Basic Python Programming

[Session 1] Getting Started with Python

Contents

- Programming
- Intro. to Python
- Installation
- “Hello, world!”
- Basic Concepts

Programming

It matters to all of us today

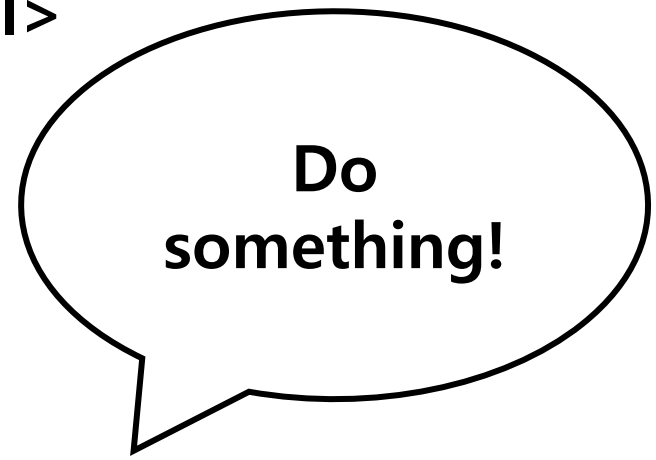
Programming [1]

- **What is programming?**

Programming [2]

- **Computer does many task for us**
  - Complicated calculation
  - Repetitive and automated task
  - So on…

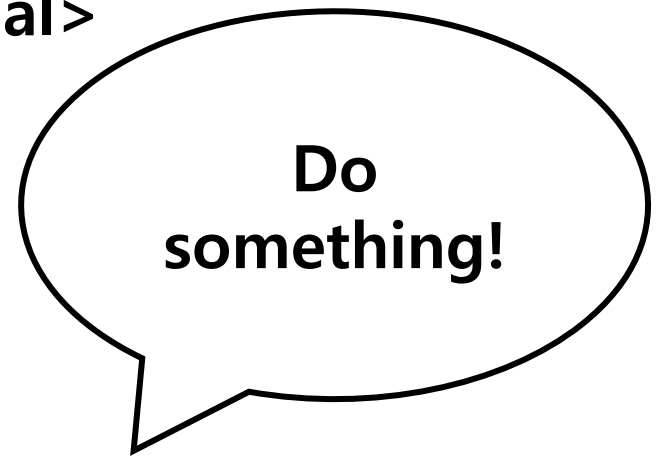
<ideal>

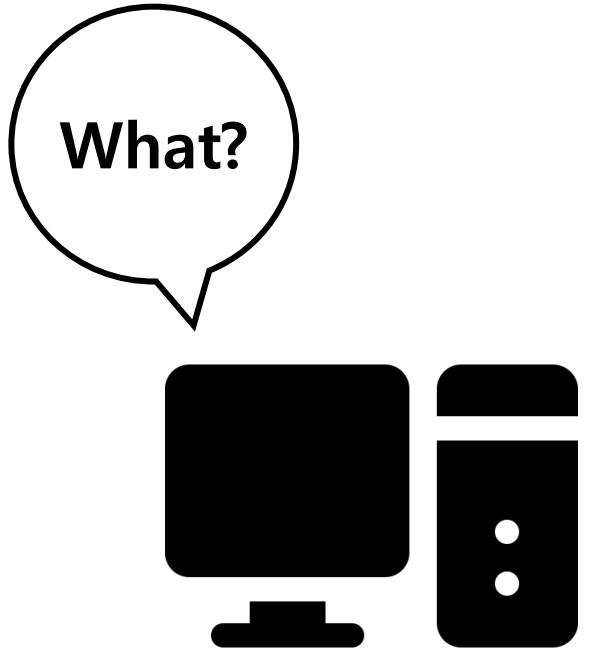


Programming [3]

- **Unfortunately, computer cannot understand what we say**

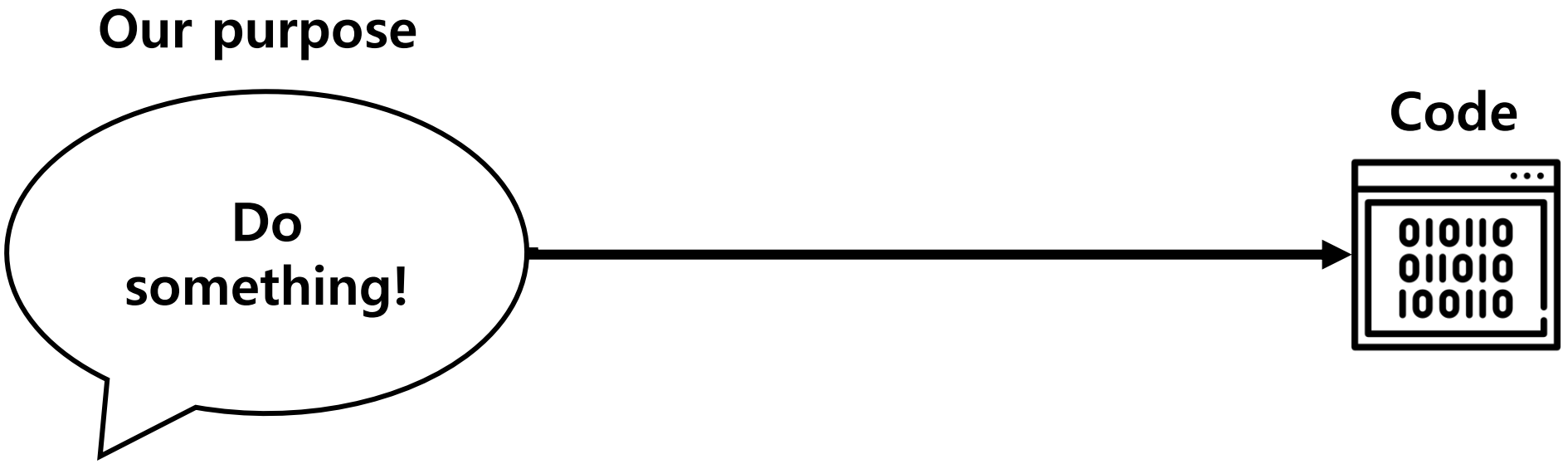
<actual>





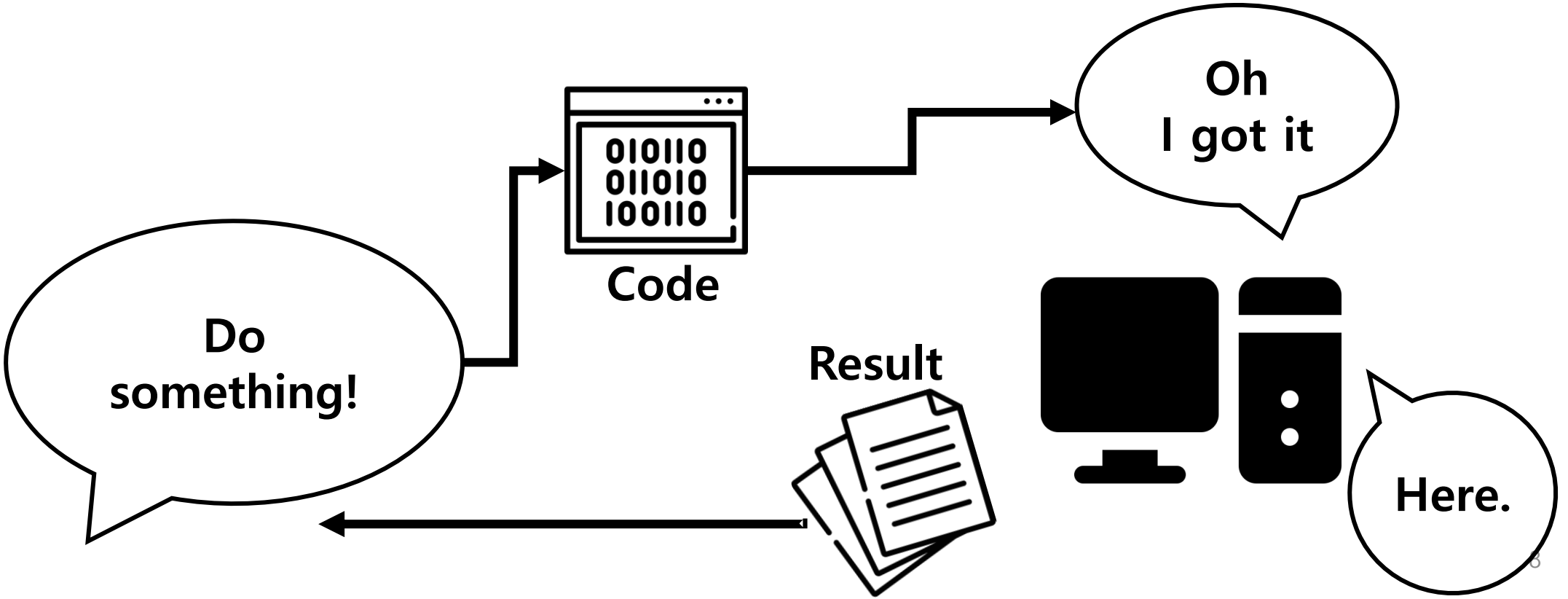
Programming [4]

