction could be achieved using exceptions, but it is not a
good practice
class Shape:
    def __init__(self):
        if type(self) is Shape:
            raise Exception('This is an abstract class')
    def area(self):
        raise Exception('This is an abstract class')
    def perimeter(self):
        raise Exception('This is an abstract class')
```

## 16.7. SOLID

```
SOLID
    SRP - Single Responsibility Principle
    OCP - Open/Closed Principle
    LSP - Liskov Substitution Principle - introduced by Barbara Liskov in a 1987
    ISP - Interface Segregation(KEEP SEPARATE) Principle
    DIP - Dependency Inversion Principle

SRP - Single Responsibility Principle
Each class is responsible for only one thing and
should have only one reason to change
class that has many responsibilities is coupling these
responsibilities together, which leads to complexity and fragility
We can avoid the domino effect if the application changes by
splitting the class
class Book: - splitting by adding Library class and removing location
    def __init__(self, title, author, ~~location~~):
        self.title = title
        self.author = author
~~        self.location = location~~
        self.page = 0
    def turn_page(self, page):
        self.page = page
class Library:
    def __init__(self):
        self.books: List[Book] = []
    def find_book(self, book_title) -> (Book, str):
        try:
            return [b for b in self.books if b.title == book_title]
        except IndexError:
            return "no book"
```

```
OCP - Open/Closed Principle
classes, modules, and functions should be open for extension but
closed for modifications
can be achieved through: Abstraction, Mix-ins
Monkey-Patching, Generic functions (using overloading)
class StudentTaxes: Keep the class unchanged
    def __init__(self, name, semester_tax, avg_grade):
        self.name = name
        self.semester_tax = semester_tax
        self.average_grade = avg_grade
    def get_discount(self):
        if self.average_grade > 5:
            return self.semester_tax * 0.4
Extend the base class functionality by adding new class
class AdditionalDiscount(StudentTaxes):
    def get_discount(self):
        result = super().get_discount()
        if result:
            return result
        if 4 < self.average_grade <= 5:
            return self.semester_tax * 0.2
```

LSP - Liskov Substitution Principle - introduced by Barbara Liskov in a 1987

Derived types must be completely substitutable for their base types
Derived classes only extend functionalities of the base class
and must not remove base class behavior

Design Smell - Violations:

- If the code is checking the type of class
- Overridden methods change their behavior
- Override a method of the superclass by an empty method
- Base class depends on its subtypes

ISP - Interface Segregation Principle
Python doesn't have interfaces
A client should not depend on methods it does not use
Class Shape draws rectangle and circle
Class Circle or Rectangle implementing the Shape class must define the methods draw_rectangle() and draw_circle()

```
class Shape(ABC):  WRONG
    @abstractmethod
    def draw_rectangle(self):
        ...
    @abstractmethod
    def draw_circle(self):
        ...

class Shape(ABC):  CORRECT
    @abstractmethod
    def draw(self):
        ...
class Rectangle(Shape):
    def draw(self):
        pass
class Circle(Shape):
    def draw(self):
        pass
```

DIP - Dependency Inversion Principle
High-level modules should not depend on low-level modules. Both should depend on abstractions
Abstractions should not depend on details. Details should depend on abstractions

## 16.8. Iterators-and-Generators

Iterator object must implement two methods,
 `__iter__()` and `__next__()` (iterator protocol)
for loop is implemented as:

```
iter_obj = iter(iterable)
while True:
    try:
        element = next(iter_obj) # get the next item
        # do something with element
    except StopIteration:
        # if StopIteration is raised, break from loop
        break
```

for loop creates an iterator object (iter_obj) by calling iter() on the iterable

If a function contains at least one yield statement, it becomes a generator function
Generator saves memory
Yield - pauses the function saving all its states, and later continues from there on successive calls
 generator expression creates an anonymous generator function
generator expression is similar to that of a list comprehension - difference between them is that generator expression produces one item at a time

## 16.9. Decorators

```
return wrapper  # if wrapper() print(decorate). if wrapper print(decorate())
```

Python allows a nested function(closure) to access the outer scope (enclosure) of the enclosing function

closure is a critical concept in decorators

**Decorators** allow programmers to **modify** the behavior of a function or a class
Decorators allow us to **wrap** another function in order to extend the behavior of the wrapped function

To use a class as a decorator, we need to implement the __call__ method
__call__ method allows class instances to be called as functions

```
from functools import wraps
def vowel_filter(function):
    vowels = "aeiouy"
    @wraps(function)
    def wrapper():
        return [x for x in function() if x in vowels]
    return wrapper
@vowel_filter
def get_letters():
    return ["a", "b", "c", "d", "e"]
print(get_letters())
print(get_letters.__name__)  # wrapper if not @wraps(function) else get_letters

class Fibonacci: # fib = Fibonacci() - __call__- fib(7)
    def __init__(self):
        self.cache = {} # calculated staff is stored, so it is not recalculate
    def __call__(self, n): # allows class instances to be called as functions - fib(7) n=7
        if n not in self.cache:
            if n == 0:
                self.cache[0] = 0
            elif n == 1:
                self.cache[1] = 1
            else:
                self.cache[n] = self(n - 1) + self(n - 2)
                # self.cache[n] = self(n - 1)
                # self.cache[n] = self(n + 1) + self(n - 2)  # infinite recursion
                # self.cache[n] = self(n - 1) + self(n + 2)  # infinite recursion
        return self.cache[n]
# with self.cache = {} we did not calculate the staff calculated yet
fib = Fibonacci()
# print(fib(7))  # 7 can be entered because of n in def __call__(self, n):
# print(fib(7))  # 7 can be entered because of n in def __call__(self, n):
# print(fib(9))  # 9 can be entered because of n in def __call__(self, n):

for i in range(7+1):  # with self.cache = {} we do not recalculate the calculated staff
    print(fib(i))
    print(fib.cache)

print(fib.cache)  # {1: 1, 0: 0, 2: 1, 3: 2, 4: 3, 5: 5, 6: 8, 7: 13}
```

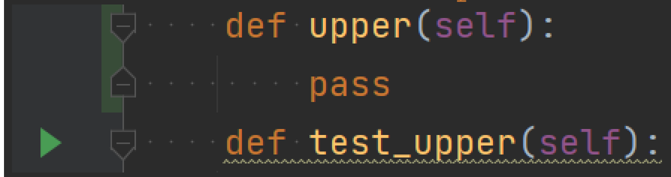
```python
# region Class Decorator
class func_logger:
    _logfile = 'out.log'
    def __init__(self, func):
        self.func = func
    def __call__(self, *args):
        log_string = self.func.__name__ + " was called"
        with open(self._logfile, 'a') as opened_file:
            opened_file.write(log_string + '\n')
        return self.func(*args)
        # return f"{self.func(*args)} - tested"- not working
@func_logger
def say_hi(name):
    print(f"Hi, {name}")
@func_logger
def say_bye(name):
    print(f"Bye, {name}")
say_hi("Peter")  #  Hi, Peter + out.log file containing self.func.__name__ + " was called"
say_bye("Peter") #  Bye, Peter +out.log file containing self.func.__name__ + " was called"
out.log file containing say_hi was called \n say_bye was called
# endregion

# region @repeat(3)
def repeat(n):
    def decorator(func):
        def wrapper(*args, **kwargs):
            for _ in range(n):
                func(*args, **kwargs)

        return wrapper

    return decorator
@repeat(3)
def say_hi():
    print("Hello")
say_hi()  # Hello\n Hello\n Hello\n
print(say_hi())  # Hello\n Hello\n Hello\n + None - because
say_hi() doesn't have return prints None
# endregion
```

Execution time measure

04_OOP/09_Decorators/Exercises/00_execution_time.py

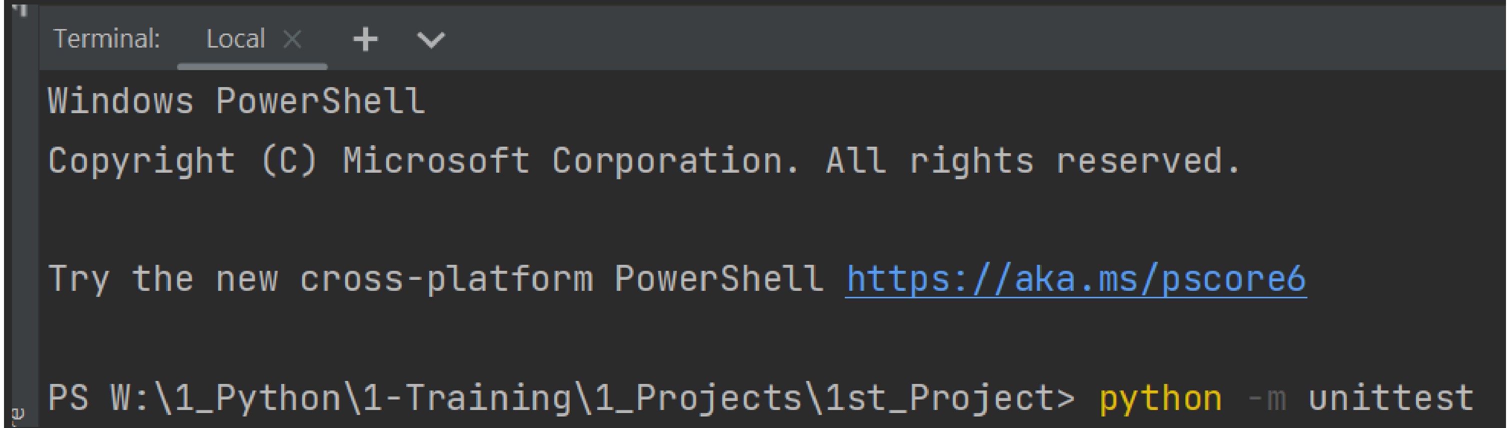
## 16.10. Testing

