13 - Programming with Python

Exercises for Module "Programming with Python"

▶ Solution 1: Working with Lists

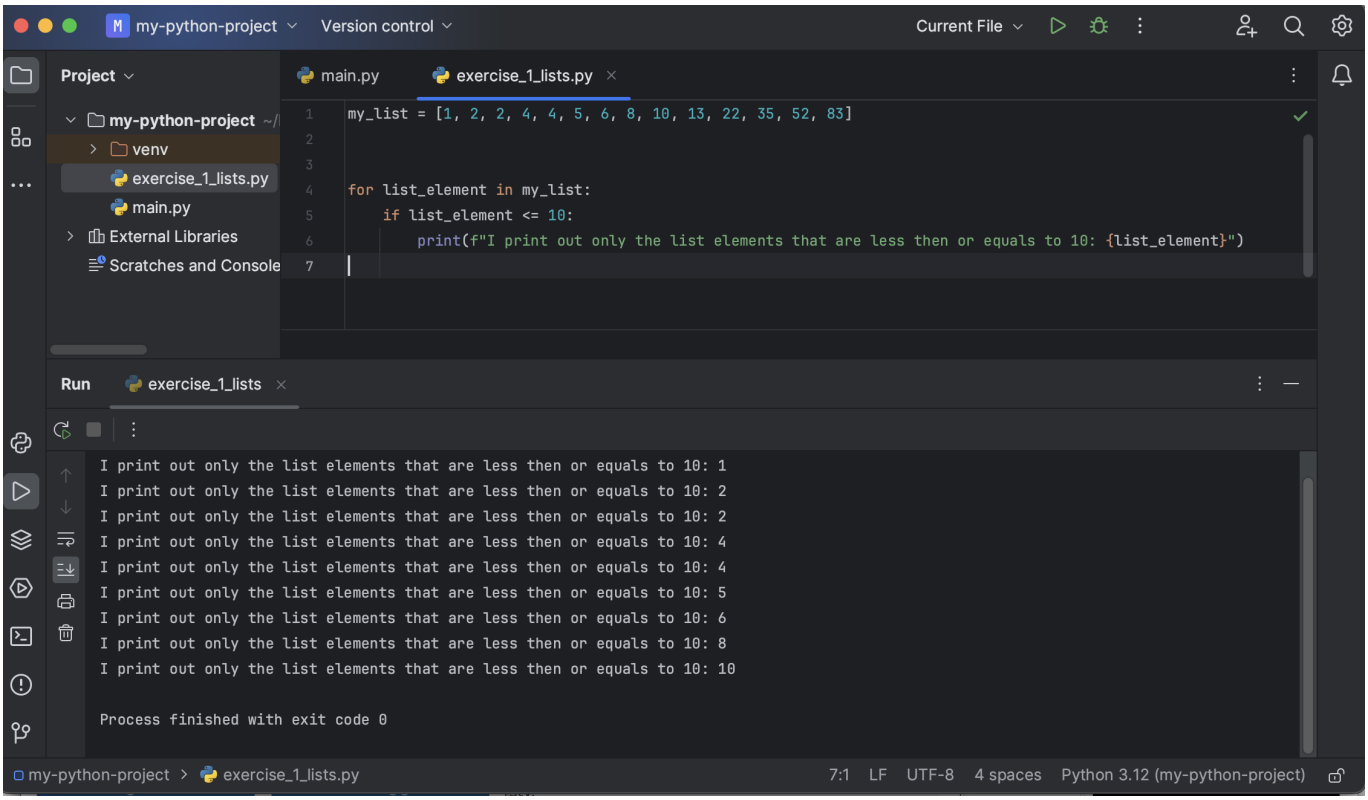
> EXERCISE 1: Working with Lists Using the following list:

my_list = [1, 2, 2, 4, 4, 5, 6, 8, 10, 13, 22, 35, 52, 83]

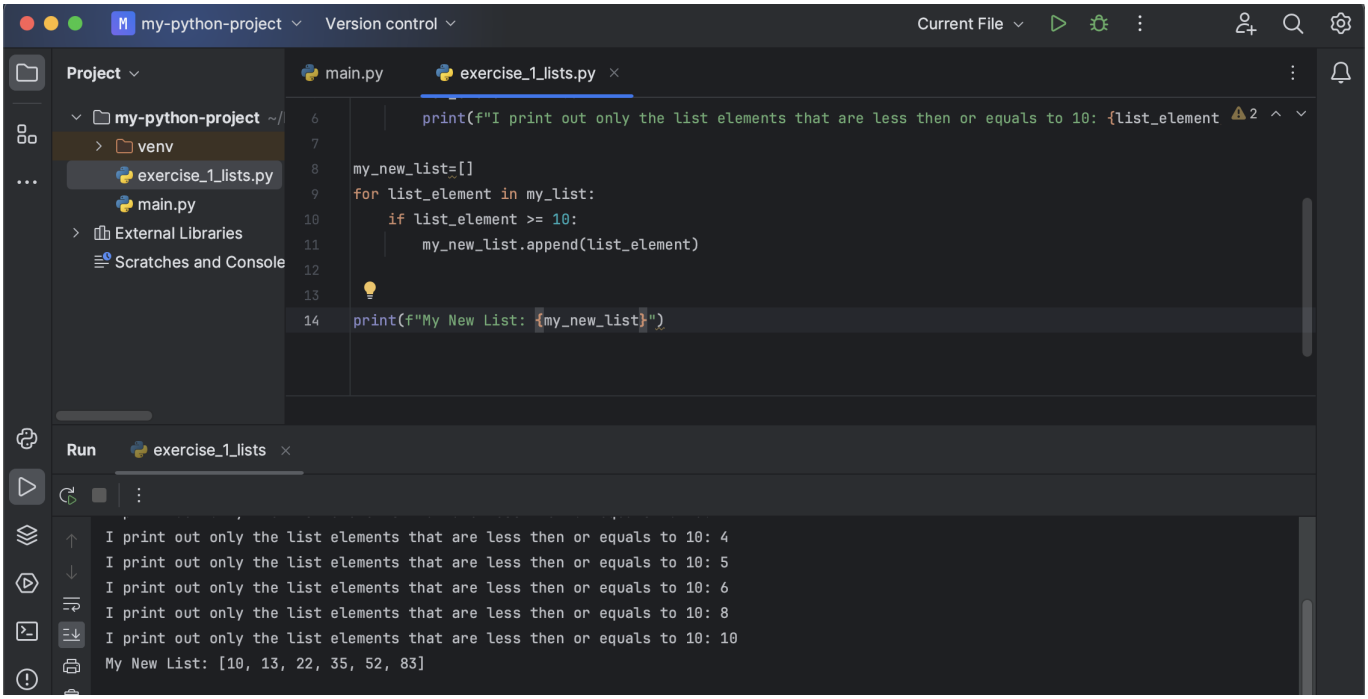
- Write a program that prints out all the elements of the list that are higher than or equal 10.
- Instead of printing the elements one by one, make a new list that has all the elements higher than or equal 10 from this list in it and print out this new list.
- Ask the user for a number as input and print a list that contains only those elements from my_list that are higher than the number given by the user.

exercise_1_lists.py

```
my_list = [1, 2, 2, 4, 4, 5, 6, 8, 10, 13, 22, 35, 52, 83]


for list_element in my_list:
    if list_element <= 10:
        print(f"I print out only the list elements that are less then or
equals to 10: {list_element}")
```

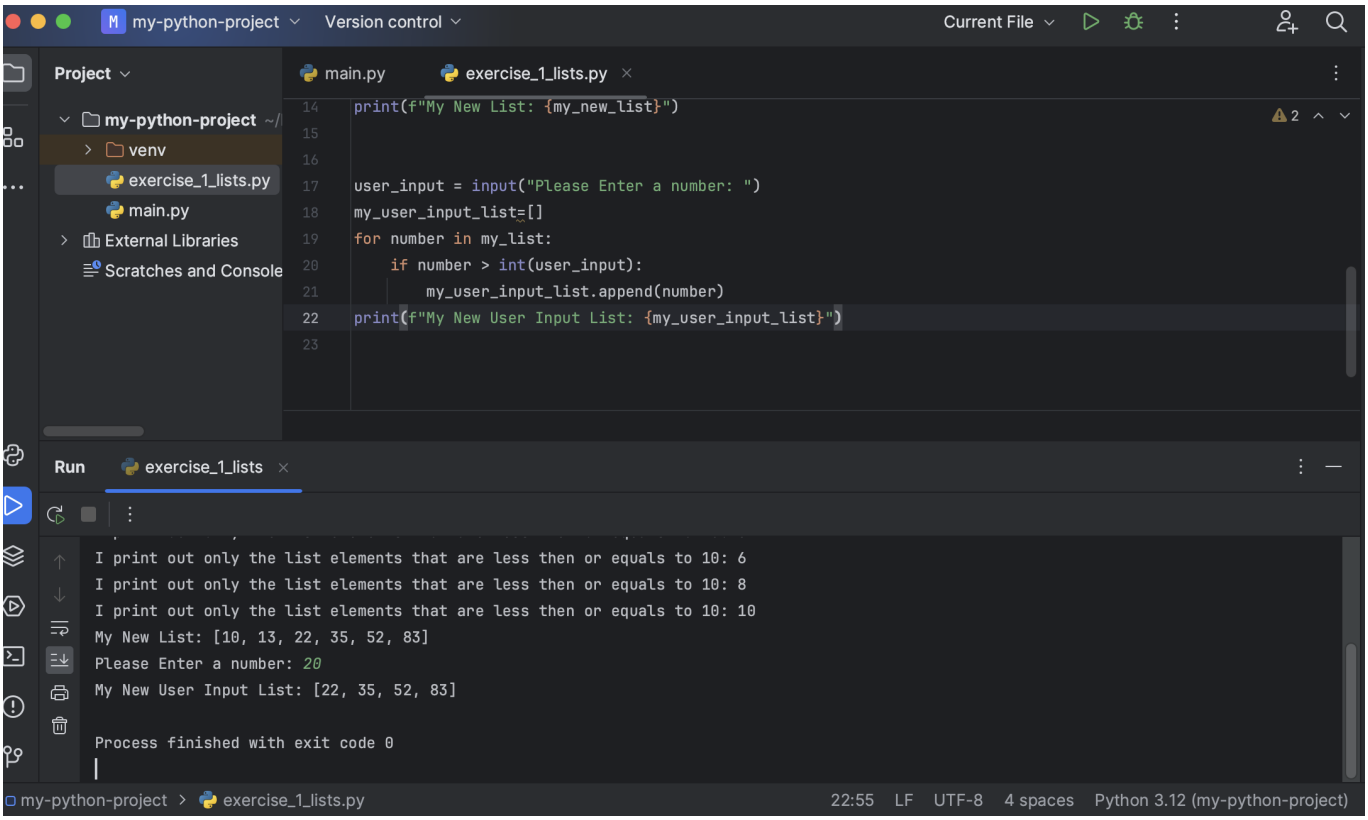


```python
my_list = [1, 2, 2, 4, 4, 5, 6, 8, 10, 13, 22, 35, 52, 83]


for list_element in my_list:
    if list_element <= 10:
        print(f"I print out only the list elements that are less then or
equals to 10: {list_element}")

my_new_list=[]
for list_element in my_list:
    if list_element >= 10:
        my_new_list.append(list_element)


print(f"My New List: {my_new_list}")
```




```python
my_list = [1, 2, 2, 4, 4, 5, 6, 8, 10, 13, 22, 35, 52, 83]


for list_element in my_list:
    if list_element <= 10:
        print(f"I print out only the list elements that are less then or
equals to 10: {list_element}")

my_new_list=[]
for list_element in my_list:
    if list_element >= 10:
        my_new_list.append(list_element)


print(f"My New List: {my_new_list}")
```

```python
user_input = input("Please Enter a number: ")
my_user_input_list=[]
for number in my_list:
    if number > int(user_input):
        my_user_input_list.append(number)
print(f"My New User Input List: {my_user_input_list}")
```