- **Programming is a translation our purpose into the instruction(code).**



Programming [5]

- **Programming is the way to get computer to work according to our purpose**



Programming [6]

- **Then, why should we learn programming?**

Programming [7]

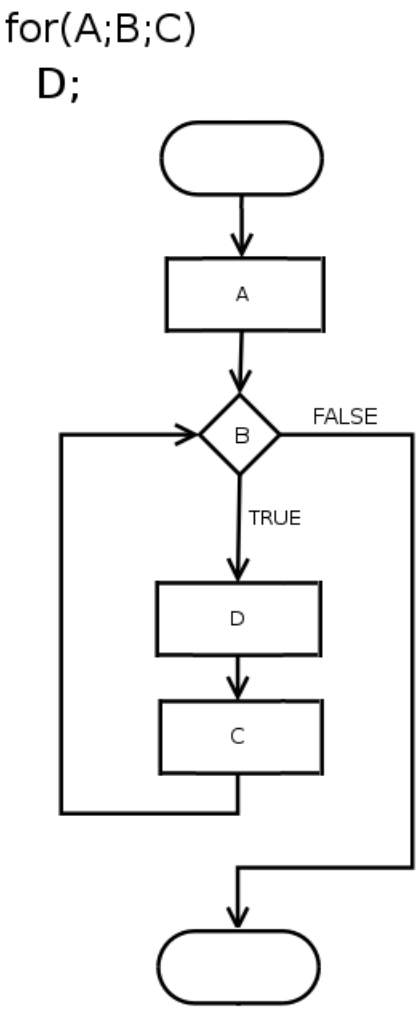
- **Programming can be a “tool”**
  - We can use it in many ways
  - It eases our life

Programming [8]

- **Programming is involved in many fields already.**
  - AI
  - Data science
  - Bioinformatics
  - Chemical / Physical simulation
  - Robotics
  - Mathematics
  - Economics
  - …

Programming [9]

- **Programming is helpful for logical thinking**
  - Algorithm
  - Logical flow
  - Prediction
  - …



Programming [10]

- **Then, let's start!**

Intro. to Python

Python?

- **Python is a programming language used in many fields.**

- **It is “very” popular programming language, why?**
  - Easy to learn
  - Easy to program
  - Many developers made useful libraries
  - There are lots of documents, guides, and forums

Why Python in This Course? [1]

- **Python has a lot of libraries, so we can make various program with Python**

Why Python in This Course? [2]

- **By using easy-to-learn language, we can focus on the BIG PICTURE of programming**
  - How to solve the given problem
  - Algorithmic / computational thinking
  - Logical flow of programs
  - So on…