```
import unittest
from unittest import TestCase, main

def  test_something - is must start with test_ to run as a test in
unittest module - import unittest
```



```
    def upper(self):
        pass
    def test_upper(self):
```

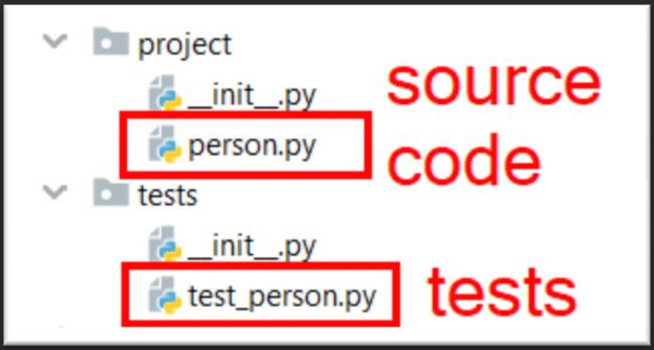
```
Test case - one green tick on judge - A set of conditions used to
determine if a system works correctly
Test suite(SET OF ROOMS) - collection of testcases - all tests in
judge - all green ticks
Manual testing
Automated testing - Unit testing, Integration testing, Many more
types of testing
Test fixture - (unittest, pytest- are Test fixture) a baseline for
running tests to ensure there is a fixed environment in which
tests are run so that results are repeatable

Mocking - A way to simulate a third party service in our app
Mocking - simulate the real behavior, we mock the services
and methods from other classes and simulate the real behavior


# Error region
Traceback (most recent call last):
  File "C:\Users\Happy\AppData\Local\Programs\PyCharm Community\plugins\python-
ce\helpers\pycharm\_jb_unittest_runner.py", line 38, in <module>
    sys.exit(main(argv=args, module=None, testRunner=unittestpy.TeamcityTestRunner,
car = Car("a", "b", 1, 4) - remove
car.make = ""- remove
print(car) - remove
# End Error region

# import unittest
from unittest import TestCase, main
# class SimpleTest(unittest.TestCase):
class SimpleTest(TestCase):
    def test_upper(self):
        string = 'foo'  # arrange
        result = string.upper()  # act
        expected_result = 'FOOa'
        self.assertEqual(expected_result, result)  # assert
# expected_result to be on left - Expected :FOOa Actual   :FOO
# Run by the following block of code
if __name__ == '__main__':
    # unittest.main()
    main()

class PersonTests(unittest.TestCase):
    def setUp(self):  # it’s part of unittest.TestCase
        self.person = Person("Luc", "Peterson", 25) # arrange
setUp method is called automaticaly immediately before each test method

with self.assertRaises(Exception) as ex:
    self.worker.work()
print(ex.exception)  # Not enough energy.
print(type(ex.exception))  # <class 'Exception'>
self.assertEqual('Not enough energy.', str(ex.exception))
04_OOP/10_Testing/Lab/01-Test-Worker/project/test_worker.py

self.assertFalse(self.cat.fed) ⇔ self.assertEqual(False, self.cat.fed)
self.assertNotIn(2, self.integer_list._IntegerList__data)
self.assertIn(2, self.integer_list._IntegerList__data)
```

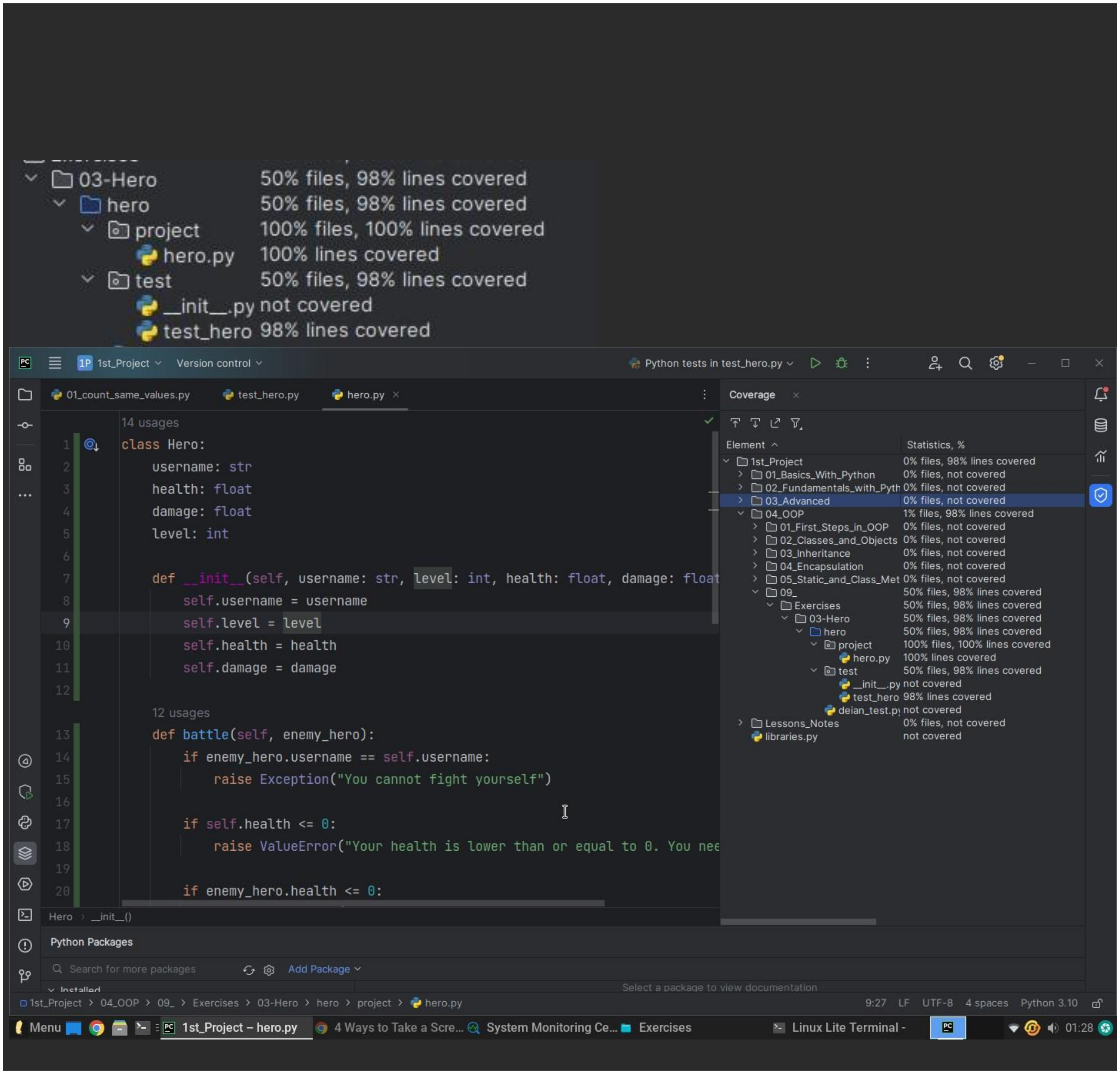


```
Terminal:  Local
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS W:\1_Python\1-Training\1_Projects\1st_Project> python -m unittest
```

It runs all tests in project. If we need to run just one file:

```
PS W:\1_Python\1-Training\1_Projects\1st_Project> python -m unittest test_main.py
```





```
03-Hero              50% files, 98% lines covered
  hero               50% files, 98% lines covered
    project          100% files, 100% lines covered
      hero.py        100% lines covered
    test             50% files, 98% lines covered
      __init__.py    not covered
      test_hero      98% lines covered
```