▶ Solution 2: Working with Dictionaries

> EXERCISE 2: Working with Dictionaries Using the following dictionary:

```python
employee = {
  "name": "Tim",
  "age": 30,
  "birthday": "1990-03-10",
  "job": "DevOps Engineer"
}
```

Write a Python Script that:

- Updates the job to Software Engineer
- Removes the age key from the dictionary
- Loops through the dictionary and prints the key:value pairs one by one

Using the following 2 dictionaries:

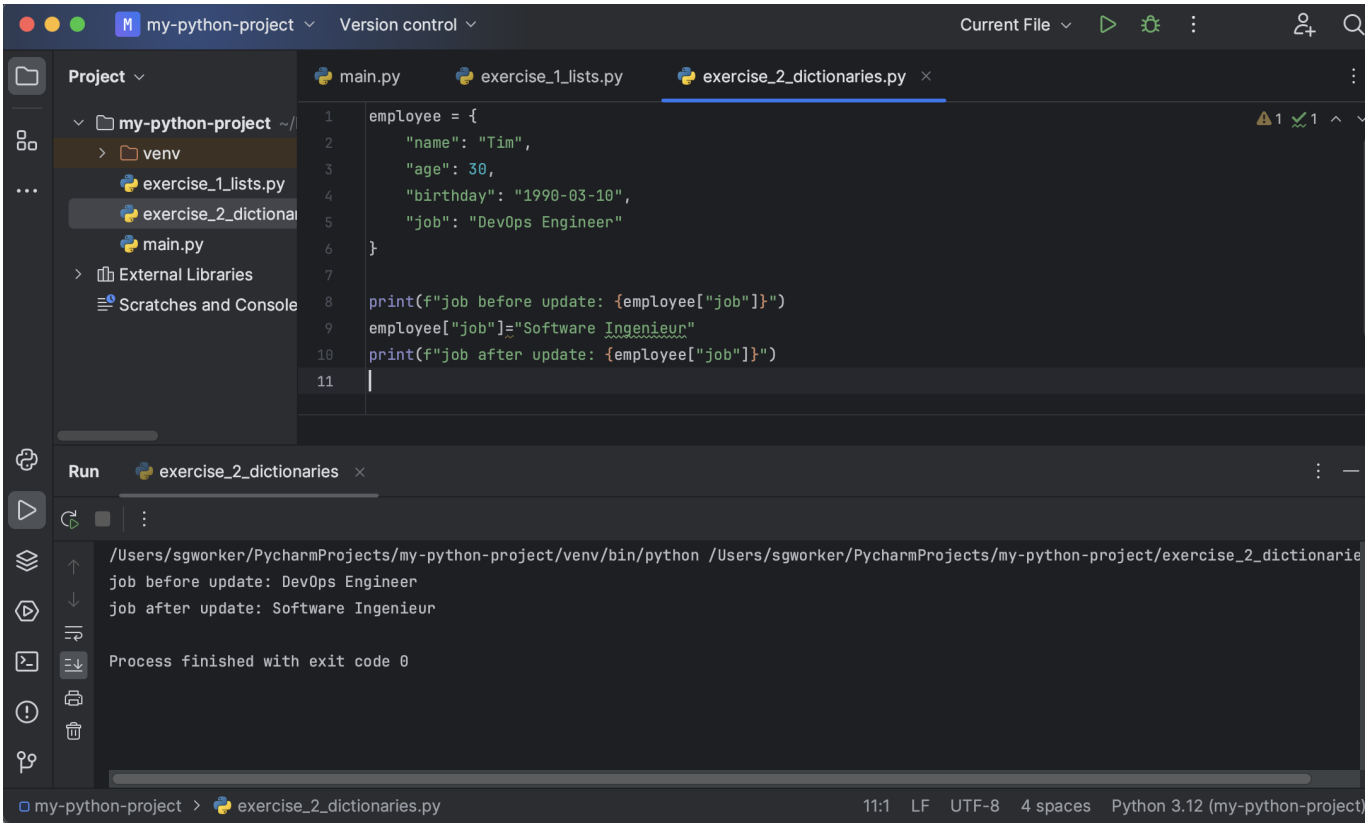
```
dict_one = {'a': 100, 'b': 400}
dict_two = {'x': 300, 'y': 200}
```

Write a Python Script that:

- Merges these two Python dictionaries into 1 new dictionary.
- Sums up all the values in the new dictionary and print it out
- Prints the max and minimum values of the dictionary

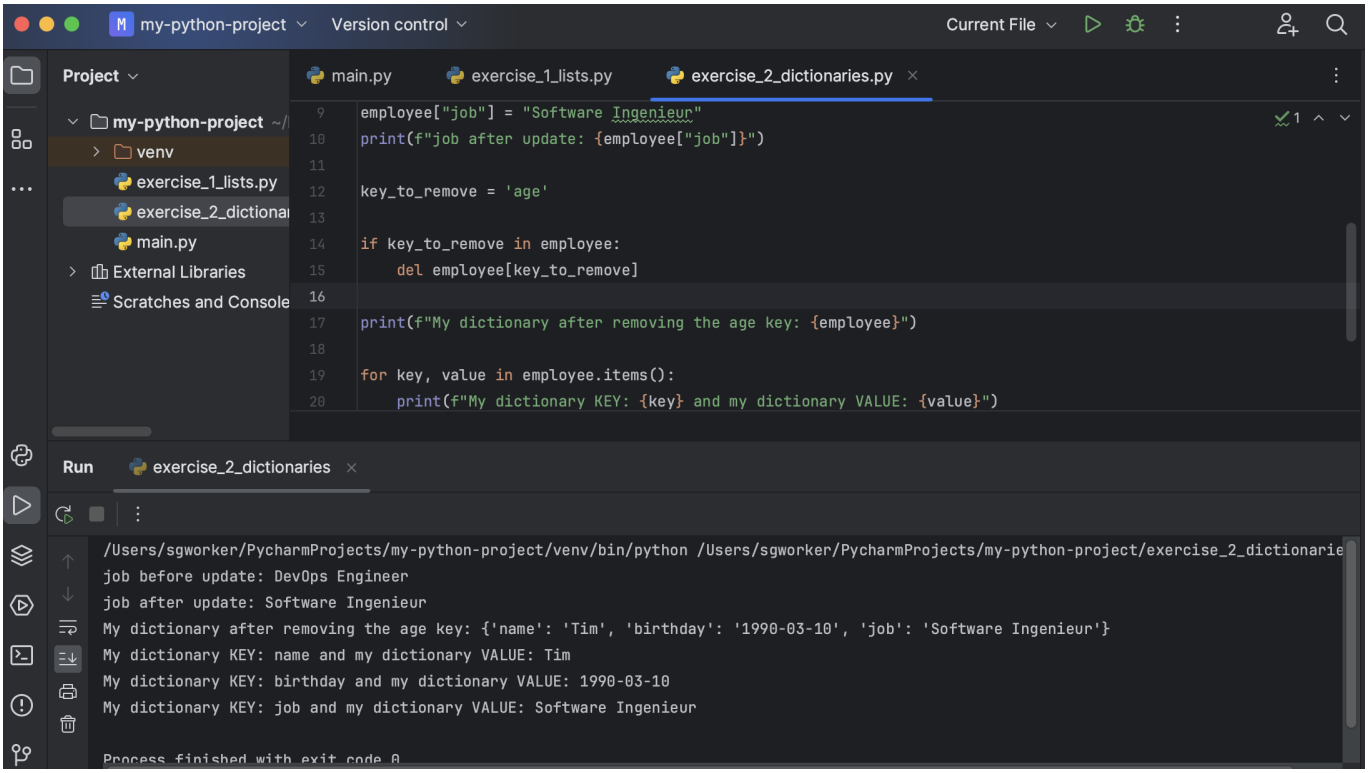
exercise_2_dictionaries.py

```
employee = {
    "name": "Tim",
    "age": 30,
    "birthday": "1990-03-10",
    "job": "DevOps Engineer"
}

print(f"job before update: {employee["job"]}")
employee["job"]="Software Ingenieur"
print(f"job after update: {employee["job"]}")
```



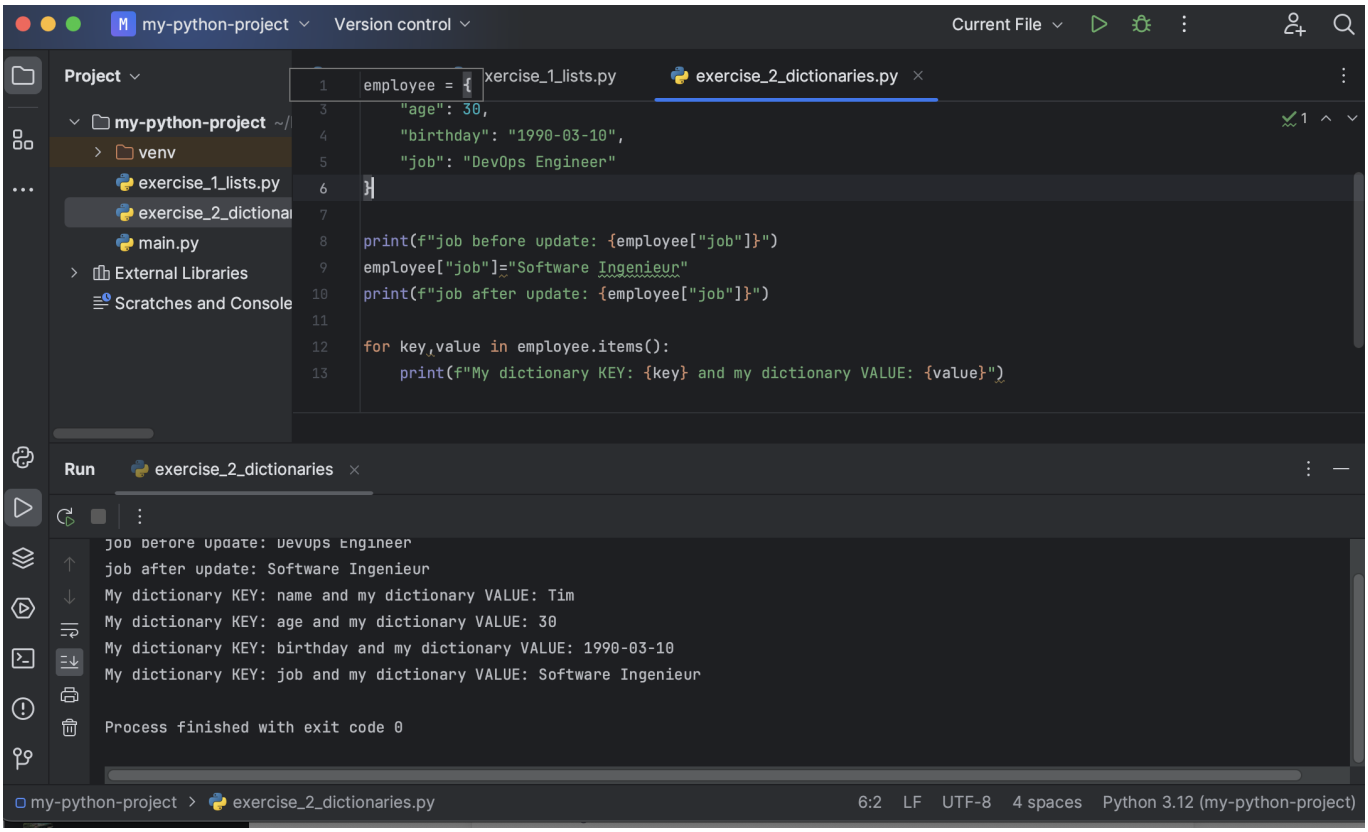

```
employee = {
    "name": "Tim",
```

```python
    "age": 30,
    "birthday": "1990-03-10",
    "job": "DevOps Engineer"
}

print(f"job before update: {employee["job"]}")
employee["job"]="Software Ingenieur"
print(f"job after update: {employee["job"]}")

key_to_remove = 'age'

if key_to_remove in employee:
    del employee[key_to_remove]

print(f"My dictionary after removing the age key: {employee}")
```