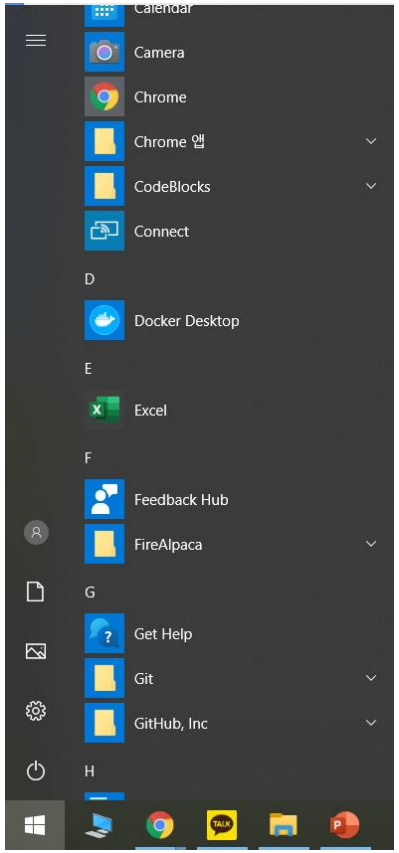
Installation

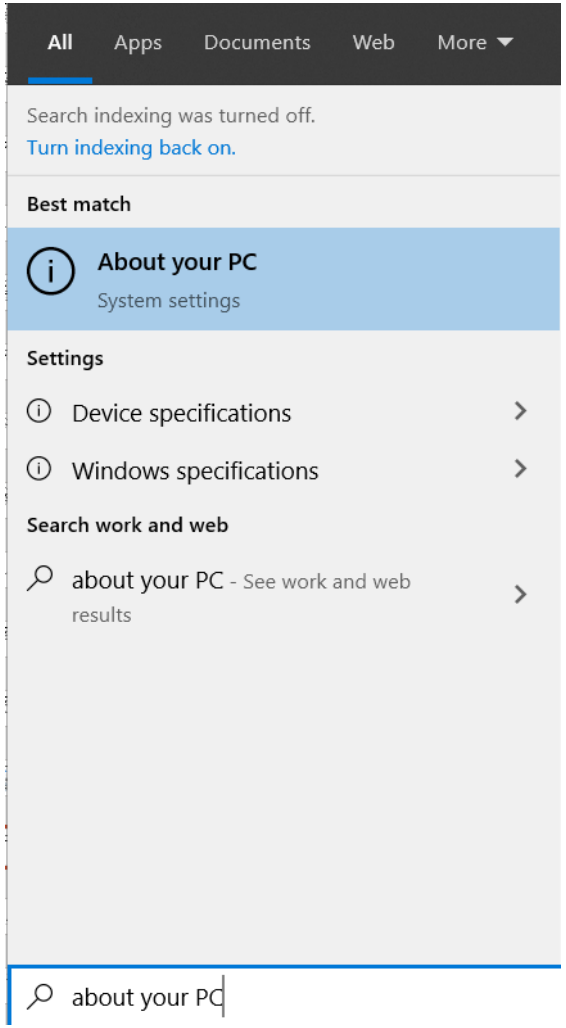
We need…

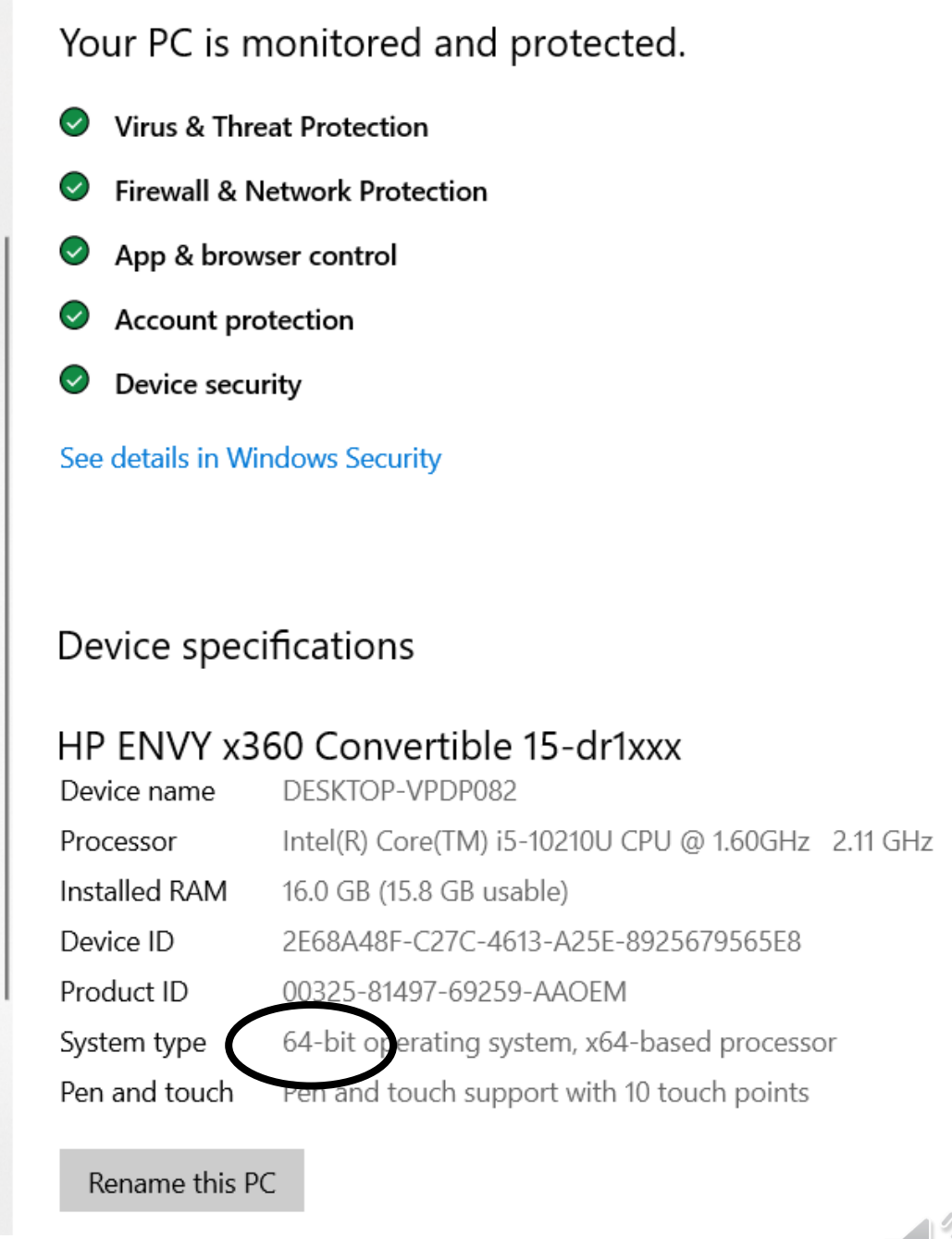
- **Python 3.7.8**
  - Interpreter for Python language
  - Be careful of the version!
- **PyCharm**
  - Editor(or IDE) for Python
- **Recommend to use Windows 10**
  - Ubuntu, MacOS, etc. are also OK.
  - But the procedure is slightly different

Installing Python [1]

- Check your PC







Installing Python [2]

- https://www.python.org/downloads/release/python-378/

For Linux

For Mac OS

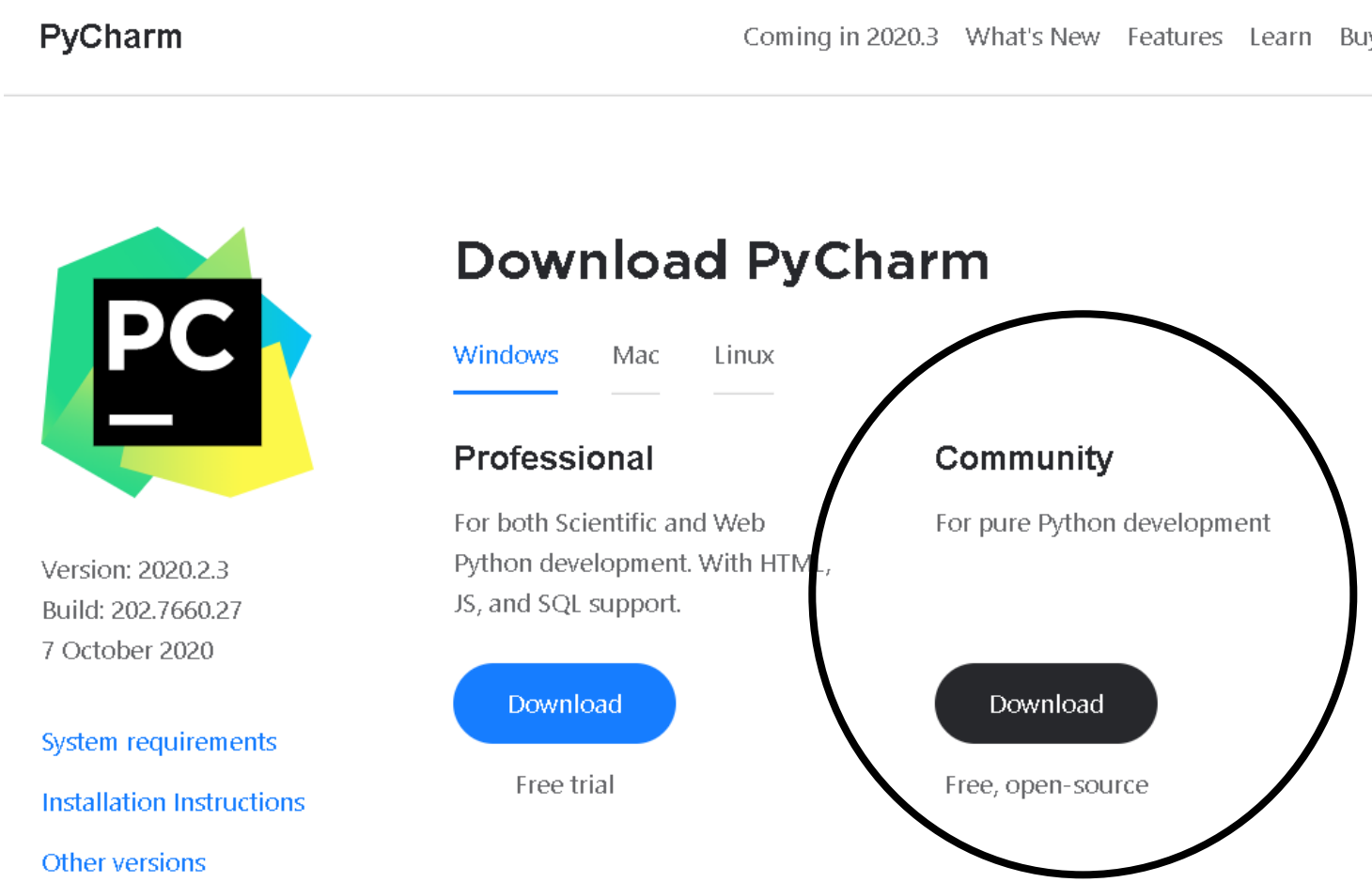
For Windows (64bit)

For Windows (32bit)

Version	Operating System	Description	MD5 Sum	File Size	GPG
Gzipped source tarball	Source release		4d5b16e8c15be38eb0f4b8f04eb68cd0	23276116	SIG
XZ compressed source tarball	Source release		a224ef2249a18824f48fba9812f4006f	17399552	SIG
macOS 64-bit installer	Mac OS X	for OS X 10.9 and later	2819435f3144fd973d3dea4ae6969f6d	29303677	SIG
Windows help file	Windows		65bb54986e5a921413e179d2211b9bfb	8186659	SIG
Windows x86-64 embeddable zip file	Windows	for AMD64/EM64T/x64	5ae191973e00ec490cf2a93126ce4d89	7536190	SIG
Windows x86-64 executable installer	Windows	for AMD64/EM64T/x64	70b08ab8e75941da7f5bf2b9be58b945	26993432	SIG
Windows x86-64 web-based installer	Windows	for AMD64/EM64T/x64	b07dbb998a4a0372f6923185ebb6bf3e	1363056	SIG
Windows x86 embeddable zip file	Windows		5f0f83433bd57fa55182cb8ea42d43d6	6765162	SIG
Windows x86 executable installer	Windows		4a9244c57f61e3ad2803e900a2f75d77	25974352	SIG
Windows x86 web-based installer	Windows		642e566f4817f118abc38578f3cc4e69	1324944	SIG