```python
14 usages
class Hero:
    username: str
    health: float
    damage: float
    level: int

    def __init__(self, username: str, level: int, health: float, damage: float
        self.username = username
        self.level = level
        self.health = health
        self.damage = damage

    12 usages
    def battle(self, enemy_hero):
        if enemy_hero.username == self.username:
            raise Exception("You cannot fight yourself")

        if self.health <= 0:
            raise ValueError("Your health is lower than or equal to 0. You nee

        if enemy_hero.health <= 0:
```

```
Element                          Statistics, %
1st_Project                      0% files, 98% lines covered
  01_Basics_With_Python          0% files, not covered
  02_Fundamentals_with_Pyth      0% files, not covered
  03_Advanced                    0% files, not covered
  04_OOP                         1% files, 98% lines covered
    01_First_Steps_in_OOP        0% files, not covered
    02_Classes_and_Objects       0% files, not covered
    03_Inheritance               0% files, not covered
    04_Encapsulation             0% files, not covered
    05_Static_and_Class_Met      0% files, not covered
    09_                          50% files, 98% lines covered
      Exercises                  50% files, 98% lines covered
        03-Hero                  50% files, 98% lines covered
          hero                   50% files, 98% lines covered
            project              100% files, 100% lines covered
              hero.py            100% lines covered
            test                 50% files, 98% lines covered
              __init__.py        not covered
              test_hero          98% lines covered
            deian_test.py        not covered
  Lessons_Notes                  0% files, not covered
  libraries.py                   not covered
```

## 16.11. Design-Patterns

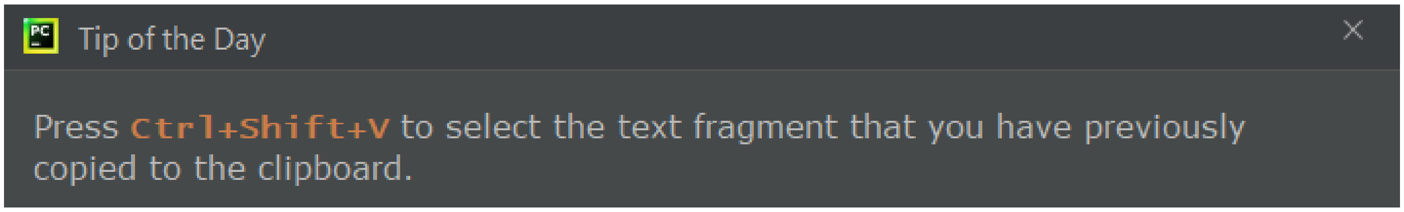
Gfsfshshg

## 16.12. "magic" or "dunder" methods

```
def food_can_eat(self) -> List[Food]: #  [Meat]
def feed(self, food: Food) -> (str, None):
    if type(food) not in self.food_can_eat:

def __getitem__(self, item):
    return self.people
def __getitem__(self, idx: int):
    return self.people[idx]
or  return f"Person {idx}: {self.people[idx]}"

MRO - Method Resolution Order - mro() -> list ; __mro__ -> tuple
teacher = Teacher()
print(Teacher.mro())
# [<class '__main__.Teacher'>, <class '__main__.Person'>, <class '__main__.Employee'>, <class 'object'>]
print(Teacher.__mro__)
# (<class '__main__.Teacher'>, <class '__main__.Person'>, <class '__main__.Employee'>, <class 'object'>)
print(teacher.__class__.__bases__[0].__name__)  # Person
print(teacher.__class__.__bases__[1].__name__)  # Employee
print(teacher.__class__.__name__)                # Teacher

__str__() - returns a printable string representation
__repr__() - returns a machine-readable representation
__doc__() - Provides a documentation of the object as a string
class MyClass:
    """This is MyClass."""
    def example(self):
        """This is the example module of MyClass."""
print(MyClass.__doc__) # This is MyClass.
print(MyClass.example.__doc__) # This is the example module of MyClass.


__dict__() is a dictionary containing a module's symbol table
class Dog:
    def __init__(self, name):
        self.name = name
x = Dog("Max")
print(x.__dict__) # {"name": "Max"}

def __reversed__(self):
def __len__(self):
def add_book(self…):
      if len(self) > self.books_limit: -> when is called within class
len(book_object)-> when is called outside class
self.assertEqual(3, self.store.__len__()) ⇔
self.assertEqual(3, len(self.store))
```

# 17. PyCharm

## 17.1. Shortcuts


Tip of the Day

Press Ctrl+Shift+V to select the text fragment that you have previously copied to the clipboard.

Choose lookup item and replace - Tab

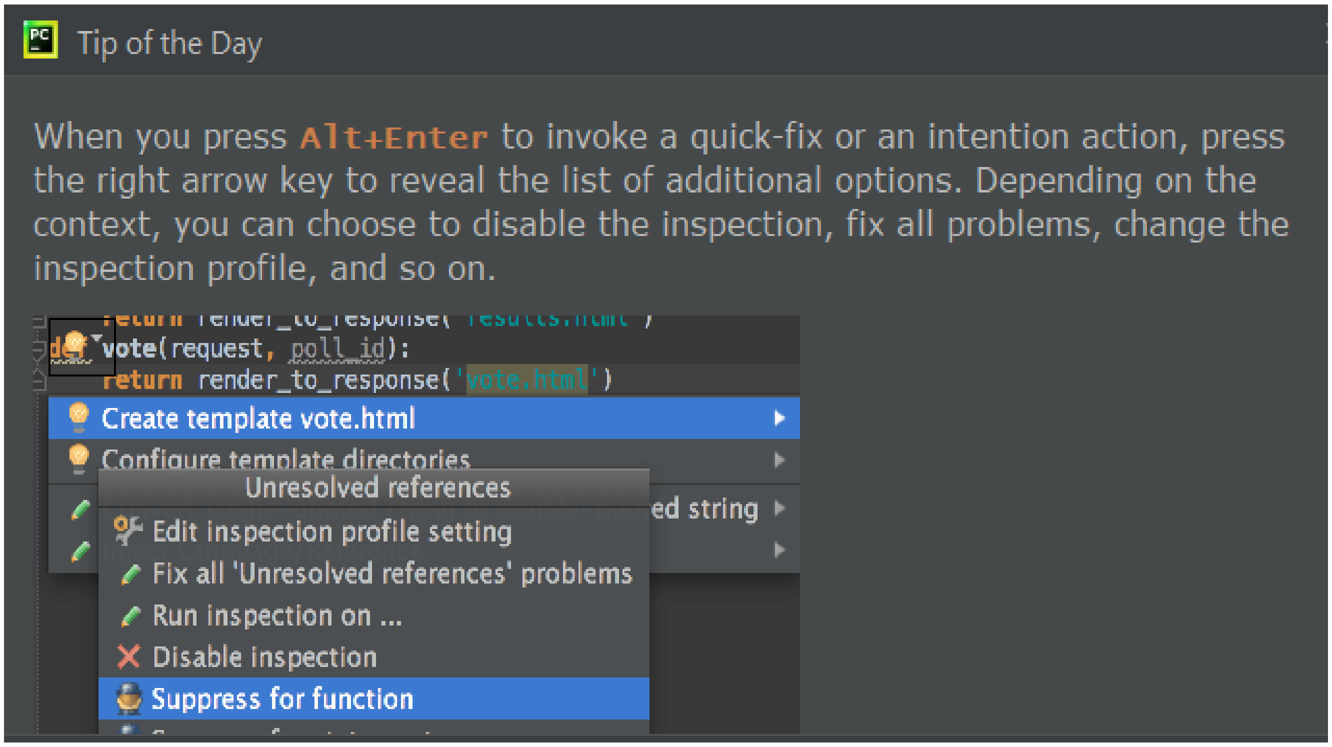
Word - Ctrl + =

Alt + click , Shift + end …..

Ctrl + Alt + M – turn selected code fragment into a method
Ctrl + H - Hierarhy Tree
Ctrl + M - Scroll to Center

Toggle Full Screen mode Ctrl+Shift+F1

```
prop + Tab @property (getter)
props + Tab @property and @???.setter (getter + setter)

class Bird(Animal):
    def __init__cursor(self) press Alt + Enter and result is:
class Bird(Animal):
    def __init__(self, name: str, weight: float):
        super().__init__(name, weight)
```


Tip of the Day

When you press Alt+Enter to invoke a quick-fix or an intention action, press the right arrow key to reveal the list of additional options. Depending on the context, you can choose to disable the inspection, fix all problems, change the inspection profile, and so on.