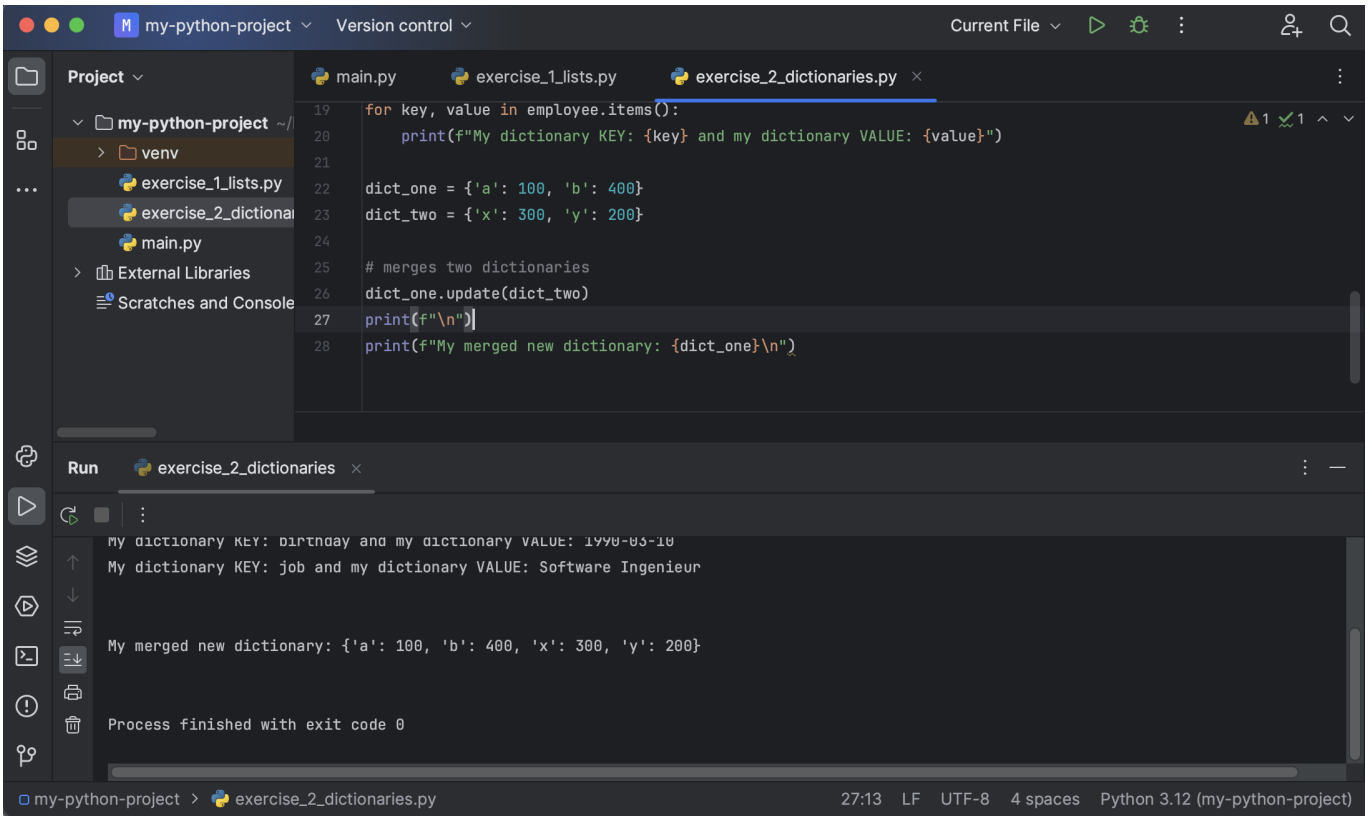



```python
employee = {
    "name": "Tim",
    "age": 30,
    "birthday": "1990-03-10",
    "job": "DevOps Engineer"
}

print(f"job before update: {employee["job"]}")
employee["job"] = "Software Ingenieur"
print(f"job after update: {employee["job"]}")

key_to_remove = 'age'

if key_to_remove in employee:
```

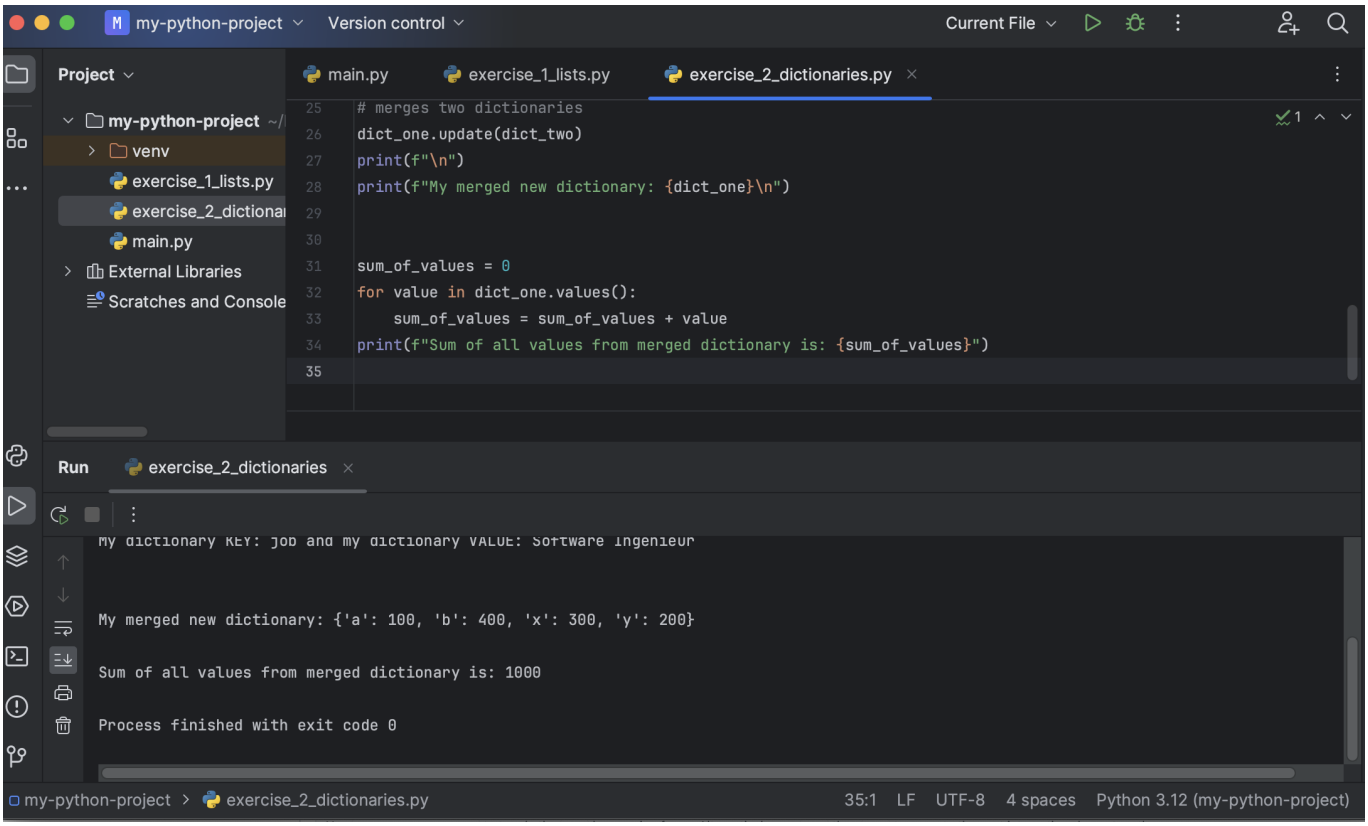
```python
    del employee[key_to_remove]

print(f"My dictionary after removing the age key: {employee}")

for key, value in employee.items():
    print(f"My dictionary KEY: {key} and my dictionary VALUE: {value}")
```



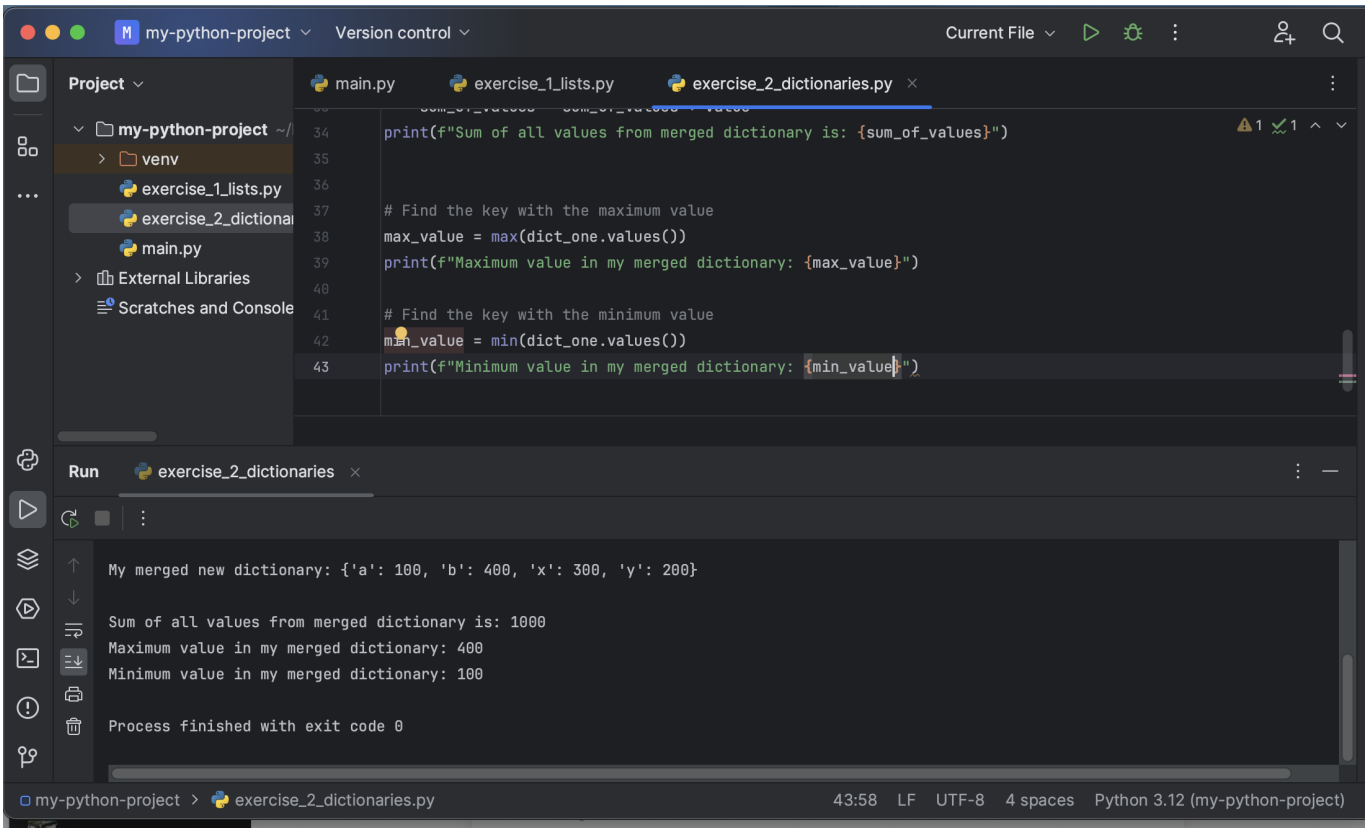

```python
dict_one = {'a': 100, 'b': 400}
dict_two = {'x': 300, 'y': 200}

# merges two dictionaries
dict_one.update(dict_two)
print(f"\n")
print(f"My merged new dictionary: {dict_one}\n")
```




```python
dict_one = {'a': 100, 'b': 400}
dict_two = {'x': 300, 'y': 200}

# merges two dictionaries
dict_one.update(dict_two)
print(f"\n")
print(f"My merged new dictionary: {dict_one}\n")


sum_of_values = 0
for value in dict_one.values():
    sum_of_values = sum_of_values + value
print(f"Sum of all values from merged dictionary is: {sum_of_values}")
```




```python
dict_one = {'a': 100, 'b': 400}
dict_two = {'x': 300, 'y': 200}

# merges two dictionaries
dict_one.update(dict_two)
print(f"\n")
print(f"My merged new dictionary: {dict_one}\n")


sum_of_values = 0
for value in dict_one.values():
    sum_of_values = sum_of_values + value
print(f"Sum of all values from merged dictionary is: {sum_of_values}")


# Find the key with the maximum value
max_value = max(dict_one.values())
print(f"Maximum value in my merged dictionary: {max_value}")

# Find the key with the minimum value
min_value = min(dict_one.values())
print(f"Minimum value in my merged dictionary: {min_value}")
```