Installing PyCharm

- https://www.jetbrains.com/pycharm/download/

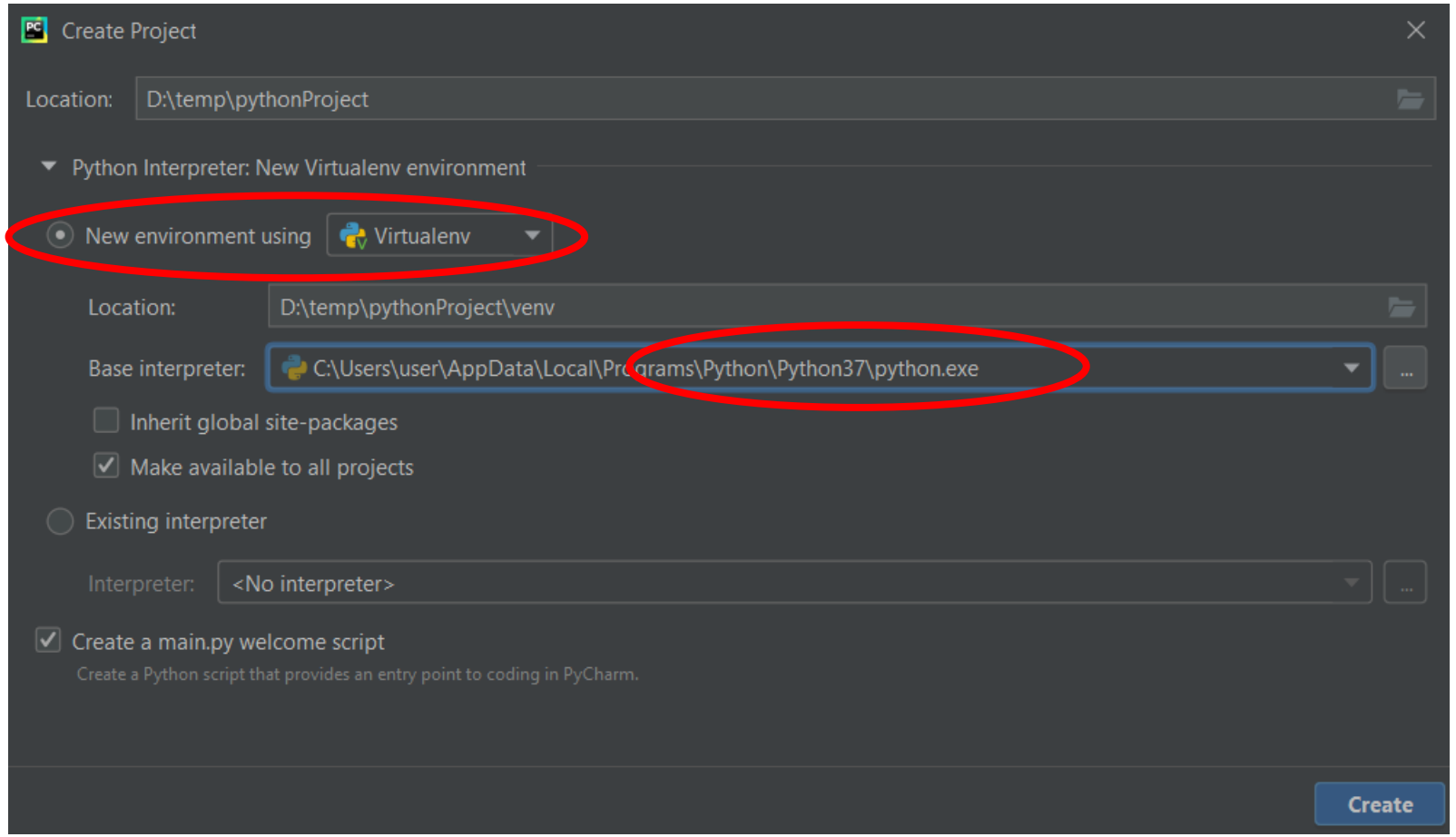


Hello, World!

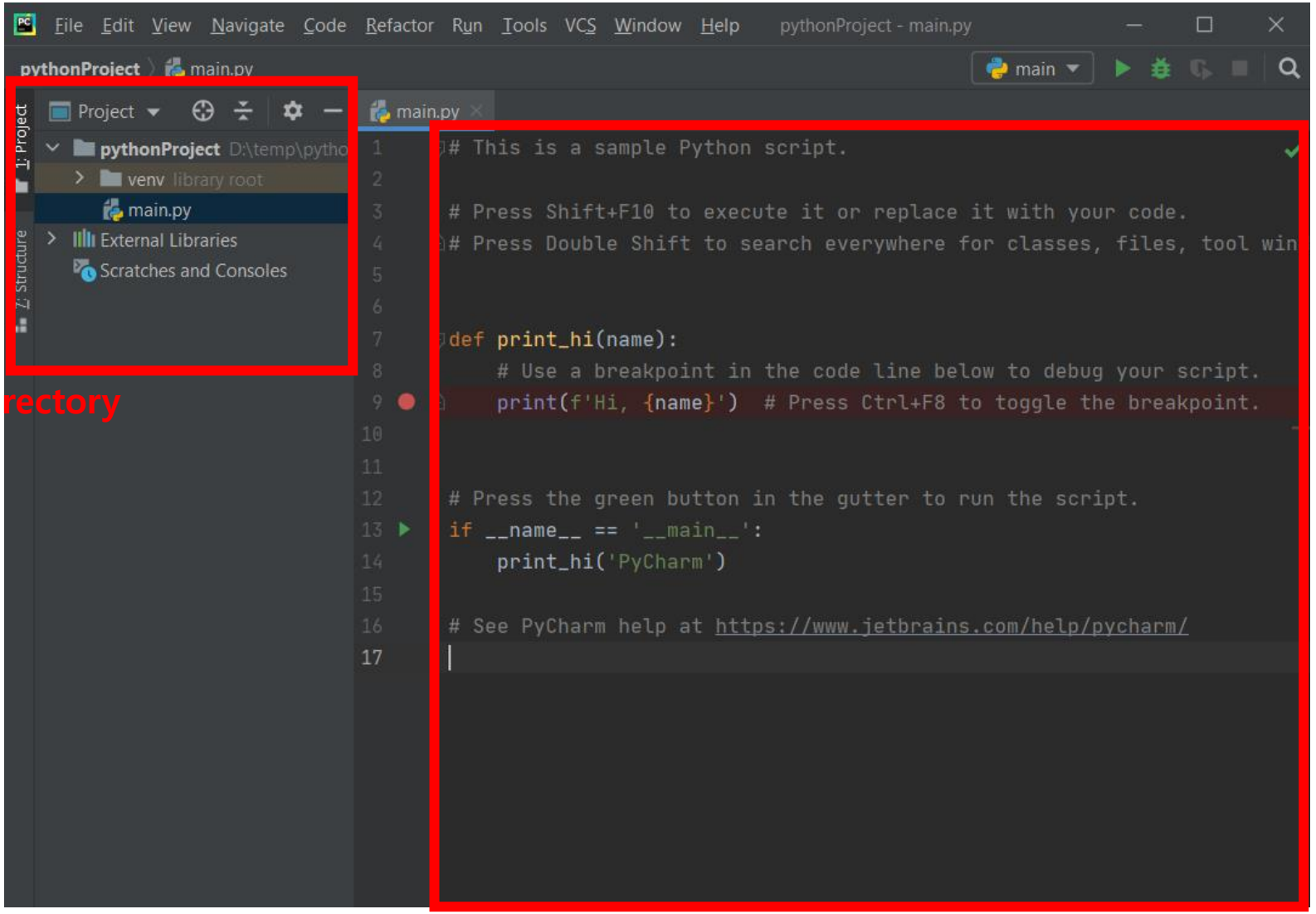
The beginning of everything

Looking around PyCharm [1]

- **Create a new project**



Looking around PyCharm [2]