```
def vote(request, poll_id):
    return render_to_response('vote.html')
```

Create template vote.html
Configure template directories
Unresolved references
Edit inspection profile setting
Fix all 'Unresolved references' problems
Run inspection on ...
Disable inspection
Suppress for function

Successively press Alt+J to find and select the next occurrence of case-sensitively matching word or text range. To remove selection from the last selected occurrence, press Alt+Shift+J
After the second or any consecutive selection was added with Alt+J, you can skip it and select the next occurrence with F3. To return the selection to the lastly skipped occurrence, press Shift+F3
Press Ctrl+Alt+Shift+J to select all case-sensitively matching words or text ranges in the document.

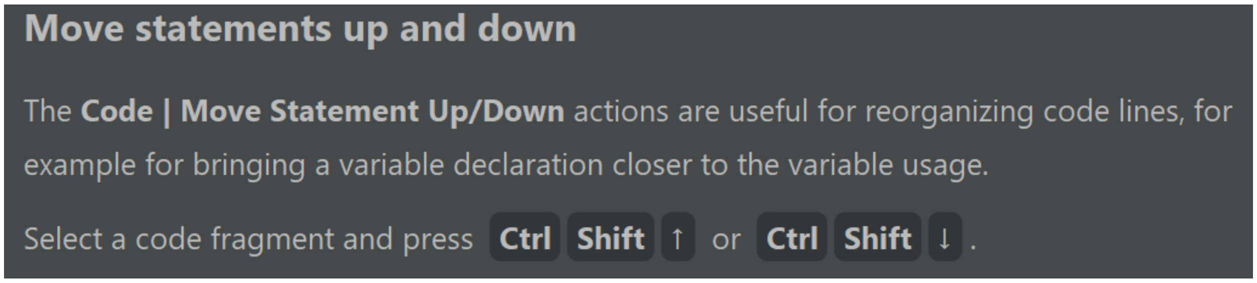
To redo Ctrl + Shift + Z
Ctrl+Alt+T  - To surround with (if or try or ....)
Duplicate current line or selection - **Ctrl + D**

**Alt + Left (Right) –** select Left (Right) tab - swetches betwin open files

To select multiple fragments (create multiple cursors) in the press and hold **Ctrl+Alt+Shift** and drag the mouse (Windows and Linux):

Press  Alt  F7  to quickly locate all occurrences of code referencing the symbol at the caret, no matter if the symbol is a part of a class, method, field, parameter, or another statement.

To toggle between the upper and lower case for the selected code fragment, press Ctrl+Shift+U


**Move statements up and down**

The **Code | Move Statement Up/Down** actions are useful for reorganizing code lines, for example for bringing a variable declaration closer to the variable usage.

Select a code fragment and press **Ctrl** **Shift** ↑ or **Ctrl** **Shift** ↓ .

**Ctrl + Enter** new raw while caret stays

Complete statement **Shift + Enter** (Ctrl + Shift + Enter)

Start new line with - **Ctrl + Shift + Enter** (Shift + Enter)

**Ctrl + Alt + L or Ctrl+** automatically format code with spaces and lines

Move Caret To Code Block End with - **Ctrl + right bracket ]**

Extend selection - **Ctrl+W**

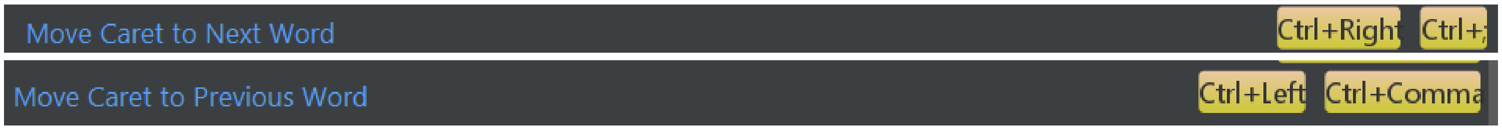
Decrease selection - **Ctrl+Shift+W or Ctrl+**

Select Several Rows To Be Simultaneously Edited - **Mouse Middle Click**

Duplicate current line or selection - **Ctrl + D**

Comment with line comment - **Ctrl + /**

New Python File - **Shift + Right Mouse Click**