▶ Solution 3: Working with List of Dictionaries

> EXERCISE 3: Working with List of Dictionaries Using a list of 2 dictionaries:

```
employees = [{
  "name": "Tina",
  "age": 30,
  "birthday": "1990-03-10",
  "job": "DevOps Engineer",
  "address": {
    "city": "New York",
    "country": "USA"
  }
},
{
  "name": "Tim",
  "age": 35,
  "birthday": "1985-02-21",
  "job": "Developer",
  "address": {
    "city": "Sydney",
    "country": "Australia"
  }
}]
```

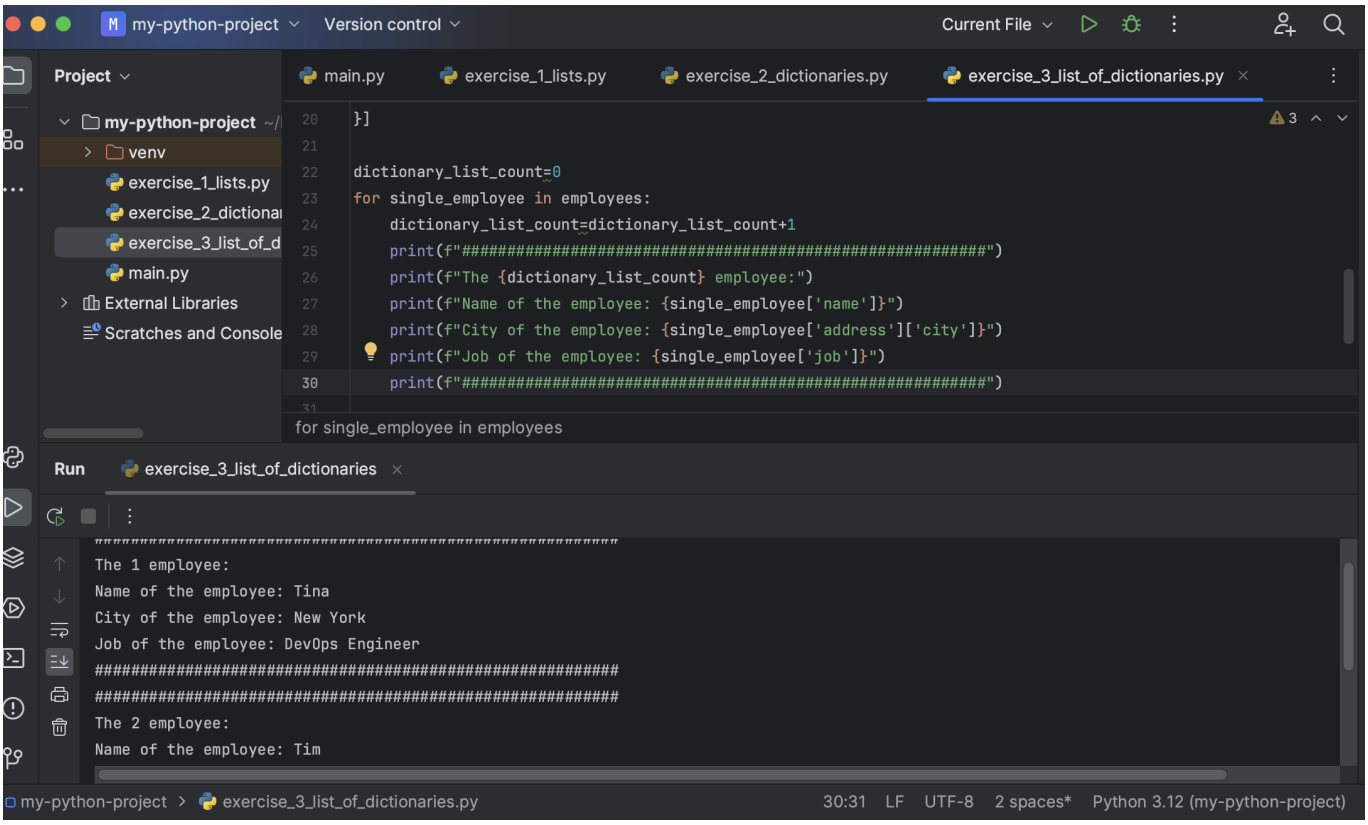
Write a Python Program that:

- Prints out - the name, job and city of each employee using a loop. The program must work for any number of employees in the list, not just 2.

- Prints the country of the second employee in the list by accessing it directly without the loop.

exercise_3_list_of_dictionaries.py

```
employees = [{
  "name": "Tina",
  "age": 30,
  "birthday": "1990-03-10",
  "job": "DevOps Engineer",
  "address": {
    "city": "New York",
    "country": "USA"
  }
},
{
  "name": "Tim",
  "age": 35,
  "birthday": "1985-02-21",
  "job": "Developer",
  "address": {
    "city": "Sydney",
    "country": "Australia"
  }
}]

dictionary_list_count=0
for single_employee in employees:
    dictionary_list_count=dictionary_list_count+1
    print(f"##########################################################")
    print(f"The {dictionary_list_count} employee:")
    print(f"Name of the employee: {single_employee['name']}")
    print(f"City of the employee: {single_employee['address']['city']}")
    print(f"Job of the employee: {single_employee['job']}")
    print(f"##########################################################")


country = employees[1]["address"]["country"]
print(f"country of the second employee: {country}")
```



▶ Solution 4: Working with Functions

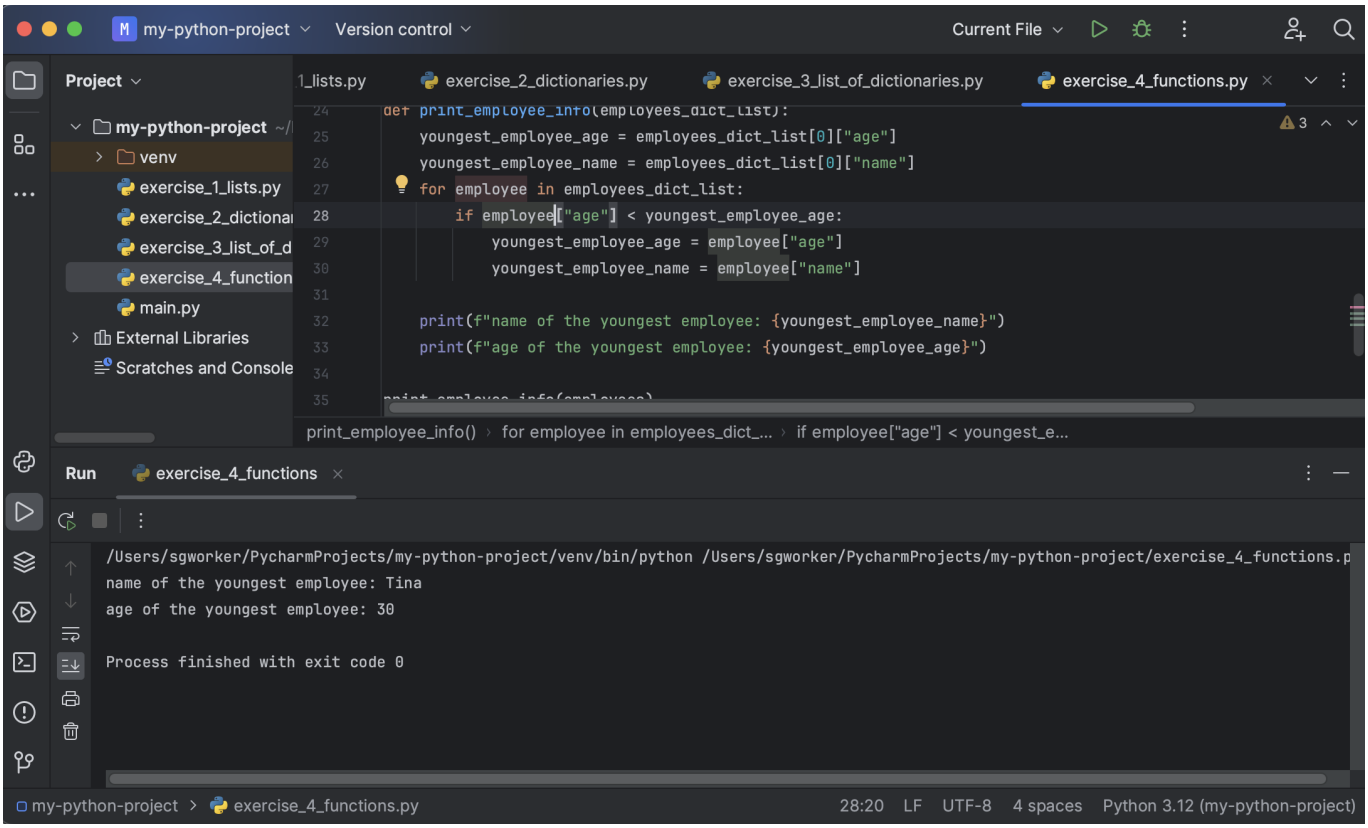
> EXERCISE 4: Working with Functions Write a function that accepts a list of dictionaries with employee age (see example list from the Exercise 3) and prints out the name and age of the youngest employee.