Project directory

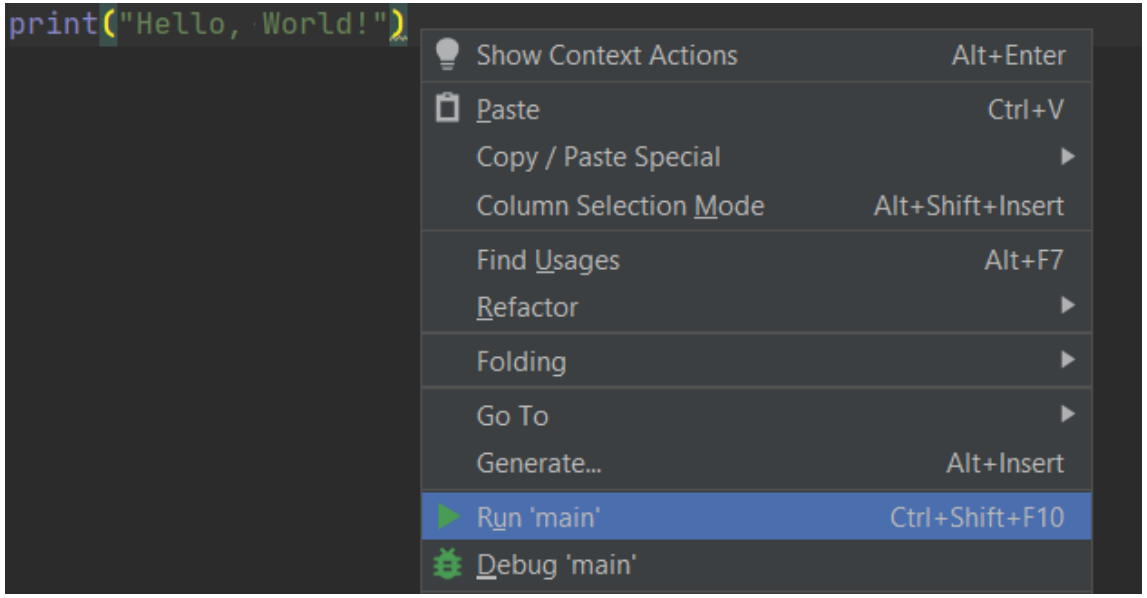
Code

Printing “Hello, World!” [1]

- **Erase all and write this:**

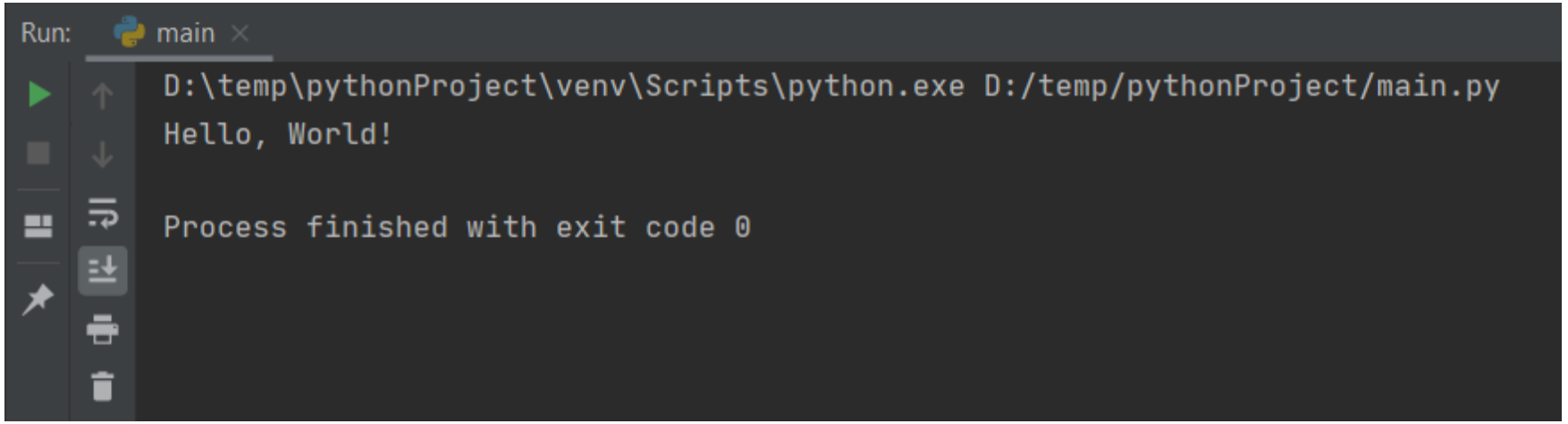
```
print("Hello, World!")
```

- **Right-click and Run ‘main’**



Printing “Hello, World!” [2]

- **Was it successful?**



Basic Concepts [1]

`print()` Function [1]

- Almost everything can be printed out by `print()` function
- We should be able to use this function to see our code's result.

```
print(contents)
```


`print()` Function [2]

- **Note that:**
  - Contents can be variable, value, or something else
  - We can print multiple things, with “,”
    - `print(10, 20, 30)`

- **Practice yourself!**

Variables [1]

- Variable is a name containing some value.

- For example, `x = 150` is a variable named “x”, containing a value, 150.

- It can contain various type of value

```
x = 150
y = "hello"
z = True
```

```
x = 100

x_new = x + 100
```

```
w = 100 + 255
```


Variables [2]

- **How can we use variable?**

```
1    x = 100
2    print(x)
3    print(x)
4
5    x = 150
6    print(x)
```

We must declare the variable before use!!!

- **From this, we can know:**
  - Variable can be reused
  - The value in a variable can be changed

Data Types

- **Many kinds of data types are supported in Python**
  - Integer (int)
  - Float (float)
  - Boolean (bool)
  - String (str)
  - List / Tuple / Set (list, tuple, set)
  - Dictionary (dict)
  - Bytes (bytes)
  - Complex (complex)
  - …

Numeric Types

- Integer, float and complex are numeric type data

- Basic arithmetic operations are supported (if valid)

```
1  print(3 + 2)
2  print(1.5 * 4)
3  print(0 ** 10)
4  print(10 / 4)
5  print(10 // 4)
6  print(10 % 4)
```


Boolean

- **Basically, boolean type can have two types of value**
  - True: Equivalent to non-zero number
  - False: Equivalent to zero

```
bool_1 = True
bool_2 = False
```

- **The result of comparison expression is Boolean**

```
1    print(5 == 3)
2    print(15 != 4)
3    print(100 > 5)
```

```
Run:  main
D:\temp\pythonProject\venv\Scripts\p
False
True
True

Process finished with exit code 0
```


String [1]

- **We can use ‘ ’, “ ”, or ‘‘‘ ’’’ to represent string**
  - ‘‘‘ ’’’ is for multiple-line string

```
a = "Hello"
b = 'How are you?'
c = "I am fine"
d = '''
Hello, how are you?
I'm fine
'''
```


String [2]

- **String can be added and repeated with + and ***

```
1  print("abc" + "d")
2  print("AAAA" * 5)
```

```
D:\temp\pythonProject\venv\Scripts\python.exe
abcd
AAAAAAAAAAAAAAAAAAAA

Process finished with exit code 0
```


List / Tuple [1]

- **List / tuple can contain multiple items**
  - Ex) (1, 2, 3, 4, 5), (“a”, “bc”, “def”)

- **The only difference between these is:**
  - List uses [(item1), (item2), …] and **mutable**
  - Tuple uses ((item1), (item2), …) and **immutable**

```
1    list_1 = [5, 3, 2, 1]
2    tuple_1 = (1, 2, 3, 4, 5)
```


List / Tuple [2]

- **They can have any type of item**
  - Even if the item is list/tuple! (nested)

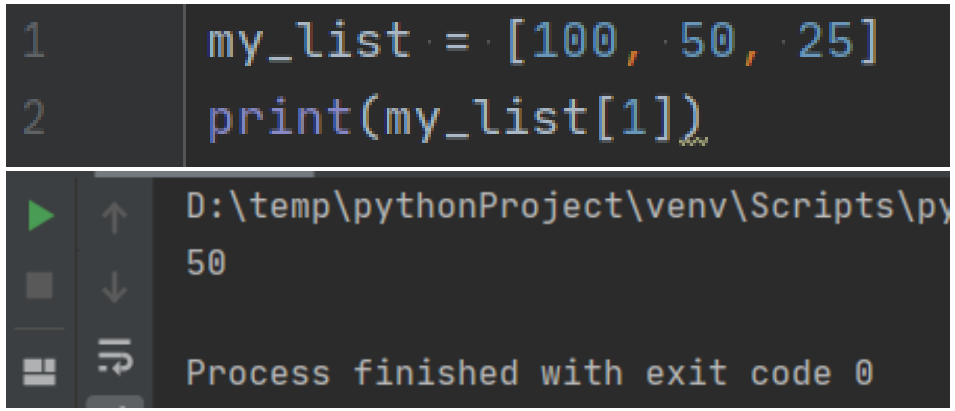
```
list_1 = [[1,2,3], [4,5,6], [7,8,9]]
```

  - A list/tuple can have different kinds of items:

```
list_2 = ["abc", 123, 5.4, True, [("cd", "ef"), []]]
```


List / Tuple [3]

- We can get i-th item from list / tuple (indexing)



- Note that the index starts at 0, not 1.