Move Caret to Next Word Ctrl+Right Ctrl+;

Move Caret to Previous Word Ctrl+Left Ctrl+Comma

To scroll a file horizontally, **turn the mouse wheel** while keeping **shift** pressed

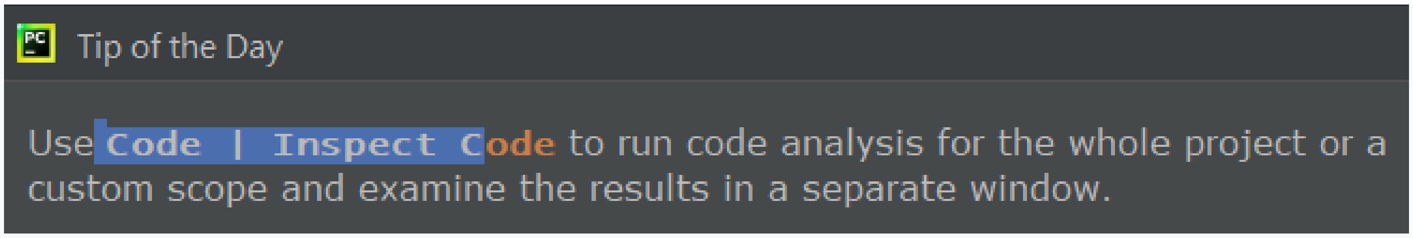
Press **Ctrl + Shift + V** to select the text fragment that you have previously copied to the clipboard

Press **Ctrl + Shift + mouse** to select the text word by word fragm

Press **Ctrl + ` - zoomit** command

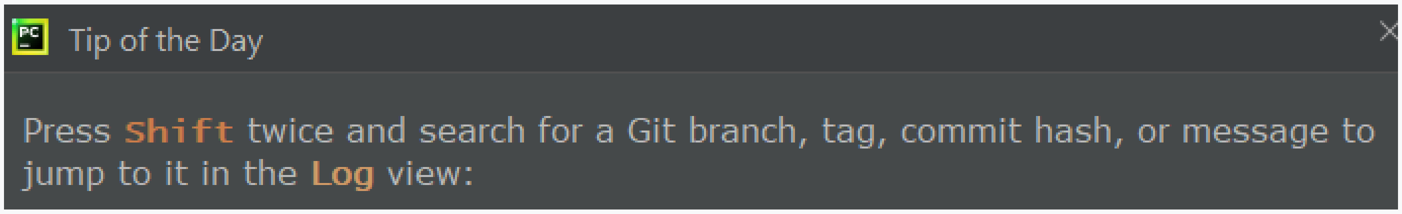
Mouse Middle Click or **Alt + shift + left mouse click** - select several rows to be simultaneously edited

Move Caret To Code Block End - **Ctrl +]**



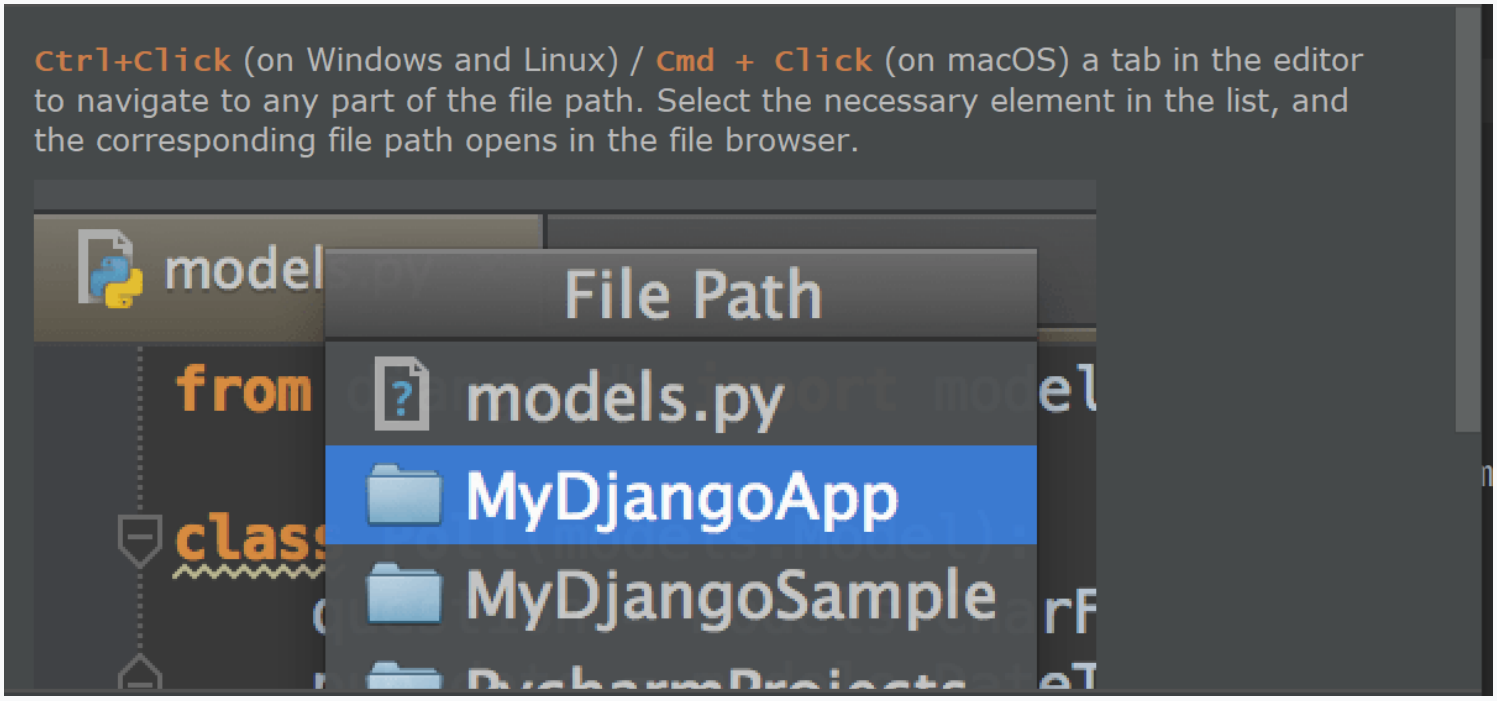
Tip of the Day

Use `Code | Inspect Code` to run code analysis for the whole project or a custom scope and examine the results in a separate window.

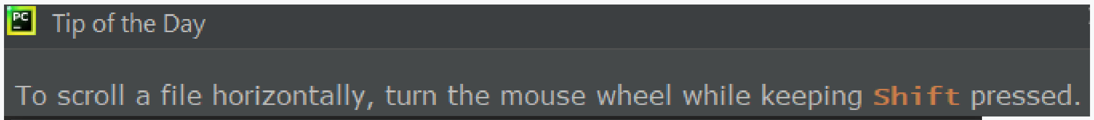


Tip of the Day

Press `Shift` twice and search for a Git branch, tag, commit hash, or message to jump to it in the `Log` view:

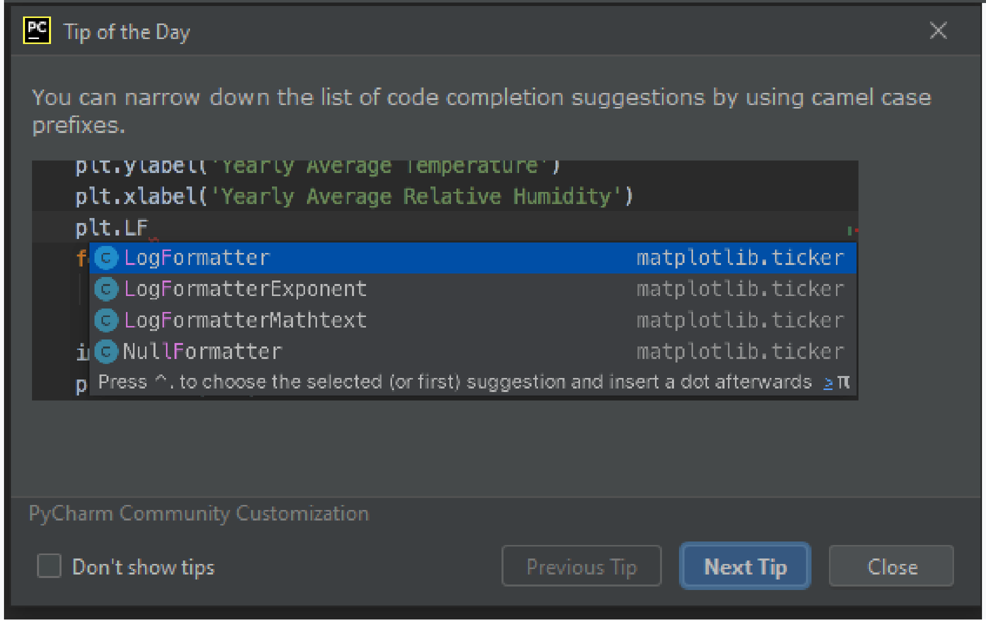


`Ctrl+Click` (on Windows and Linux) / `Cmd + Click` (on macOS) a tab in the editor to navigate to any part of the file path. Select the necessary element in the list, and the corresponding file path opens in the file browser.



Tip of the Day

To scroll a file horizontally, turn the mouse wheel while keeping `Shift` pressed.



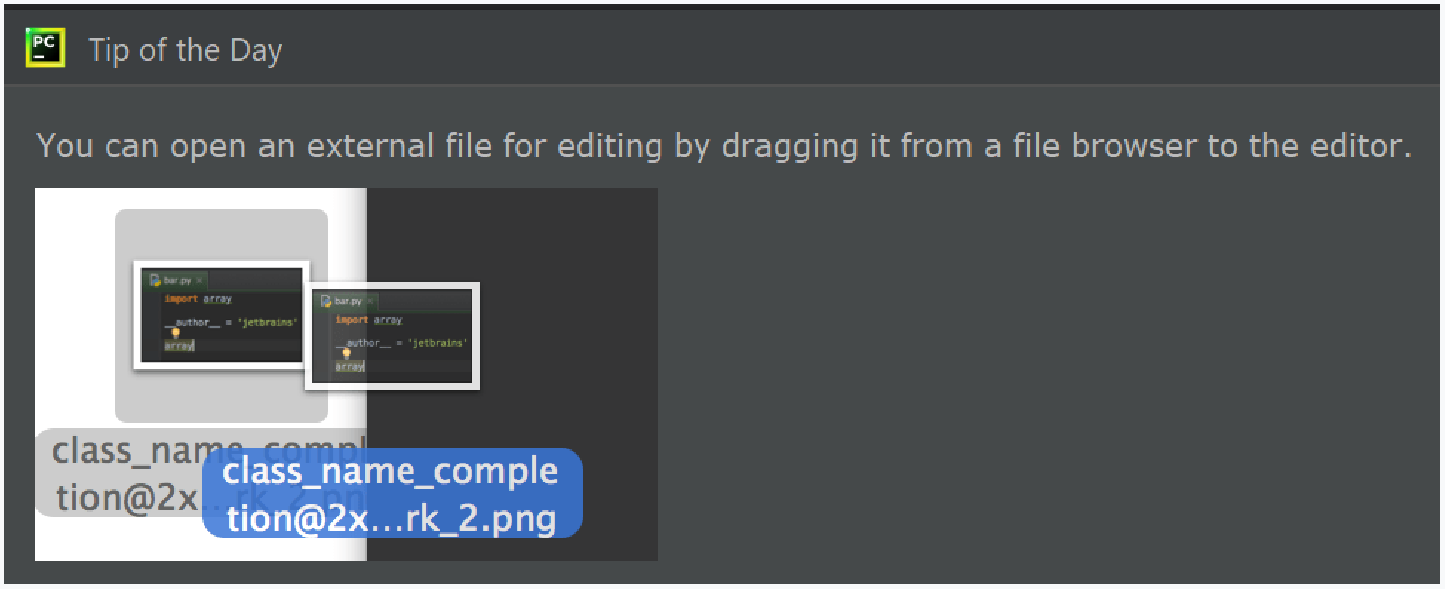
Tip of the Day

You can narrow down the list of code completion suggestions by using camel case prefixes.

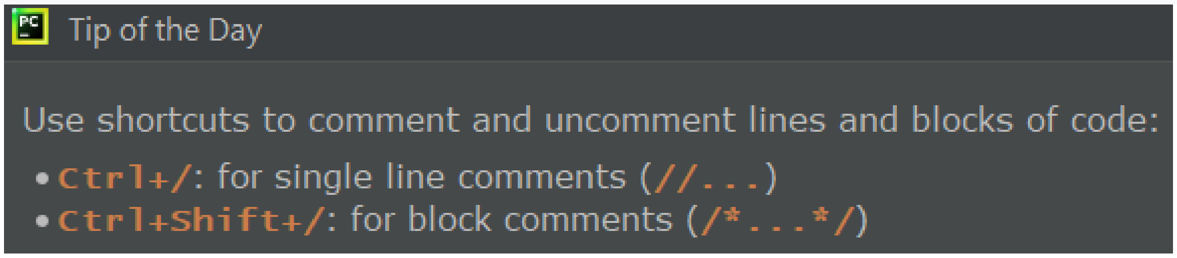
```
plt.ylabel('Yearly Average Temperature')
plt.xlabel('Yearly Average Relative Humidity')
plt.LF
   LogFormatter                         matplotlib.ticker
   LogFormatterExponent                 matplotlib.ticker
   LogFormatterMathtext                 matplotlib.ticker
   NullFormatter                        matplotlib.ticker
   Press ^. to choose the selected (or first) suggestion and insert a dot afterwards ≥ π
```

PyCharm Community Customization

☐ Don't show tips


Tip of the Day

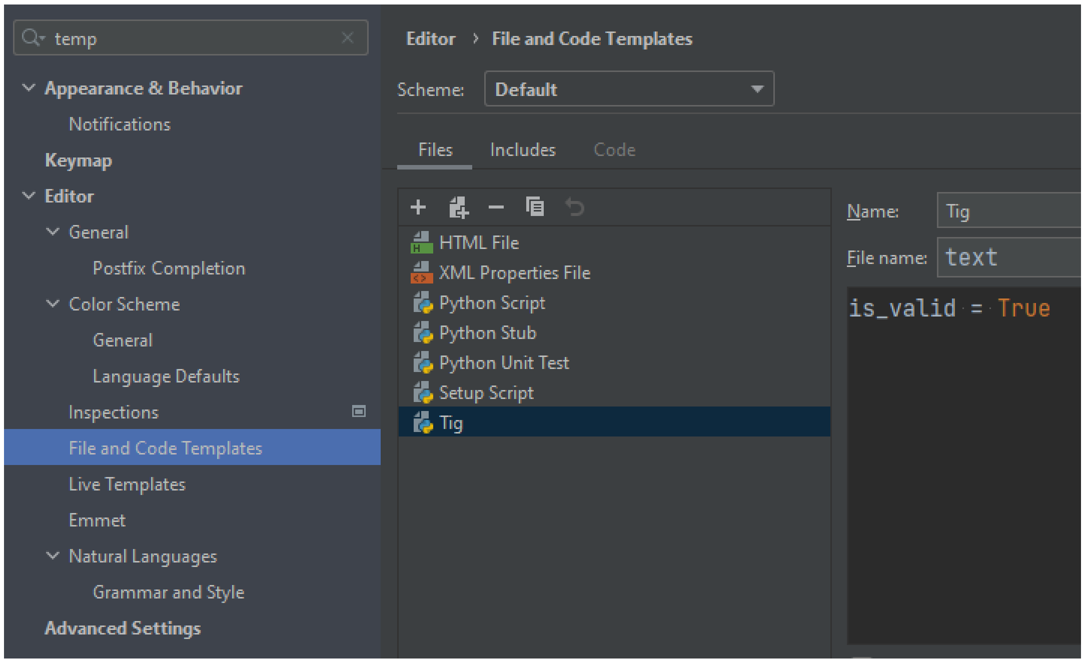
You can open an external file for editing by dragging it from a file browser to the editor.


Tip of the Day

Use shortcuts to comment and uncomment lines and blocks of code:

- Ctrl+/: for single line comments (//...)
- Ctrl+Shift+/: for block comments (/*...*/)

## 17.2. Settings



Scheme: Default

Name: Tig

File name: text