- Write a function that accepts a string and calculates the number of upper case letters and lower case letters.
- Write a function that prints the even numbers from a provided list.
- For cleaner code, declare these functions in its own helper Module and use them in the main.py file

exercise_4_functions.py

```
employees = [{
  "name": "Tina",
  "age": 30,
  "birthday": "1990-03-10",
  "job": "DevOps Engineer",
  "address": {
    "city": "New York",
    "country": "USA"
  }
},
{
  "name": "Tim",
  "age": 35,
  "birthday": "1985-02-21",
  "job": "Developer",
  "address": {
    "city": "Sydney",
```

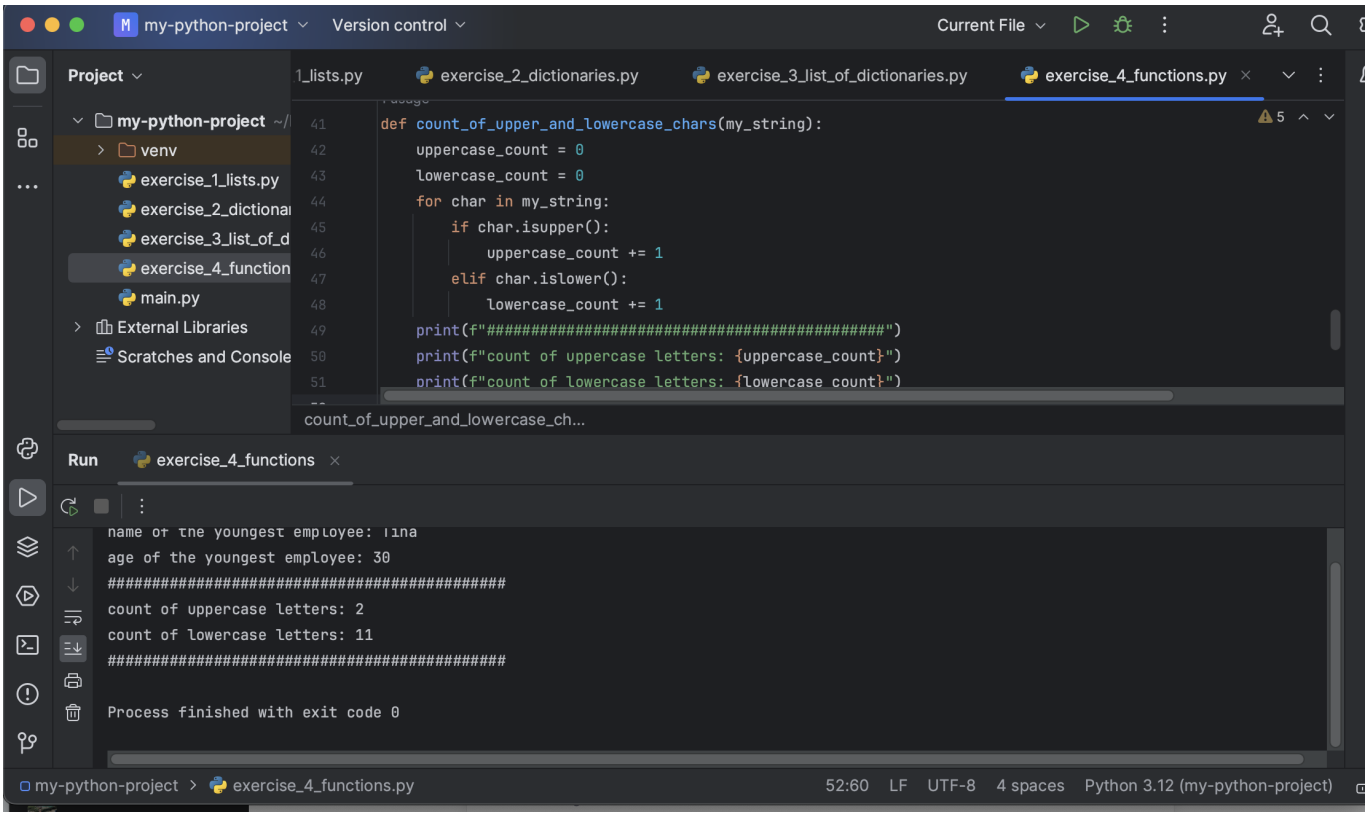
```
    "country": "Australia"
  }
}]


def print_employee_info(employees_dict_list):
    youngest_employee_age = employees_dict_list[0]["age"]
    youngest_employee_name = employees_dict_list[0]["name"]
    for employee in employees_dict_list:
        if employee["age"] < youngest_employee_age:
            youngest_employee_age = employee["age"]
            youngest_employee_name = employee["name"]

    print(f"name of the youngest employee: {youngest_employee_name}")
    print(f"age of the youngest employee: {youngest_employee_age}")

print_employee_info(employees)
```



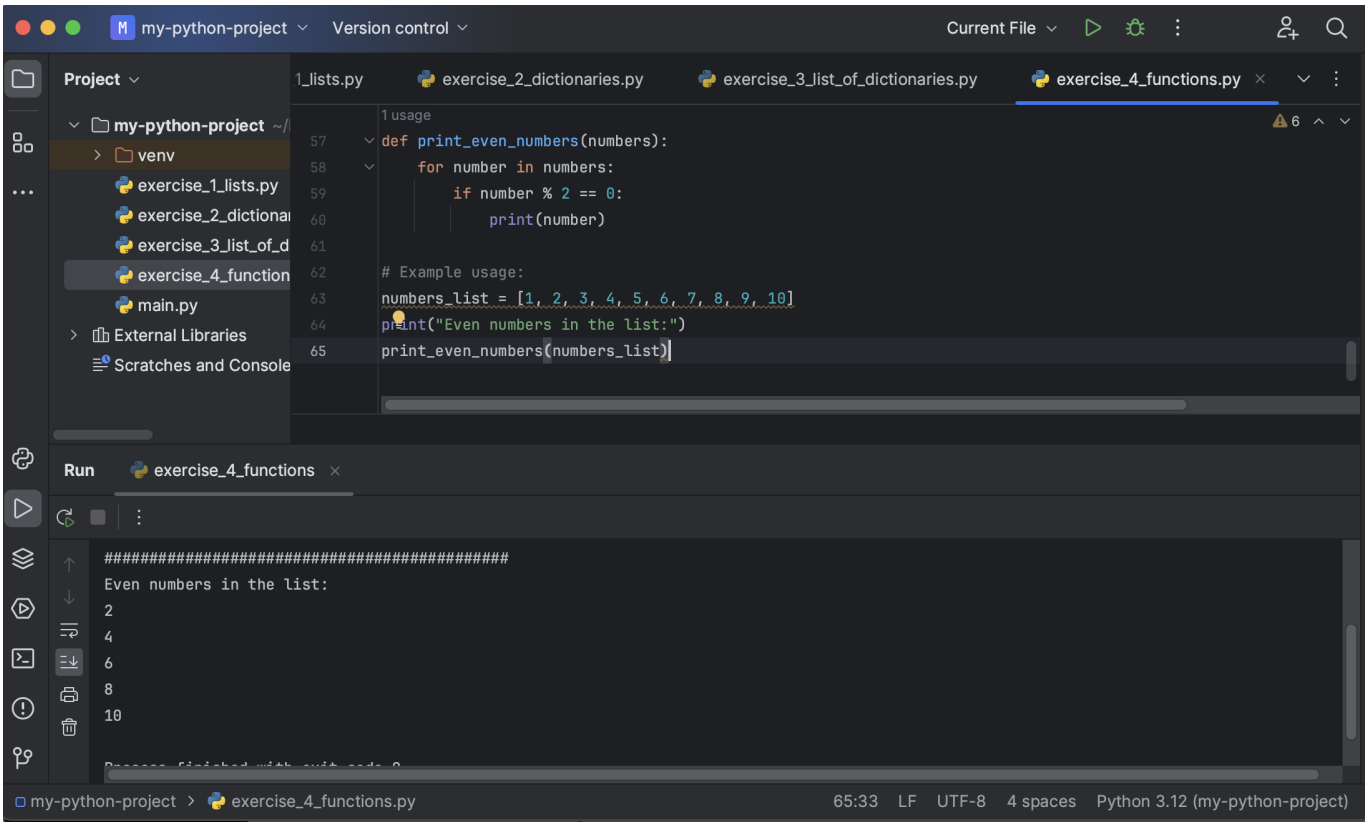
my_text = "Hello Bootcamp"

```
def count_of_upper_and_lowercase_chars(my_string):
    uppercase_count = 0
    lowercase_count = 0
    for char in my_string:
        if char.isupper():
            uppercase_count += 1
        elif char.islower():
```

```
            lowercase_count += 1
    print(f"##############################################")
    print(f"count of uppercase letters: {uppercase_count}")
    print(f"count of lowercase letters: {lowercase_count}")
    print(f"##############################################")

count_of_upper_and_lowercase_chars(my_text)
```




```python
def print_even_numbers(numbers):
    for number in numbers:
        if number % 2 == 0:
            print(number)

# Example usage:
numbers_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
print("Even numbers in the list:")
print_even_numbers(numbers_list)
```



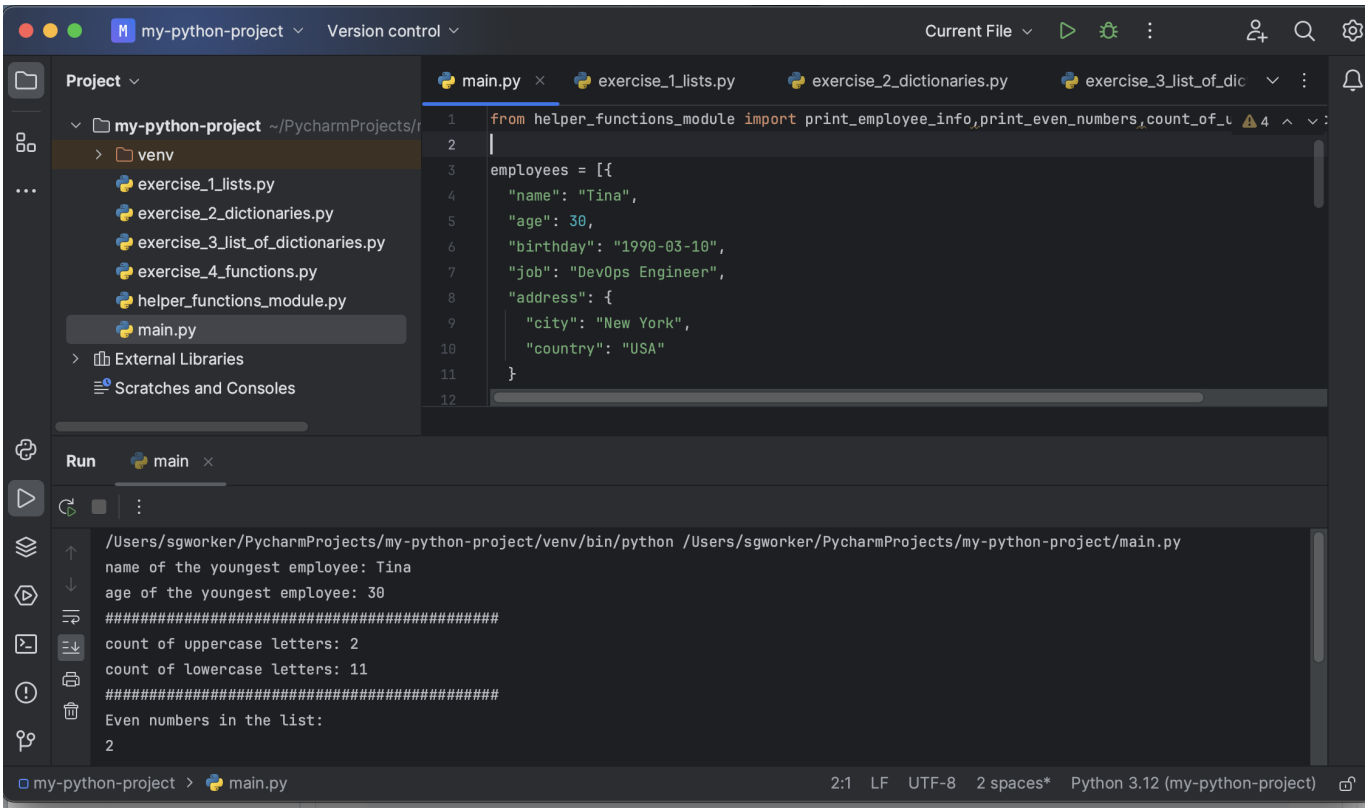
main.py

```
from helper_functions_module import
print_employee_info,print_even_numbers,count_of_upper_and_lowercase_chars

employees = [{
  "name": "Tina",
  "age": 30,
  "birthday": "1990-03-10",
  "job": "DevOps Engineer",
  "address": {
    "city": "New York",
    "country": "USA"
  }
},
{
  "name": "Tim",
  "age": 35,
  "birthday": "1985-02-21",
  "job": "Developer",
  "address": {
    "city": "Sydney",
    "country": "Australia"
  }
}]

print_employee_info(employees)

my_text = "Hello Bootcamp"
count_of_upper_and_lowercase_chars(my_text)
```

```python
# Example usage:
numbers_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
print("Even numbers in the list:")
print_even_numbers(numbers_list)
```