List / Tuple [4]

- We can modify the item of list (not tuple)

```
1  my_list = [100, 50, 25]
2  my_list[1] = 10000
3  print(my_list)
```

```
D:\temp\pythonProject\venv\Scripts\pyth
[100, 10000, 25]
```


List / Tuple [5]

- We can index multiple items (slicing)

```
1  my_list = [1, 2, 3, 4, 5, 6, 7, 8, 9]
2  print(my_list[2:7])
```

```
D:\temp\pythonProject\venv\Scr
[3, 4, 5, 6, 7]
```

- Note that `my_list[7]` was not included

Notes

- **Detailed explanation is in the supplement**
- **It is important to try and practice yourself!**

Some Important Functions

- **Before we learn about function, we should know some of important functions**
  - `print()`: already covered
  - `input()`: get the input from user (in console)
  - `int()`, `str()`, `list()`, …: change the type
  - `len()`: get the length of list, tuple, string, etc.
  - …

input()

- **We can get the input from user**

- **Basic use: input((message))**
  - Message can be omitted

```
1  name = input("please write your name: ")
2  print("Hello, " + name + "!")
```

```
D:\temp\pythonProject\venv\Scripts\pytho
please write your name: eunseong park
```

Write and then enter!

```
D:\temp\pythonProject\venv\Scripts\pyth
please write your name: eunseong park
Hello, eunseong park!

Process finished with exit code 0
```


int(), str(), list(), …

- **If possible, we can change the type of value / variable**
  - For example, we may want to take “121” as an integer, but it is string…

```
1    print("121" + 25)
```

  - This may cause an error

- **`int()` function can be remedy in this situation**

```
1    print(int("121") + 25)
```

```
D:\temp\pythonProject\venv\Scripts
146
```


len()

- We may want to know the “length” of list or string

```
1  my_list = [1, 2, 3, 10, 12]
2  my_string = "University of Ghana"
3  print(len(my_list))
4  print(len(my_string))
```

```
D:\temp\pythonProject\venv\Scripts\python.exe
5
19
```

- How about this?

```
1  my_list = [[1,2,3], [4,5,6]]
2  print(len(my_list))
```


Conditionals: `if`

- **We can execute different code according to the condition**

```
if (condition1):
        (code_1)
elif (codition2):
        (code_2)
elif (condition3):
        (code_3)
...
else:
        (code_else)
```

```
1    a = 3
2    if a > 3:
3        print("Greater than 3")
4    elif a == 3:
5        print("Equal to 3")
6    else:
7        print("Less than 3")
```


Loops: `while`

- **We can repeat some work, by `while` and `for` statement**

while (condition):

(code)

```
1   a = 3
2   while a < 6:
3       a += 1
4       print(a)
```

```
D:\temp\pythonProject\venv\Scripts\python.ex
4
5
6

Process finished with exit code 0
```


Loops: for [1]

- **In for statement, an "iterator" traverses given list-like object**
  - Iterator is a (usually) temporary variable
  - The list-like object is called "iterable" object

for (iterator) in (iterable):

(code)

```
sum = 0
for i in [1, 2, 3, 4, 5]:
    sum += i
print(sum)

for j in ("abc", "def"):
    print(j)
```

```
Run:  main
D:\temp\pythonProject\venv\Scripts\python.exe D:/
15
abc
def
```

Loops: for [2]

- **range() function provides a sequence of number**
  - It is useful for using for statement
  - It gives "range" type, it is also iterable (list-like)

```
range(start:end:step)
```

Starting number (can be omitted)

Ending number
The number, end is not included in the result

step (can be omitted)

```
1  sum = 0
2  for i in range(101):
3      sum += i
4  print(sum)
```

```
D:\temp\pythonProject\venv\Scripts\python
5050

Process finished with exit code 0
```


Exercises

- Some exercises for you are in “exercises”

Indentation

- It matters in Python, unlike other programming languages
- Usually, indent after some statement with “:”
  - `If (condition): / while (condition): / for i in (iterable):`
- Inappropriate indentation can cause an error

```
1   sum = 0
2
3   for i in range(101):
4   sum += i
5
6   print(sum)
```

```
D:\temp\pythonProject\venv\Scripts\python.exe D:
  File "D:/temp/pythonProject/main.py", line 4
    sum += i
      ^
IndentationError: expected an indented block

Process finished with exit code 1
```

Comment [1]

- **You can add some comment(memo) in your code**
  - In program, it does nothing

- **Why we use comment?**
  - To explain to co-worker my code
  - Also, to explain to **myself in tomorrow**

  - Sometimes we use it to disable some code temporarily

from https://monkeyuser.com

Comment [2]

- **Two ways:**
  - using # for a line
  - using ‘’’ ‘’’ for multiple lines

```
1  print("hello")  # This is for printing hello
2  '''
3  The following function
4  prints "hello" !
5  '''
6  print("hello")
7  # print("hello")
```


Basic Concepts [2]

Functions, classes, libraries

Functions in Programming

- **Function in programming is slightly different with that in math**
  - Function in math just give some value
    - For example, in f(x) = 2x, f(10) gives 20.
    - Just calculation, no side-effects
  - Function in programming is a code sequence that does some work
    - We can get some value like the function in math
      - The value is called "return value"
    - We can make some side-effects, other than just calculation
      - Print out some message
      - Change some variable
      - Cause an error

Why Do We Use Function?

- We can avoid repetitive task and code
- It makes maintenance easier
- Reusability
- So on…

```
sum_1 = 0
sum_2 = 0
sum_3 = 0
for i in range(100):
    sum_1 += i
for j in range(1000):
    sum_2 += j
for k in range(10000):
    sum_3 += k
```

Defining Function [1]

- We use def keyword

```
def function_name(parameter):
    (body)
```

```
1  def print_helloworld():
2      print("Hello, World!")
```

```
6  def sum_from_1_to(n):
7      sum = 0
8      for i in range(n+1):
9          sum += i
10     return sum
```


Defining Function [2]

- Some function may not have parameter

```
1  def print_helloworld():
2      print("Hello, World!")
```

- Some function may have two or more parameters

```
1  def print_number(a, b):
2      print(a, b)
```


Defining Function [3]

- **We can set a default argument**
  - we omit the argument, then default value is used

```
1    def print_number(n=100):
2        print(n)
```

- **Note that non-default one must precede default one!**
  - This causes an error

```
1    def print_number(a=100, b):
2        print(a, b)
```


Defining Function [4]

- **return keyword determines a return value of the function**
  - When we return, the function is terminated, immediately

```
6   def sum_from_1_to(n):
7       sum = 0
8       for i in range(n+1):
9           sum += i
10      return sum
```

- **Some function may not have a return value**

Using Function [1]

- We can call some function with `(FunctionName)(parameters)`

```
1  def adder(a, b):
2      return a + b
3
4
5  print(adder(5, 3))
```

```
D:\temp\pythonProject\venv\Scripts\py
8

Process finished with exit code 0
```


Using Function [2]

- We can indicate the parameter explicitly (if needed)

```
1    def adder(a, b):
2        return a + b
3
4
5    print(adder(a=5, b=3))
```


More about Function [1]

- A function can call another function

```
1  def add_and_square(a, b):
2      return adder(a, b) ** 2
3
4
5  def adder(a, b):
6      return a + b
7
8
9  print(add_and_square(5, 3))
```

```
D:\temp\pythonProject\venv\Scripts\pytho
64

Process finished with exit code 0
```


More about Function [2]

- **Even it can call itself! (called “recursion”)**

```
1  def factorial(n):
2      if n <= 1:
3          return 1
4      else:
5          return n * factorial(n-1)
6
7
8  print(factorial(5))
```

```
D:\temp\pythonProject\venv\Scripts\python.exe
120

Process finished with exit code 0
```


Exercises

- Some exercises for you are in “exercises”

Class: Motivation

- **How can we store / deal with each student's information?**
  - It includes name, ID, grade, GPA, etc.
  - …like this? What if there are 3~4000 students?

```
name_kevin = "kevin"
id_kevin = 20191154
grade_kevin = 2
GPA_kevin = 4.1

name_john = "john"
id_john = 20152243
grade_john = 4
GPA_john = 3.74
```