```
is_valid = True
```

## 18. Referenced

```
List, Set, Dictionary – mutable – referenced – it’s pointing to place
in memory, even if you change it. But if you reassigned it would
point to a different place in memory.
Int, str, float, tuple, frozenset – immutable – not referenced.
If you change it, it’ll point different place in memory

# All values in Python are references. What you need to worry about
is if a type is mutable. The basic numeric and string types, as well
as tuple and frozenset are immutable; names that are bound to an
object of one of those types can only be rebound, not mutated.

a = 10
b = a
a = 30 #  now  a = 30 but b remains 10

list1 = [10,20,30,40]
list2 = list1  #[10,20,30,40] list1 and list2 are one and the same object
list1 = [3,4] # this list1 is different from the list1 up, because
it’s reassigned ( it’s different object, written on a different place
in memory and it’s not possible to invoke list1 anymore)
# list1 ==> [3,4]
# list2 ==> [10,20,30,40]
-------------------------------------------------------------------------------
list1 = [10,20,30,40]
list2 = list1 #[10,20,30,40] - one and the same object
# change value of list 1 at a certain index say index 0
list1[0] = 500 # now list1 is the same object as list1 with changed
attribute value – mutated value
# If you check again the values of list1 and list2 you will be surprised.
#list1 ==> [500,20,30,40]
#list2 ==> [500,20,30,40]
--------------------------------------------------------------------------------
Set
a = {"a", "b", "c"}
b = a
a.add("d")
print(a)  # {'d', 'a', 'c', 'b'}
print(b)  # {'d', 'a', 'c', 'b'} set b is also changed
--------------------------------------------------------------------------------
Dictionary
a = {"a": 1, "b": 2, "c": 3}
b = a
a["a"] = 7
print(a)  # {'a': 7, 'b': 2, 'c': 3}
print(b)  # {'a': 7, 'b': 2, 'c': 3} dictionary b is also changed
```

```
----------------------------------------------------------------------
def add_number_12(num_seq):
    num_seq.append(12)
# no return, but list nums is modified
nums = [1, 2, 3]
print(nums)  # [1, 2, 3]
add_number_12(nums)  # no return, but list nums is modified, because lists
are referenced. num_seq and nums are pointing to one and the same place in
memory
print(nums)  # [1, 2, 3, 12]
-----------------------------------------------------------------------
def update_set(num_seq):
    num_seq.update("a", "s")
# no return, but set nums is modified
nums = {1, 2, 3}
print(nums)  # {1, 2, 3}
update_set(nums)  # no return, but set nums is modified, because sets are
referenced. num_seq and nums are pointing to one and the same place in memory
print(nums)  # {1, 2, 3, 's', 'a'}
---------------------------------------------------------------------
def update_dictionary(num_seq):
    num_seq.update({7: "s"})
# no return, but dictionary nums is modified
nums = {1: "z", 2: "x", 3: "e"}
print(nums)  # {1: 'z', 2: 'x', 3: 'e'}
update_dictionary(nums)  # no return, but dictionaries nums is modified, because
dictionaries are referenced. num_seq and nums are pointing to one and the same
place in memory
print(nums)  # {1: 'z', 2: 'x', 3: 'e', 7: 's'}
```

# 19. Regex

```
import re

([0]|[1-9][0-9]*) -> matches 0 but not 00 or 01
(?:   ) - does not capture/assign a group ID.
(   ) - group with ID. \+359([\s-])\d\1 -> \1 recall group with ID=1 ([\s-])
(?P<name>   ) - group with name. \+359(?P<sep>[\s-])\d(?P=sep) -> (?P=sep) recall
group (?P<sep>[\s-])
\b - only letters, nums and _, but not +-@....
([0]|[1-9]\d*)(\.\d+)? vs ([0]|[1-9]\d*\.?\d+)
\w [a-zA-Z0-9_] be careful for _ !!!!!!
(^|(?<=\s)) new line or space
(^|\s) new line or space, but add the space to the result

word = input()
pattern = rf'\b{word}\b'  #  rf''
```

## re.compile

```
email = input()
VALID_DOMAINS = (".com", ".bg", ".net", ".org")
regex_domain = re.compile(r'\.[a-z]+')
if regex_domain.findall(email)[-1] not in VALID_DOMAINS:
    print("Domain must be one of the following: .com, .bg, .org, .net")


word = input().casefold()
pattern = rf'\b{word}\b'  # -> how?
# matches = re.findall(rf'(^|(?<=\s)){word}($|(?=\s))', text) # will not much
HOW+?
matches = re.findall(rf'\b{word}\b', text)
print(len(matches))

if there is more than 1 group, do not use re.findall(), but re.finditer or
(?:...)
(?:...) means do not create a group ID, but act as a group

result = re.findall() # finds all, returns list
result = re.search() # finds first, not iterable, returns match type or None
re.match is anchored at the start
re.fullmatch is anchored at the start and end of the pattern
re.search is not anchored
result = re.match() # finds first, if it's at the beginning only, but
if re.search(pattern, names):
    print("yes")
else:
    print("no")