helper_functions_module.py

```python
def print_employee_info(employees_dict_list):
    youngest_employee_age = employees_dict_list[0]["age"]
    youngest_employee_name = employees_dict_list[0]["name"]
    for employee in employees_dict_list:
        if employee["age"] < youngest_employee_age:
            youngest_employee_age = employee["age"]
            youngest_employee_name = employee["name"]

    print(f"name of the youngest employee: {youngest_employee_name}")
    print(f"age of the youngest employee: {youngest_employee_age}")


def count_of_upper_and_lowercase_chars(my_string):
    uppercase_count = 0
    lowercase_count = 0
    for char in my_string:
        if char.isupper():
            uppercase_count += 1
        elif char.islower():
            lowercase_count += 1
    print(f"##############################################")
    print(f"count of uppercase letters: {uppercase_count}")
    print(f"count of lowercase letters: {lowercase_count}")
    print(f"##############################################")


def print_even_numbers(numbers):
    for number in numbers:
        if number % 2 == 0:
            print(number)
```



▶ Solution 5: Python Program 'Calculator'

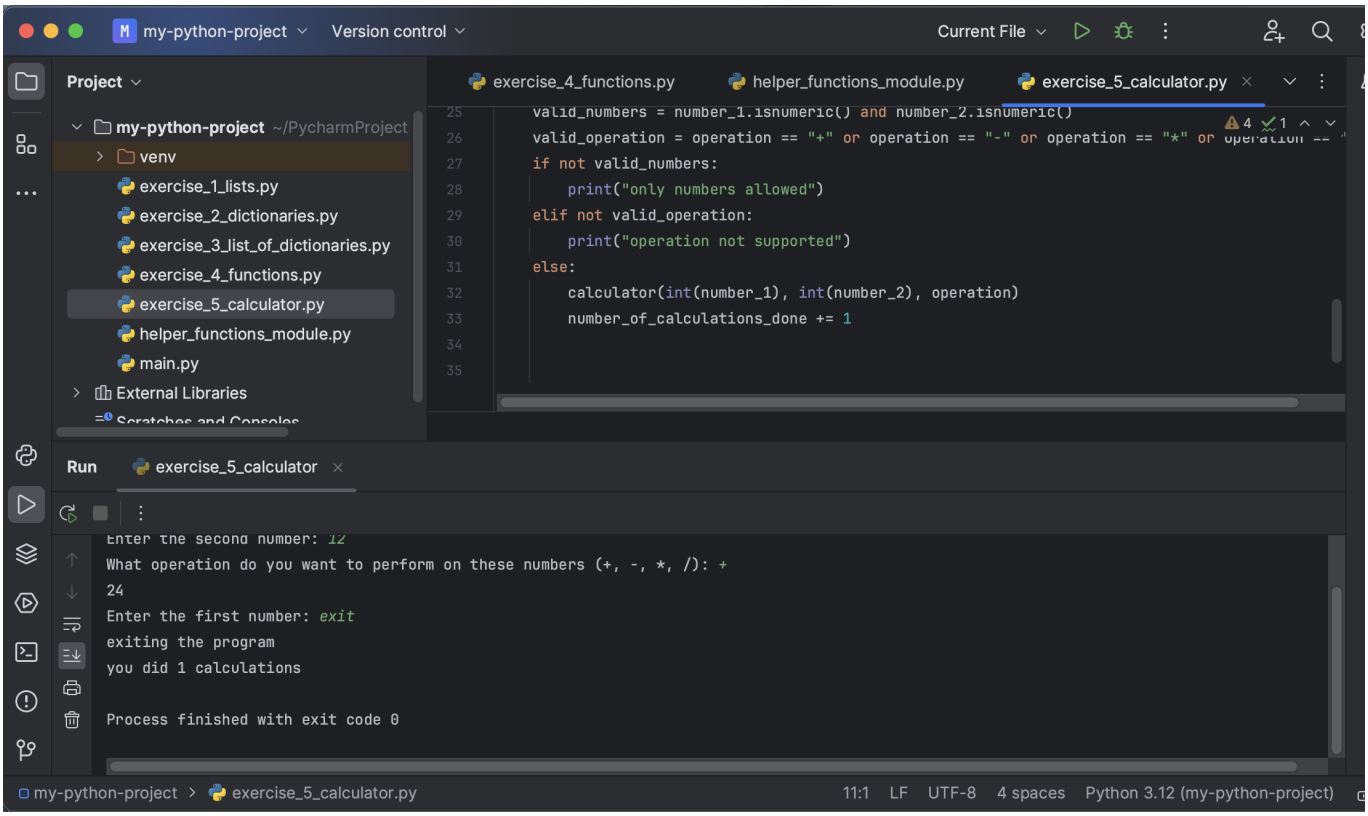
> EXERCISE 5: Python Program 'Calculator' Write a simple calculator program that:

- takes user input of 2 numbers and operation to execute handles following operations: plus, minus, multiply, divide
- does proper user validation and give feedback: only numbers allowed
- Keeps the Calculator program running until the user types “exit”
- Keeps track of how many calculations the user has taken, and when the user exits the calculator program, prints out the number of calculations the user did
- Concepts covered: working with different data types, conditionals, type conversion, user input, user input validation

exercise_5_calculator.py

```
def calculator(number_1, number_2, operation):
    # from Python verson 3.10, you can use match-case
    if operation == "+":
        print(number_1 + number_2)
    elif operation == "-":
        print(number_1 - number_2)
    elif operation == "*":
        print(number_1 * number_2)
    elif operation == "/":
        print(number_1 / number_2)


number_of_calculations_done = 0
while True:
    number_1 = input("Enter the first number: ")
```

```
    if number_1 == "exit":
        print("exiting the program")
        print(f"you did {number_of_calculations_done} calculations")
        break

    number_2 = input("Enter the second number: ")
    operation = input("What operation do you want to perform on these
numbers (+, -, *, /): ")

    valid_numbers = number_1.isnumeric() and number_2.isnumeric()
    valid_operation = operation == "+" or operation == "-" or operation ==
"*" or operation == "/"
    if not valid_numbers:
        print("only numbers allowed")
    elif not valid_operation:
        print("operation not supported")
    else:
        calculator(int(number_1), int(number_2), operation)
        number_of_calculations_done += 1
```



▶ Solution 6: Python Program 'Guessing Game'

> EXERCISE 6: Python Program 'Guessing Game' Write a program that:

- runs until the user guesses a number (hint: while loop)
- generates a random number between 1 and 9 (including 1 and 9)
- asks the user to guess the number

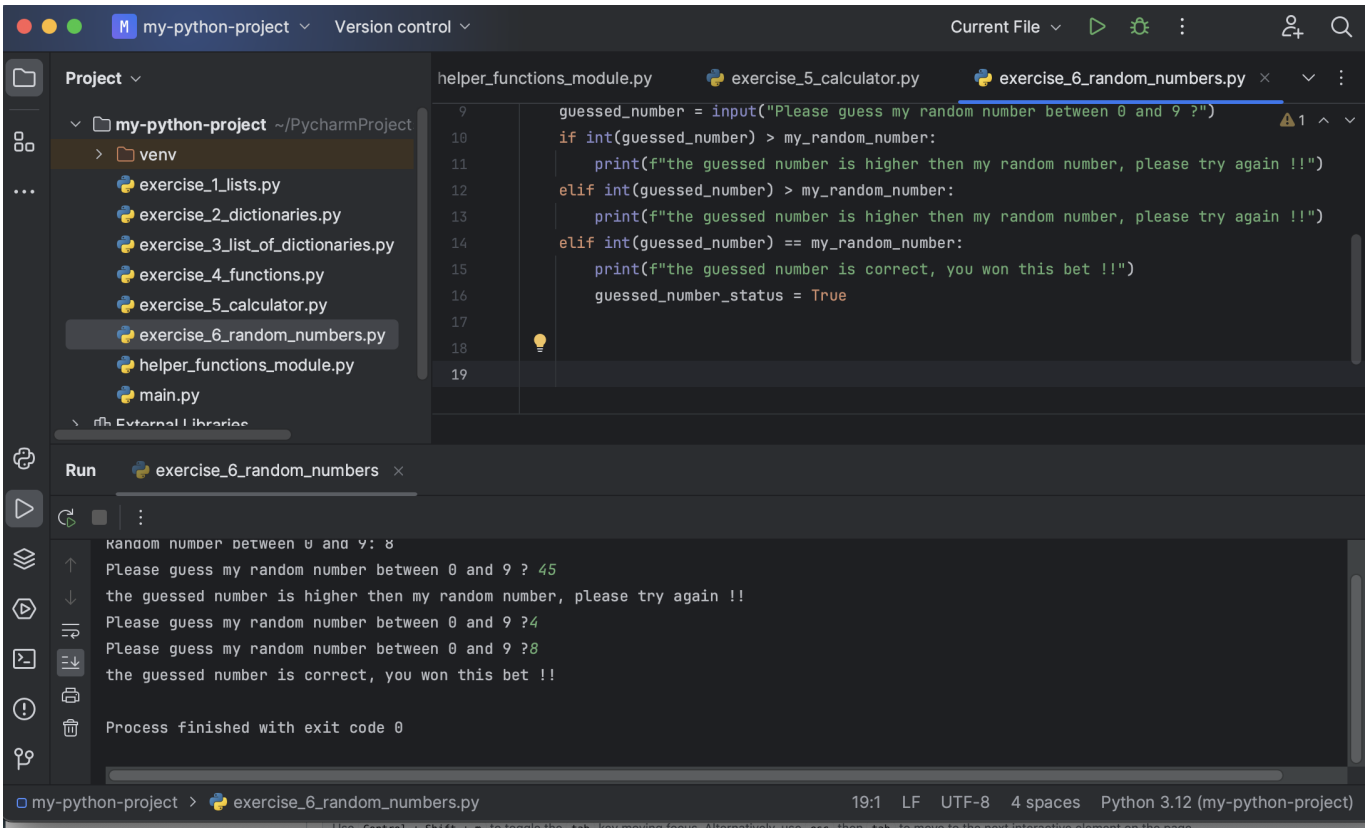
- then prints a message to the user, whether they guessed too low, too high
- if the user guesses the number right, print out YOU WON! and exit the program

> Hint: Use the built-in random module to generate random numbers
> https://docs.python.org/3.3/library/random.html

Concepts covered: Built-In Module, User Input, Comparison Operator, While loop

exercise_6_random_numbers.py

```
from random import randint

# Generate a random number between 0 and 9 (inclusive)
my_random_number = randint(1, 9)
print("Random number between 0 and 9:", my_random_number)
guessed_number_status = False

while not guessed_number_status:
    guessed_number = input("Please guess my random number between 0 and 9
?")
    if int(guessed_number) > my_random_number:
        print(f"the guessed number is higher then my random number, please
try again !!")
    elif int(guessed_number) > my_random_number:
        print(f"the guessed number is higher then my random number, please
try again !!")
    elif int(guessed_number) == my_random_number:
        print(f"the guessed number is correct, you won this bet !!")
        guessed_number_status = True
```



▶ Solution 7: Working with Classes and Objects

> EXERCISE 7: Working with Classes and Objects Imagine you are working in a university and need to write a program, which handles data of students, professors and lectures. To work with this data you create classes and objects:

- a) Create a Student class

```
with properties:

first name
last name
age
lectures he/she attends

with methods:

can print the full name
can list the lectures, which the student attends
can add new lectures to the lectures list (attend a new lecture)
can remove lectures from the lectures list (leave a lecture)
```

- b) Create a Professor class

```
with properties:

first name
last name
```

```
age
subjects he/she teaches
with methods:

can print the full name
can list the subjects they teach
can add new subjects to the list
can remove subjects from the list
```

- c) Create a Lecture class

```
with properties:

name
max number of students
duration
list of professors giving this lecture
with methods:

printing the name and duration of the lecture
adding professors to the list of professors giving this lecture
```

- d) Bonus task

As both students and professors have a first name, last name and age, you think of a cleaner solution: Inheritance allows us to define a class that inherits all the methods and properties from another class.