- **Is there any wiser way?**

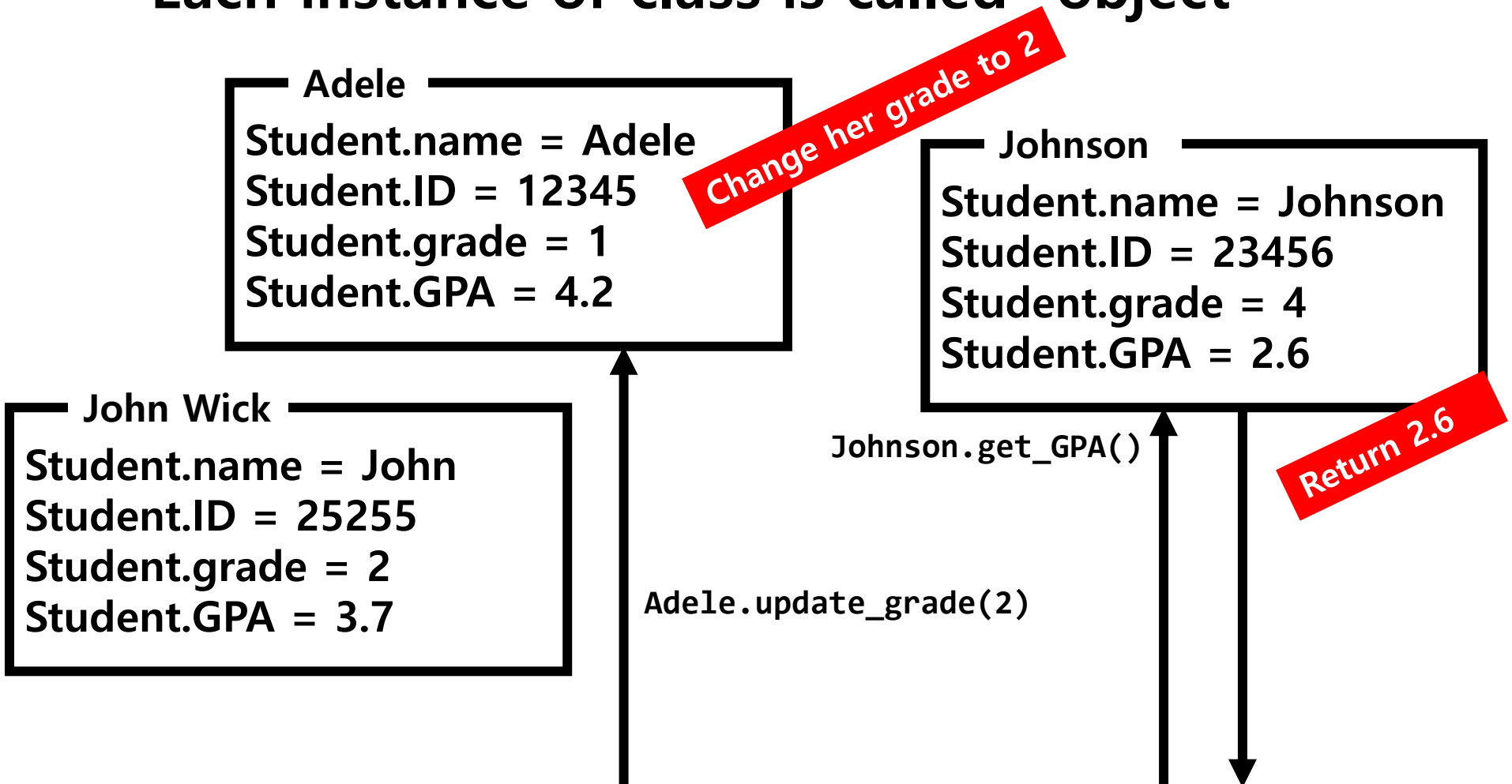
Class [1]

- **Class is a frame that contains:**
  - Several variables (member variable)
  - Several functions to deal with the data (method)

- **For example of student…**
  - Member variable: name, ID, grade, GPA, …
  - Methods: changing GPA, getting information, increasing grade, …

Class [2]

- Each instance of class is called “object”



Defining Class [1]

- We can define a class with `class` keyword

methods

```
class Student:
    def __init__(self, name_input, id_input, grade_input, gpa_input):
        self.name = name_input
        self.id = id_input
        self.grade = grade_input
        self.gpa = gpa_input
        print("New student is created!")

    def get_id(self):
        return self.id

    def update_gpa(self, new_gpa):
        self.gpa = new_gpa
```


Defining Class [2]

- We can define a class with `class` keyword

```
class Student:
    def __init__(self, name_input, id_input, grade_input, gpa_input):
        self.name = name_input
        self.id = id_input
        self.grade = grade_input
        self.gpa = gpa_input
        print("New student is created!")

    def get_id(self):
        return self.id

    def update_gpa(self, new_gpa):
        self.gpa = new_gpa
```

First parameter of method is (usually) `self`

Defining Class [3]

- We can define a class with `class` keyword

```
class Student:
    def __init__(self, name_input, id_input, grade_input, gpa_input):
        self.name = name_input
        self.id = id_input
        self.grade = grade_input
        self.gpa = gpa_input
        print("New student is created!")

    def get_id(self):
        return self.id

    def update_gpa(self, new_gpa):
        self.gpa = new_gpa
```

It is a special method (initializer)

Defining Class [4]

- We can define a class with `class` keyword

```
class Student:
    def __init__(self, name_input, id_input, grade_input, gpa_input):
        self.name = name_input
        self.id = id_input
        self.grade = grade_input
        self.gpa = gpa_input
        print("New student is created!")

    def get_id(self):
        return self.id

    def update_gpa(self, new_gpa):
        self.gpa = new_gpa
```

Declaring member variable

Using Object [1]

- **We can make an object with `(ClassName)(some arguments)`**
  - Use the parameter of `__init()__`

```
class Student:
    def __init__(self, name_input, id_input, grade_input, gpa_input):
        self.name = name_input
        self.id = id_input
        self.grade = grade_input
        self.gpa = gpa_input
        print("New student is created!")
```

```
20      Patrick = Student('a', 12345, 4, 3.27)
```


Using Object [2]

- Call method with `(ObjectName).(MethodName)(argument)`
- Note that (including `__init()__`) we omit the argument for `self`

```python
class Student:
    def __init__(self, name_input, id_input, grade_input, gpa_input):
        self.name = name_input
        self.id = id_input
        self.grade = grade_input
        self.gpa = gpa_input
        print("New student is created!")

    def get_id(self):
        return self.id

    def update_gpa(self, new_gpa):
        self.gpa = new_gpa
```

```python
Patrick = Student('a', 12345, 4, 3.27)
Patrick.update_gpa(4.30)
```


Using Object [3]

- **We can directly access to member variables**
  - With `(ObjectName).(VarName)`

```
20  Patrick = Student('a', 12345, 4, 3.27)
21  print(Patrick.name)
22  Patrick.grade = 3
23  print(Patrick.grade)
```

```
D:\temp\pythonProject\venv\Scripts\python.exe D:/
New student is created!
a
3
```

  - Of course, if this is public…

Exercises

- Some exercises for you are in “exercises”

Libraries

- **To put it simply, library is a collection of data, function, and classes.**

- **There are many of libraries for many purposes**
  - For math
  - For statistics
  - For image processing
  - For AI, ML
  - For game
  - …There's almost everything we want

Libraries: Motivation [1]

- **Why we need libraries?**

Libraries: Motivation [2]

- **Why we need libraries?**

- **Because our time is precious!!**
  - We don’t need to implement everything
  - Just use functions made by professional developers!

Using Library: math [1]

- **Use `import` keyword to bring it**
  - If we did not use the library, the font becomes gray

```
1    import math
```

  - Else…

```
1    import math
```


Using Library: math [2]

- Then use with (LibName).(Name)

- Let's use π(pi)

```
1    import math
2
3    print(math.pi)
```

```
D:\temp\pythonProject\venv\Scripts\pyt
3.141592653589793

Process finished with exit code 0
```

- How about function?

```
1    import math
2
3    print(math.log2(256))
```

```
D:\temp\pythonProject\venv\Scripts\
8.0

Process finished with exit code 0
```


Using Library: Alias

- **We can use "alias" of the library**
  - Using math.(name) every time is annoying
    - There are libraries with long name(e.g. multiprocessing, matplotlib.pyplot)
  - How about using "m" instead of "math"?