# pattern = r"\b(?P<Day>\d{2})([./-])(?P<Month>[A-Z][a-z][a-
z])\2(?P<Year>\d{4})\b"
pattern = r"\b(?P<Day>\d{2})(?P<sep>[\./-])(?P<Month>[A-Z][a-z][a-
z])(?P=sep)(?P<Year>\d{4})\b"
text1 = "13/Jul/1928, 10-Nov-1934, , 01/Jan-1951,f 25.Dec.1937 23/09/1973,
1/Feb/2016"
dates = re.finditer(pattern, text1)
# print(dates)
for date in dates:
    print(date)
    num_dict = date.groupdict()  # Match into dict
#     print(f"Day: {num_dict['Day']}, "  # calling value of key=Day from num_dict
#           f"Month: {num_dict['Month']}, "
#           f"Year: {num_dict['Year']}")
#     print(f"Day: {num[1]}, "  # group(1) returns the group(1) Match
#           f"Month: {num[3]}, "  # group(3) returns the group(3) Match
# #         f"Month: {num['Month']}" <=> f"Month: {num[3]}" -> both can be used
#           f"Year: {num['Year']}")  # group(Year)(4) returns the group(Year)(4)
Match
# #         f"Year: {num['Year']}" <=> f"Year: {num[4]}"
# #         -> both num['Year'] and num[4] can be used, because group4 is named
Year
    print(f"Day: {num_dict['Day']}, Month: {num_dict['Month']}, Year:
{num_dict['Year']}")
    print(f"Day: {date['Day']}, Month: {date['Month']}, Year: {date['Year']}")
    print(f"Day: {date[1]}, Month: {date.group(3)}, Year: {date[4]}")
    # !!! use date.group(1) or date.group('Day'), but not date[1] or date['Day'],
    # because it could NOT be available in next release!!!
    print(date.group())  # group(0) returns the whole Match
    print(date.group(1))  # group(1) returns Day
    print(date.group('Month'))  # group(2) returns 'Month'
    print(date.groups())  # all groups as tuple ('13', '/', 'Jul', '1928')
# dates1 = re.findall(pattern, text1)
# print(dates1)
# for date in dates1:
#     print(f"Day: {date[0]}, Month: {date[2]}, Year: {date[3]}")
dates = re.match(pattern, text1)  # MATCH IS NOT ITERABLE, searches at the
BEGINNING ONLY
print(dates)  # match & search are same type, but the scope
print(type(dates))
print(dates.groupdict())
dates = re.search(pattern, text1)  # returns the same as match, BUT in ALL ROWS
print(dates)  # match & search are same type, but the scope
print(type(dates))
print(dates.groupdict())

txt = "The rain in Spain"
x = re.sub(r"\s", "9", txt, 2)  # substitute(replace)
print(x)

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

text1 = input()
text2 = input()
text3 = input()
pattern = r"\+359 2 \d{3} \d{4}\b|\+359-2-\d{3}-\d{4}\b"
num1 = re.findall(pattern, text1)  # more time
num2 = re.findall(pattern, text2)  # more time
num3 = re.findall(pattern, text2)  # more time
regex_pattern = re.compile(pattern)
```

```
num11 = regex_pattern.findall(text1)  # faster
num12 = regex_pattern.findall(text2)  # faster
num13 = regex_pattern.findall(text3)  # faster

print(*res_list, sep=', ')
print(str_res[:-2])
```

## 20. Sets - кортежи(tuple) и множества(set)

```
Unique unordered collection
Sets can be used to perform mathematical set operations
(union, intersection, symmetric difference, etc.)

usernames = set()

knight_attacks = len({(i + di, j + dj) for di, dj in positions if (i
+ di, j + dj) in knights})
knight_attacks = len({(i + di, j + dj) for di, dj in
positions}.intersection(knights))
faster than the upper row due to intersection.
intersection is faster than if !!!


sorted(set(crafted))] => return list
[print(f"{toy}: {crafted.count(toy)}") for toy in sorted(set(crafted))]
1st_Project\03_Advanced\03_Stacks_Queues_Tuples_and_Sets_Exercise\Exercises\05_santas_present_factory_a.py

A set is a collection which is unordered and unindexed.
No repeated symbols.
Sets are written with braces curly brackets
text = "Hhello"
set_text = set(text)
print(text)  # Hhello
print(set_text) # {'H', 'o', 'e', 'l', 'h'}


a = set([1, 2, 3, 4])
b = set([3, 4, 5, 6])
print(a | b)  # Union -> {1, 2, 3, 4, 5, 6}
print(a & b)  # Intersection -> {3, 4}
print(a < b)  # Subset -> False
print(a > b)  # Superset -> False
print(a - b)  # Difference -> {1, 2}
print(a ^ b)  # Symmetric Difference -> {1, 2, 5, 6}

a.union(b)                  # Equivalent to a | b
print(a.union(b))           # {1, 2, 3, 4, 5, 6}
print(a)                    # {1, 2, 3, 4}
a.intersection(b)           # Equivalent to a & b
a.issubset(b)               # Equivalent to a <= b
a.issuperset(b)             # Equivalent to a >= b
a.difference(b)             # Equivalent to a - b
a.symmetric_difference(b)   # Equivalent to a ^ b

a.update() updates the current set, by adding items from another set
```

```python
def isdisjoint(self, *args, **kwargs):
    """ Return True if two sets have a null intersection. """
```

The discard() method removes the specified item from the set. This method is different from the remove() method, because the remove() method will raise an error if the specified item does not exist, and the discard() method will not.

## 21. Shortcuts

See PyCharm chapter

**Word:**

Ctrl + F6 – switch between open Word docs

Alt+ F7 - starts spell check in MS Word

## 22. Slicing

```
[::] no beginning and end
a = "2371"
x = a[::-1]  # 1732
x = a[:-1]  # 1
y = a[::-2]  # 13
z = a[::2]   # 27
z = a[:2]   # 23
b = list(a)  # ['2', '3', '7', '1']
a = "0123456789"
x1 = a[1::2]   # 13579
x2 = a[::2]   # 02468
x3 = a[::3]   # 0369
c = list(a)
c.extend(b)
d = a[1:7]
a = [1, 2, 3, 4, 5, 6, 7]
b = a[-5:-2]  # new not referent
b = a[-3:-6:-2]  # [5, 3]
b = a[:]  # new not referent
b = a[::]  # new not referent
txt = "Welcome To My World"
x = txt[-5::]  # World
x = txt[-5:]  # World
x = txt[14:]  # World
x = txt[slice(-5, len(txt), 1)]  # World
x = txt[slice(-5, 19, 1)]  # World
x = txt[-5::2]  # Wrd
x = txt[-5:2]  # empty because 2 = -17
x = txt[-17:9]  # lcome T
x = txt
print(x)

# removing elements in the middle of the list
a_nums = a_nums[:left_idx] + a_nums[right_idx + 1:]
print(a_nums)
# =>
for i in range(idx + value, idx - value - 1, -1):
    b_nums.pop(i)
print(b_nums)
# =>
del c_nums[left_idx:right_idx + 1]
print(c_nums)
```

## 23. Symbol names

| | |
|---|---|
| = | equal |
| { | open brace |
| ( ) | parenthesis |
| [ | open bracket |
| % | percent |
| ? | Question Mark |
| \| | pipe or bar |
| ! | "bang", "exclamation point" |
| @ | "at", and rarely, "strudel" |
| # | "crunch", "hash", "pound", and rarely, "octothorpe" |
| ^ | "circumflex", "hat", "chapeau" |
| & | "ampersand", "and" |
| * | "splat", "star", "asterisk", "times" (as in multiplication) |
| _ | "underscore" |
| - | "hyphen", "dash", "minus sign" |
| . | "dot", "period" |
| , | "comma" |
| : | "colon" |
| ; | "semi-colon" |
| / | "slash" |
| \ | "backslash" |
| ~ | "twiddle", also "squiggle", or more correctly, "tilde" |
| ' | "tick", "quote", "apostrophe" |
| " " | double-quote" |
| ` | "backtick", "backquote" |
| < | "less-than", "left angle bracket" |
| > | "greater-than", "right angle bracket" |

## 24. Text

```
if email.index("@") < 5:

f"""{self.name} Library does not have {book_author}'s "{book_title}"."""

print(chr(87))   # W
print(ord('a'))   # 97

name = 'Test'
print('name is: {}'.format(name))   # name is: Test
print(f'name is: {name}')           # name is: Test
# print() is function
# .format(name) is method
Python integers are immutable
Python floats are immutable
Python strings are immutable
This means that once a string is created,
it is not possible to modify it
name = 'George'
name[0] = 'P' # Error не може да променим G
print(name)    # George
name = 'Ime'   # заделя друго място в паметта
различно  от мястото за George
print(name)    # Ime
name = 4       # заделя трето място в паметта
print(name)    # 4
string interpolation are string literals(буквален)
that allow embedded(вградени) expressions


name = 'Test New'
print(name[:2])     # Te
print(name[:3])     # Tes
print(name[ ])      # Error
print(name[3:])     # t New
print(name[2:6])    # st N


# creating new text with removed chars in the middle of the list
a_nums = a_nums[:left_idx] + a_nums[right_idx + 1:]
print(a_nums)
```