- Create a Person class, which is the parent class of Student and Professor classes
- This Person class has the following properties: "first_name", "last_name" and "age" and following method: "print_name", which can print the full name
- So you don't need these properties and method in the other two classes. You can easily inherit these. Change Student and Professor classes to inherit "first_name", "last_name", "age" properties and "print_name" method from the Person class

exercise_7_person_class.py

```
class Person:
    def __init__(self, first_name,last_name, age):
        self.first_name = first_name
        self.last_name = last_name
        self.age = age

    def print_person_info(self):
        print(f"Hello, my name is {self.first_name} {self.last_name} and I
am {self.age} years old.")

# Verwenden der Klasse
person1 = Person("Saban","Gül",30)
person2 = Person("Seher","Gül",35)
```

```
person1.print_person_info()
person2.print_person_info()
```

exercise_7_lecture_class.py

```
class Lecture:
    def __init__(self, name, max_students, duration, professors):
        self.name = name
        self.max_students = max_students
        self.duration_minutes = duration
        self.professors = professors

    def print_name_and_duration(self):
        print(f"{self.name} - {self.duration_minutes} minutes")

    def add_professors(self, new_professor):
        self.professors.append(new_professor)
```

exercise_7_professor_class.py

```
from exercise_7_person_class import Person


class Professor(Person):
    def __init__(self, first_name, last_name, age, lectures):
        super().__init__(first_name, last_name, age)
        self.lectures = lectures

    def list_lectures(self):
        print("Teaches lectures:")
        for lecture in self.lectures:
            print(f"- {lecture.name}")

    def teach_lecture(self, new_lecture):
        self.lectures.append(new_lecture)

    def remove_lecture(self, lecture):
        self.lectures.pop(lecture)
```

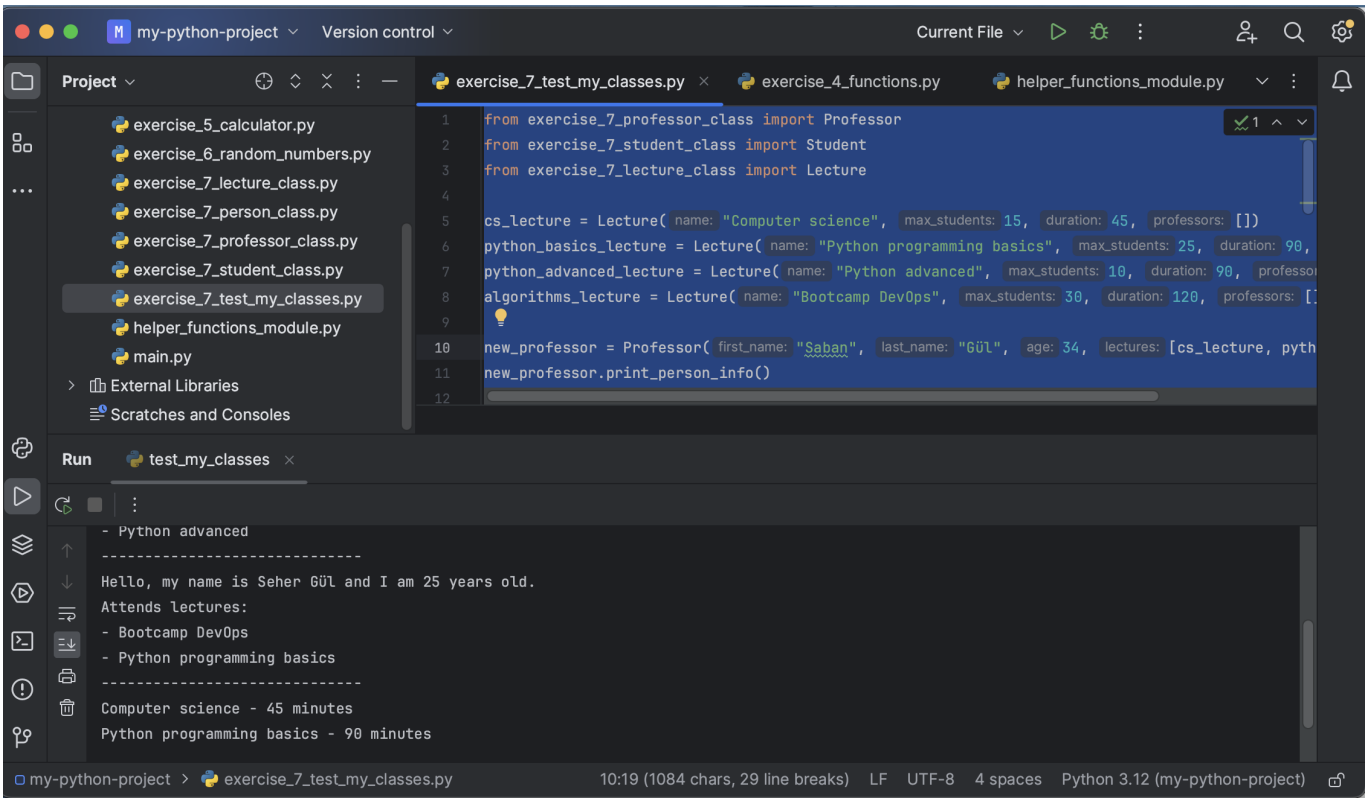
exercise_7_student_class.py

```
from exercise_7_person_class import Person


class Student(Person):
    def __init__(self, first_name, last_name, age, lectures):
        super().__init__(first_name, last_name, age)
        self.lectures = lectures

    def list_lectures(self):
        print("Attends lectures:")
        for lecture in self.lectures:
            print(f"- {lecture.name}")

    def attend_lecture(self, new_lecture):
        self.lectures.append(new_lecture)

    def leave_lecture(self, lecture):
        self.lectures.pop(lecture)
```

exercises_7_test_my_classes.py

```
from exercise_7_professor_class import Professor
from exercise_7_student_class import Student
from exercise_7_lecture_class import Lecture

cs_lecture = Lecture("Computer science", 15, 45, [])
python_basics_lecture = Lecture("Python programming basics", 25, 90, [])
python_advanced_lecture = Lecture("Python advanced", 10, 90, [])
algorithms_lecture = Lecture("Bootcamp DevOps", 30, 120, [])

new_professor = Professor("Saban", "Gül", 34, [cs_lecture,
python_basics_lecture])
new_professor.print_person_info()
new_professor.teach_lecture(python_advanced_lecture)
new_professor.list_lectures()

cs_lecture.add_professors(new_professor)
python_basics_lecture.add_professors(new_professor)
python_advanced_lecture.add_professors(new_professor)

print("------------------------------")

new_student = Student("Seher", "Gül", 25, [algorithms_lecture])
new_student.print_person_info()
new_student.attend_lecture(python_basics_lecture)
```

```
new_student.list_lectures()

print("------------------------------")

cs_lecture.print_name_and_duration()
python_basics_lecture.print_name_and_duration()
```



▶ Solution 8: Working with Dates

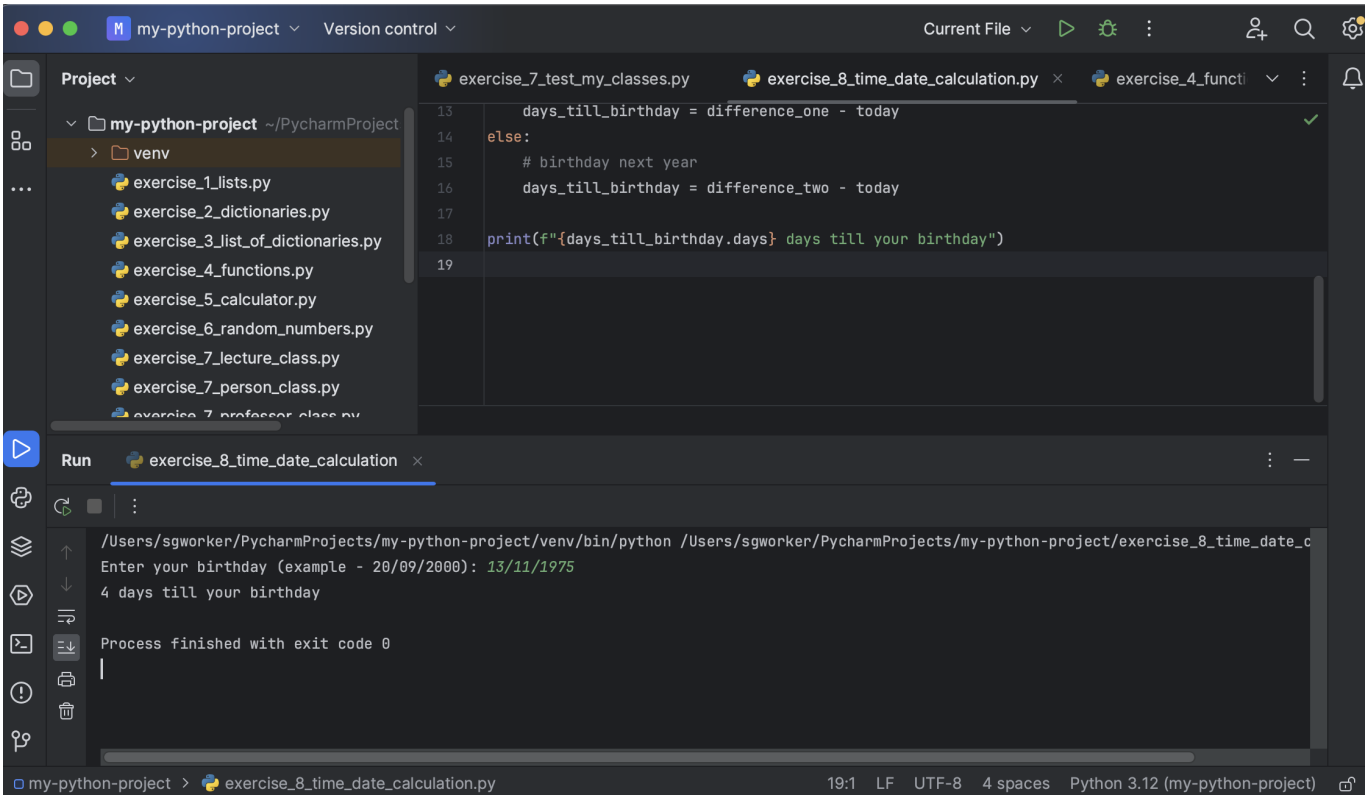
> EXERCISE 8: Working with Dates Write a program that:

- accepts user's birthday as input and calculates how many days, hours and minutes are remaining till the birthday
- prints out the result as a message to the user

exercise_8_time_date_calculation

```
from datetime import datetime
birthday_string = input("Enter your birthday (example - 20/09/2000): ")

birthday_date = datetime.strptime(birthday_string, '%d/%m/%Y').date()
today = datetime.today()

difference_one = datetime(today.year, birthday_date.month,
birthday_date.day)
difference_two = datetime(today.year + 1, birthday_date.month,
birthday_date.day)
```

```
days_till_birthday = 0
if difference_one > today:
    # birthday this year
    days_till_birthday = difference_one - today
else:
    # birthday next year
    days_till_birthday = difference_two - today

print(f"{days_till_birthday.days} days till your birthday")
```



▶ Solution 9: Working with Spreadsheets

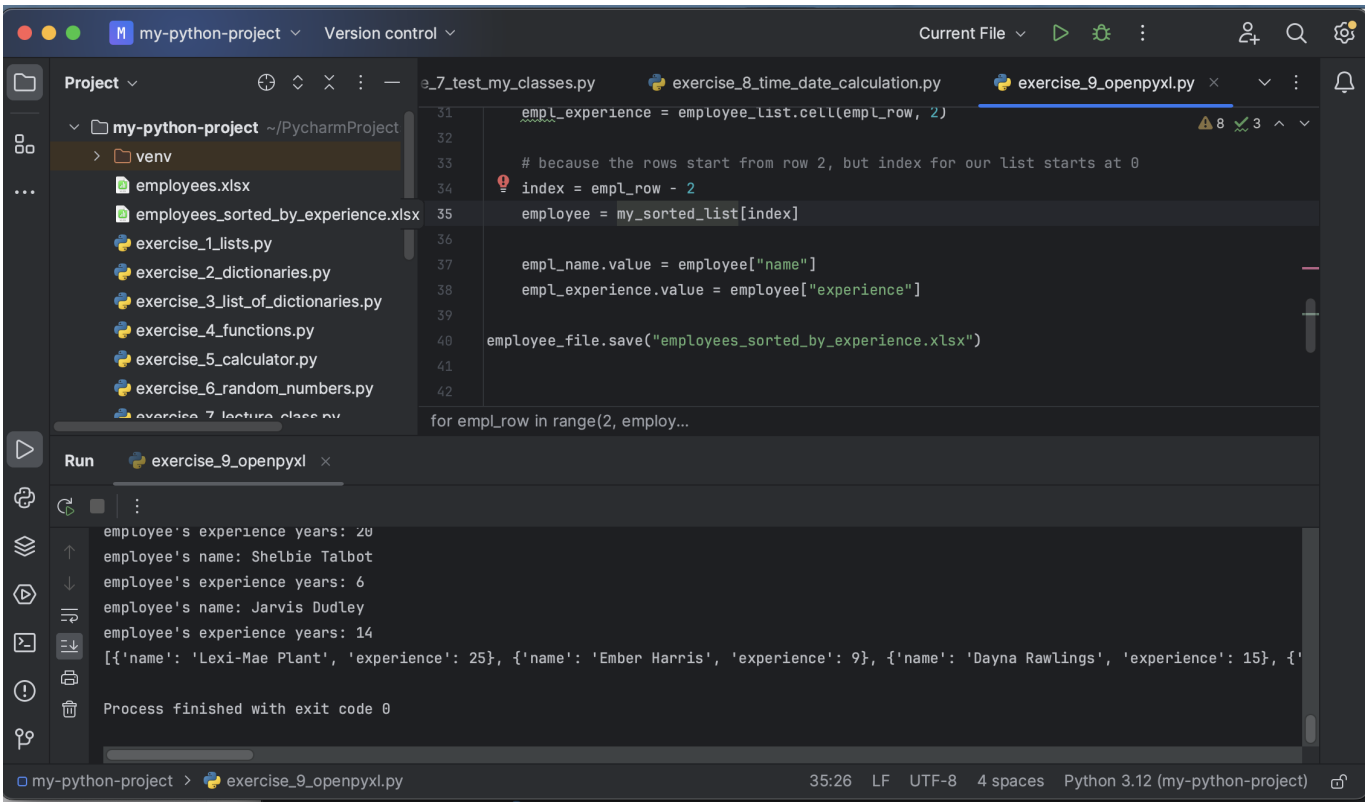
> EXERCISE 9: Working with Spreadsheets Write a program that:

- reads the provided spreadsheet file "employees.xlsx" (see Download section at the bottom) with the following information/columns: "name", "years of experience", "job title", "date of birth"
- creates a new spreadsheet file "employees_sorted.xlsx" with following info/columns: "name", "years of experience", where the years of experience is sorted in descending order: so the employee name with the most experience in years is on top.

exercise_9_openpyxl.py

```
import openpyxl
from operator import itemgetter
```

```
employee_file=openpyxl.load_workbook("employees.xlsx")
employee_list = employee_file["Sheet1"]

employee_list.delete_cols(3,4)

employees_by_experience = []

for row in range(2,employee_list.max_row+1):
    employees_name = employee_list.cell(row,1).value
    employees_years_of_experience = int(employee_list.cell(row,2).value)
    print(f"employee's name: {employees_name}")
    print(f"employee's experience years: {employees_years_of_experience}")

    employees_by_experience.append({
        "name":  employees_name,
        "experience": employees_years_of_experience
    })

print(employees_by_experience)

# sort the list of dictionaries by experience
my_sorted_list = sorted(employees_by_experience,
key=itemgetter("experience"), reverse=True)

# add entries to the spreadsheet sorted experience
for empl_row in range(2, employee_list.max_row + 1):
    empl_name = employee_list.cell(empl_row, 1)
    empl_experience = employee_list.cell(empl_row, 2)

    # because the rows start from row 2, but index for our list starts at
0
    index = empl_row - 2
    employee = my_sorted_list[index]

    empl_name.value = employee["name"]
    empl_experience.value = employee["experience"]

employee_file.save("employees_sorted_by_experience.xlsx")
```



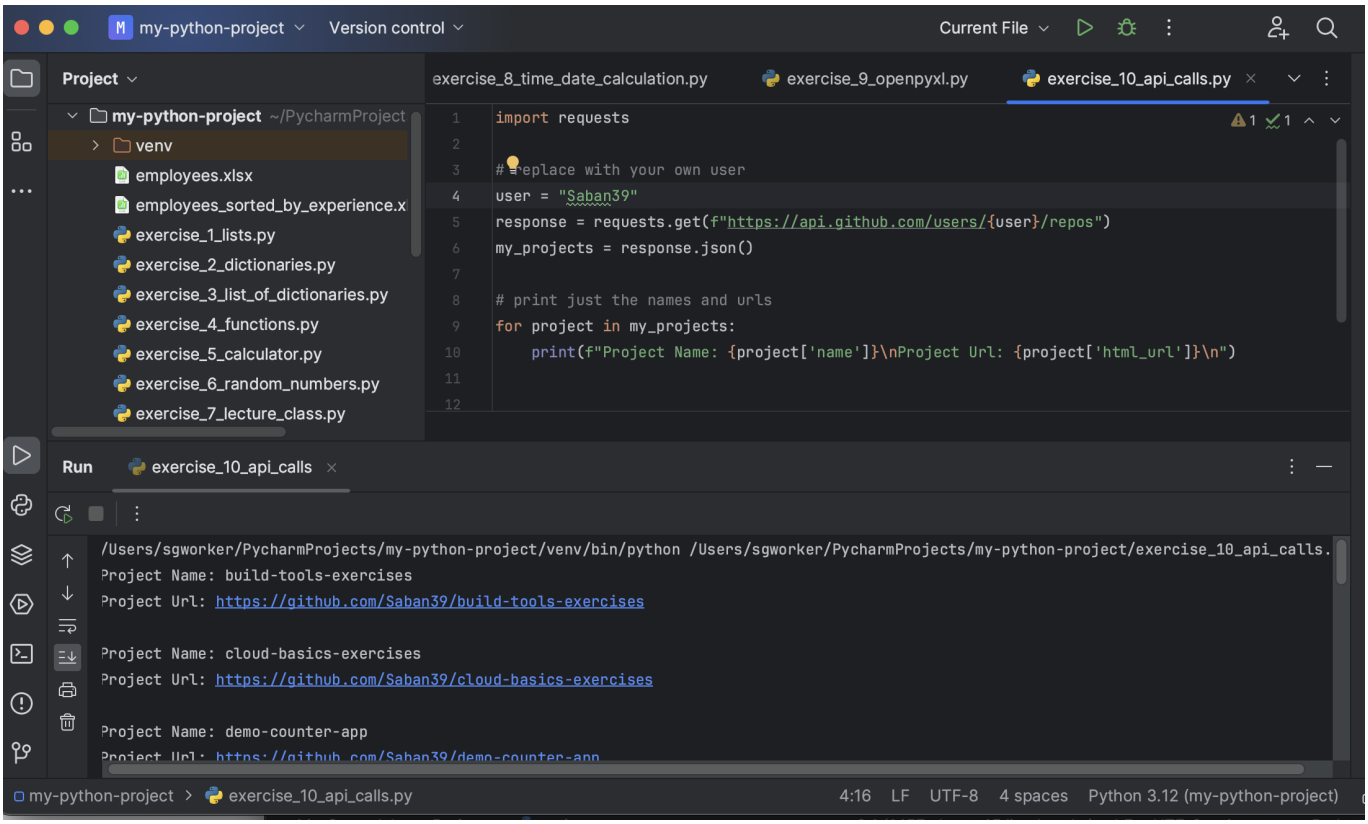
▶ Solution 10: Working with REST APIs

> EXERCISE 10: Working with REST APIs Write a program that:

- connects to GitHub API
- gets all the public repositories for a specific GitHub user
- prints the name & URL of every project

exercise_10_api_calls.py

```
import requests

# replace with your own user
user = "Saban39"
response = requests.get(f"https://api.github.com/users/{user}/repos")
my_projects = response.json()

# print just the names and urls
for project in my_projects:
    print(f"Project Name: {project['name']}\nProject Url:
{project['html_url']}\n")
```


my-python-project
Version control
Current File
Project
my-python-project ~/PycharmProject
venv
employees.xlsx
employees_sorted_by_experience.x
exercise_1_lists.py
exercise_2_dictionaries.py
exercise_3_list_of_dictionaries.py
exercise_4_functions.py
exercise_5_calculator.py
exercise_6_random_numbers.py
exercise_7_lecture_class.py
exercise_8_time_date_calculation.py
exercise_9_openpyxl.py
exercise_10_api_calls.py
import requests
# replace with your own user
user = "Saban39"
response = requests.get(f"https://api.github.com/users/{user}/repos")
my_projects = response.json()
# print just the names and urls
for project in my_projects:
print(f"Project Name: {project['name']}\nProject Url: {project['html_url']}\n")
Run
exercise_10_api_calls
/Users/sgworker/PycharmProjects/my-python-project/venv/bin/python /Users/sgworker/PycharmProjects/my-python-project/exercise_10_api_calls.
Project Name: build-tools-exercises
Project Url: https://github.com/Saban39/build-tools-exercises
Project Name: cloud-basics-exercises
Project Url: https://github.com/Saban39/cloud-basics-exercises
Project Name: demo-counter-app
my-python-project
exercise_10_api_calls.py
4:16
LF
UTF-8
4 spaces
Python 3.12 (my-python-project)