- **Use as keyword!**

```
1    import math as m
2
3    print(m.pi)
```


Using Library: from

- **Using from, we can use several items in the library**
  - Of course, we can use all items by using " * "

```
1   from math import pi
2
3   print(pi)
```

```
D:\temp\pythonProject\venv\Scripts
3.141592653589793

Process finished with exit code 0
```

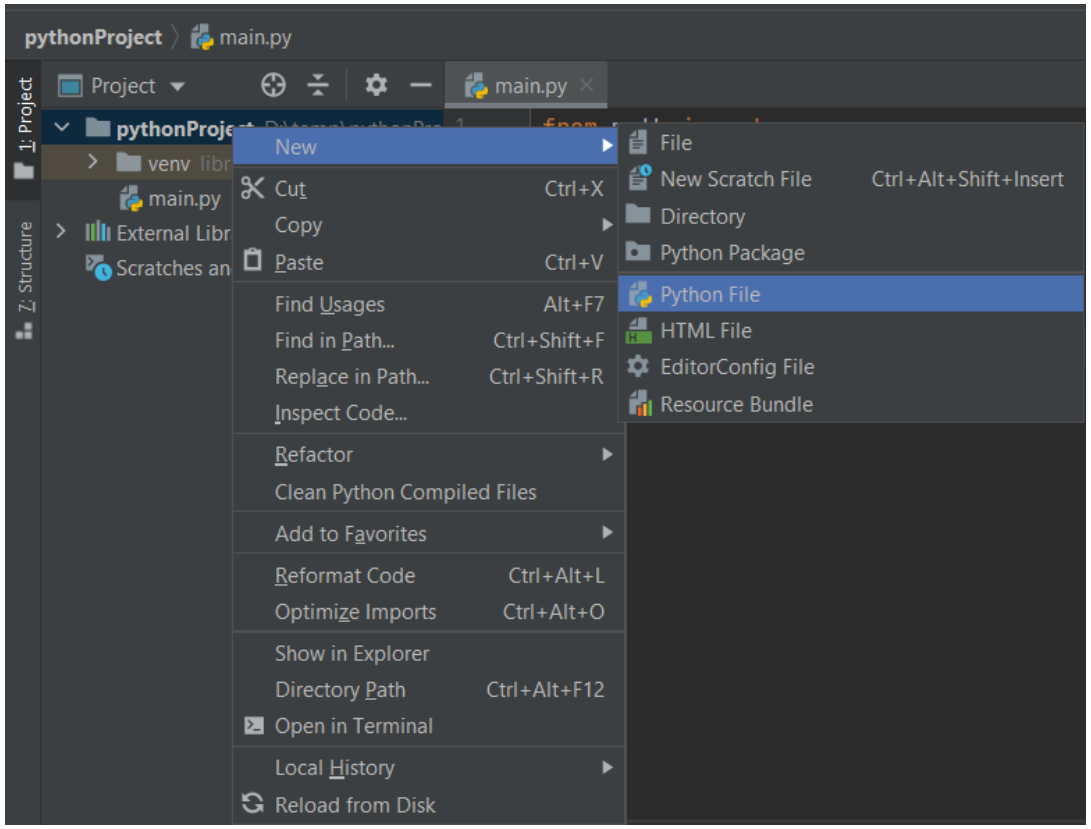
```
1   from math import *
2
3   print(sin(pi / 2))
```

```
D:\temp\pythonProject\venv\Scr
1.0

Process finished with exit coc
```

- **Note that we do not use "math." !!**

Making Library [1]

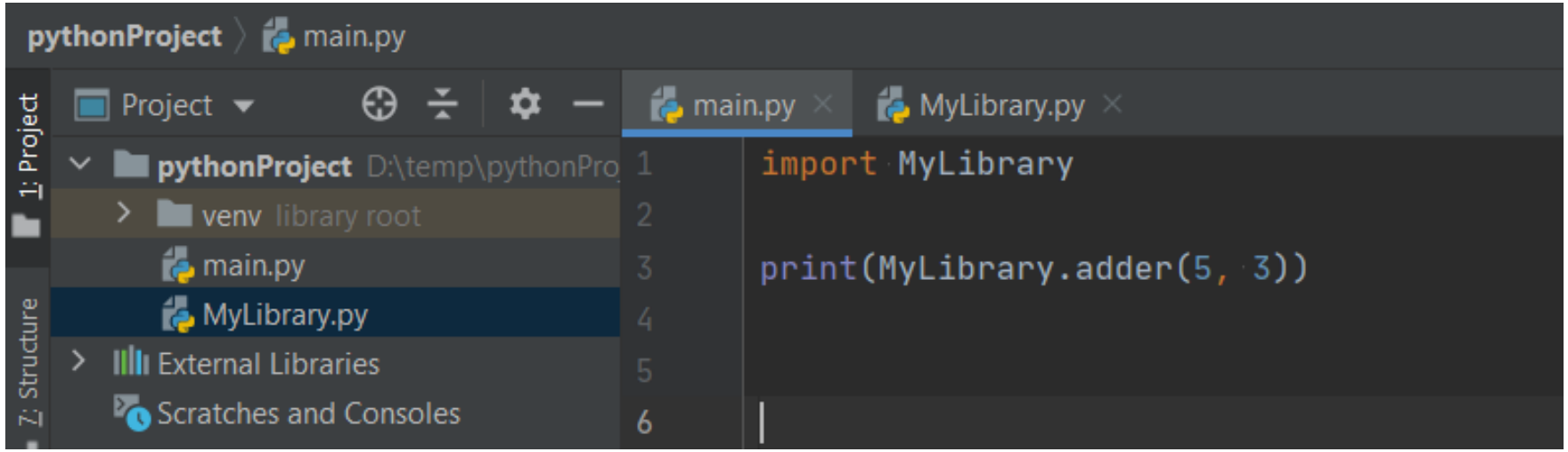
- **We can make our own library**
  - Just define some functions / classes / variables in a file!

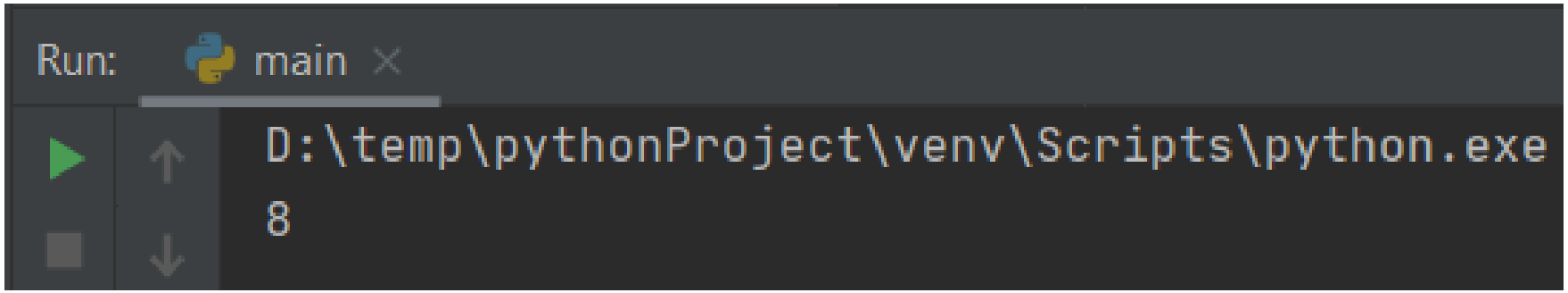



```python
my_awesome_number = 42

def adder(a, b):
    return a + b

class Human:
    def __init__(self, my_name):
        self.name = my_name
    def print_name(self):
        print(self.name)
```

Making Library [2]

- **Then, how can we use it?**
  - Just import! (If it is in same directory)





Exercises

- Some exercises for you are in “exercises”

In the Real-time Class…

- **We will have a lab session (mini project)**
  - We will upload the material soon

- **Before that, please review what we covered**
  - Supplement and exercises were uploaded
  - Feel free to ask us! Via…
    - Comment in the page (recommended!)
    - E-mail

Thank you