```
txt = "Welcome To My World"
# x = txt.casefold()  # stronger than lower()
# x = txt.lower()
# x = txt.count('l', 3, 19)  # string.count(value, start, end)
"welcome".find("com")  # 3 string.find(value, start, end)

x = "bob".center(10, '@')  # @@@bob@@@@
x = txt.encode()  # string.encode(encoding=encoding, errors=errors)
x = txt.endswith("my world.", 5, 11)  # True or False
print("H\te\tl\tl\to".expandtabs(3))  # H  e  l  l  o
print("H\te\tl\tl\to".expandtabs(5))  # H    e    l    l    o
x = "welcome".isascii()  # True
x = "wow_83".isidentifier()  # True
x = "lo!\nAre".isprintable()  # False

print(isinstance(11, float))  # False
print(isinstance(11.0, int))  # False
print(isinstance(11, float) or isinstance(11, int))  # True
print(isinstance(11.0, float) or isinstance(11.0, int))  # True
str_1 = "teststring12"
x = str_1.isalnum()  # True - "alnum" - alpha numeric
y = str_1.isalpha()  # False
z = str_1.isdigit() # False Exponents, like ², are also considered to
be a digit
a = '-1'.isdecimal()   # False 0-9
b = '3/4'.isnumeric()  # False
c = '¾'.isnumeric()    # True 0-9 like ² and ¾
d = "0.7"
print('0.7'.isnumeric())  # False
print("0.7".isdigit())  # False
print(isinstance("0.7", float))  # False
print(isinstance(0.7, float))  # True
print(d.isnumeric())  # False - AttributeError if d=0.7 instead "0.7"
print(d.isdigit())  # False - AttributeError if d=0.7 instead "0.7"
print(isinstance(d, float))  # False

txt = "     banana     "
print(txt.lstrip())  # "banana     "
print(txt.rstrip())  # "     banana"
print(txt.strip())   # "banana"
print(txt)           # "     banana     "
```

## 25. Time

```
03_Advanced\04_Multidimensional_Lists\Recapitulate\Exercises_2\03_knight_game.py
knight_attacks=len({(i+di,j+dj)for di,dj in positions if (i+di,j+dj) in knights})
knight_attacks=len({(i+di,j+dj)for di,dj in positions}.intersection(knights))
# row using intersection is faster than row with if

time.sleep(2)    #-> wait for 2 seconds (secs)

# region datetime timedelta, strptime, strftime

from datetime import datetime, timedelta

# input_time = "8:00:00"
input_time = "2023:8:00:00:17"  # Month is omitted
current_time = datetime.strptime(input_time, "%Y:%H:%M:%S:%d")
current_time += timedelta(seconds=7)
# class datetime.timedelta(days=0, seconds=0, microseconds=0, 
milliseconds=0, minutes=0, hours=0, weeks=0)
print(current_time.strftime("p[%H:%M:%S{q"))  # p[08:00:07{q
print(current_time.strftime("%H:%M:%S-(%d/%Y)"))  # 08:00:07-
(17/2023) - Month is omitted

# endregion

# region Diff = End_time - Start_time

import time

start_time = time.time()
test_list = [x for x in range(100000)]
test_list = list(range(100000))
while test_list:
    test_list.pop()
diff = time.time() - start_time
print(diff)
start_time = time.time()
test_list = [x for x in range(100000)]
while test_list:
    test_list.pop(0)
diff = time.time() - start_time
print(diff)

# endregion
```

## 26. Tuples - кортежи(tuple) и множества(set)

```
t = (1, )
t = (1, 2, 3)
t = 1, 2, 3
nums = tuple(int(x) for x in input().split())

two available tuple methods
count and index
```

Tuples are **immutable objects,** but the **objects**, inside the tuples, **are mutable**

```

nums = [1, 2]
my_tuple = (nums, 7, 9)  # tuple are immutable but variables are
mutable
print(my_tuple)  # ([1, 2], 7, 9)
nums.append(3)  # change NUMS in tuple!!! It will not work after
redefining it in the next row
nums = [1, 2, 29]  # does not change NUMS in tuple!!! create new NUMS
different from NUMS in tuple
 print(my_tuple)  # ([1, 2, 3], 7, 9) -> variables inside the tuple
are mutable
my_tuple[0][2] = 12  # if we want to access NUMS in tuple again
my_tuple[0].append(43)  # if we want to access NUMS in tuple again
print(my_tuple)  # ([1, 2, 12, 43], 7, 9) -> variables inside the
tuple are mutable
nums.append(23)  # [1, 2, 29, 23]
print(nums)  # [1, 2, 29, 23]
print(my_tuple)  # ([1, 2, 12, 43], 7, 9) -> variables inside the
tuple are mutable
```

# 27. Queues and Stacks

```python
nums = deque([0, 1, 2, 3])  # deque([0, 1, 2, 3])
print(nums)  # deque([0, 1, 2, 3])
nums1 = deque()
for i in range(5):
    nums1.appendleft(i)
print(nums1)  # deque([4, 3, 2, 1, 0])
```

## 28. ZZZ Other

### 28.1. symbol-name-list

```
=      equal
{      open brace      curly bracket
( )    parenthesis
[      open bracket
%      percent
?    Question Mark
|    pipe or bar
! "bang", "exclamation point"
@ "at", and rarely, "strudel"
# "crunch", "hash", "pound", and rarely, "octothorpe"
^ "circumflex", "hat", "chapeau"
& "ampersand", "and"
* "splat", "star", "asterisk", "times" (as in multiplication)
_ "underscore"
- "hyphen", "dash", "minus sign"
. "dot", "period"
, "comma"
: "colon"
; "semi-colon"
/ "slash"
\      "backslash"
~      "twiddle", also "squiggle", or more correctly, "tilde"
' ‵ ′       "tick", "quote", "apostrophe"  it′s ' ′  ‵  ′ ‵′′¬
"  ‶    "double-quote"
`      "backtick", "backquote"
< "less-than", "left angle bracket"
> "greater-than", "right angle bracket"
¬
```

https://en.wikipedia.org/wiki/List_of_typographical_symbols_and_punctuation_marks
https://onlymyenglish.com/symbol-name-list-english/

## 28.2. Common Mistakes

```
for meal_name, qty in client.ordered_meals.items()

ZeroDivisionError:

for attr, value in kwargs.items():
    setattr(movie, attr, value)

float problems
round_half_correctly.py

raise (return)

return f"{i + 1} astronauts participated in collecting items."

band = self.find_object(self.bands, "name", band_name)
~~musician = self.find_object(band.members, "name", musician_name)~~
if not band upper row will raise AttributeError: 'NoneType' object has no attribute 'members'
if not band: first must be checked that band exists
    raise Exception(f"{band_name} isn't a band!")
musician = self.find_object(band.members, "name", musician_name)
```

## 28.3. If... Else ... replacement

```
even_set.add(num) if num % 2 == 0 else odd_set.add(num)

map_function = {
    1: lambda x: numbers.append(x[1]),
    2: lambda x: numbers.pop() if numbers else None,
    3: lambda x: print(max(numbers)) if numbers else None,
    4: lambda x: print(min(numbers)) if numbers else None,
}  # There must be lambda x: on each Key: Value !!!
for _ in range(int(input())):
    command = [int(x) for x in input().split()]
    # map_function[command[0]](command)
    if map_function.get(command[0]):
        map_function[command[0]](command)
    else:
        print("anything")
    # try:
    #     map_function[command[0]](command)
    # except KeyError:
    #     print("anything")
----------------------------------------------------------------------
from functools import reduce
map_function = {
    '+': lambda x: reduce(lambda a, b: a + b, x),
    '-': lambda x: reduce(lambda a, b: a - b, x),
    '/': lambda x: int(reduce(lambda a, b: a / b, x)),
    # '/': lambda x: reduce(lambda a, b: a + b if a == 0 or b == 0 else a / b,
x),
    '*': lambda x: reduce(lambda a, b: a * b, x),
}
for el in data:
    if el in map_function:
        res = map_function[el](temp_list)
    else:
        temp_list.append(int(el))
---------------------------------------------------------------
map_func = {
    "Add First": lambda x: set1.update(x),
    "Add Second": lambda x: set2.update(x),
    "Remove First": lambda x: set1.difference_update(x),
    "Remove Second": lambda x: set2.difference_update(x),
    # "Check Subset": lambda x: print(set1.issubset(set2) or set2.issubset(set1))
    "Check Subset": lambda x: print("True") if set1.issubset(set2) or
set2.issubset(set1) else print("False")
}
for _ in range(int(input())):
    action1, action2, *info = input().split()

    map_func[action1 + ' ' + action2](map(int, info))
```