



THE DARIU FOUNDATION

Basic Python Programming Tutorial



Contents

I	Basic Python Commands	7
Chapter 1.	Python and Pycharm Installation Instructions	9
1	Introduction	10
2	Installing Python and Pycharm on a 32-bit machine	11
3	Installing Python on 64-bit Window	12
4	Python GUI programming interface	14
5	Installing PyCharm on Windows 64 bit	15
6	Instructions for setting up and running the program with PyCharm	19
7	The first program on PyCharm	23
8	Some other operations on PyCharm	24
9	Review questions	26
Chapter 2.	Show results in Python	27
1	Introduction	28
2	Show more information	28
3	Display with separators	28
4	Display with end character	29
5	Display with decimals	29
6	Review questions	30
Chapter 3.	Data Input and Data Type	31
1	Introduction	32
2	Input statement	32
3	Integer type on Python	32
3.1	Declare an integer variable	32
3.2	Enter integer from the keyboard	33
3.3	Integer operations	33
4	Real number on Python	34
4.1	Declaring a variable real number	34
4.2	Enter real numbers from the keyboard	34
4.3	Operations on real numbers	34
5	Handle input errors	35
6	Review questions	36
Chapter 4.	IF Conditional Statement	39
1	Introduction	40
2	Command if	40

3	Statement if else	41
4	If elif else statement	41
5	Nested statements	42
6	Exercise	43
7	Review questions	45
Chapter 5. One-dimensional array - FOR loop structure		47
1	Introduction	48
2	Declare and traverse a one-dimensional array	48
3	For loop structure	48
4	Enter array from keyboard	49
5	Exercise	50
6	Review questions	53
Chapter 6. Multidimensional array - nested FOR		55
1	Introduction	56
2	Add a Python file	56
3	Declare and traverse multidimensional arrays	58
4	Traverse a multidimensional array	58
5	Enter a multidimensional array from the keyboard	59
6	Exercise	60
7	Review questions	63
Chapter 7. While loop structure		65
1	Introduction	66
2	Loop syntax while	66
2.1	break and continue statements within while	67
2.1.1	Command break	67
2.1.2	Command continue	67
2.2	Use while on a line	67
3	Loop syntax while-else	67
4	Exercise	68
5	Review questions	70
Chapter 8. File operations		73
1	Introduction	74
2	Write data to File	74
3	Read data from File	75
4	Read 1-dimensional array from File	76
5	Read multidimensional array from File	77
6	Exercise	78
7	Review questions	79
Chapter 9. Functions and function calls		81
1	Introduction	82
2	Function definition	82
3	Function Call	82
3.1	Default Arguments	83
3.2	Return Value	83
3.3	Command pass	84

4	Write a function to calculate factorial	85
4.1	Write function using for loop	85
4.2	Write function using recursion	85
5	Exercise	86
6	Review questions	87
Chapter 10. Python Advanced Data Structures		91
1	Introduction	92
2	String (string)	92
2.1	Concatenate string, change or delete string	92
2.2	Methods for string variables	93
3	Tuple data structure	93
3.1	Initialize and retrieve elements in a tuple	93
3.2	Tuple operations	94
3.3	When to use Tuple	94
4	Set Data Structure	95
4.1	Initialize and retrieve elements of a set	95
4.2	Methods of set	95
4.3	Set operations	96
4.3.1	Union	96
4.3.2	Intersection	96
4.3.3	Difference	96
4.3.4	Symmetric difference	96
4.4	When to use set	97
5	Dictionary Data Structure	97
5.1	Initialize and retrieve dictionary elements	97
5.2	Add and update dictionary elements	97
5.3	Remove an element from the dictionary	98
5.4	Some common dictionary methods	98
5.5	When to use dictionary	98
6	Exercise	99
7	Review questions	99
II Application Projects		101
Chapter 11. Python's Virtual Assistant - Text to Speech		103
1	Introduction	104
2	Create a new application	104
3	Install library	105
4	Program implementation	106
5	Review questions	107
Chapter 12. Python's Virtual Assistant - Voice Recognition		109
1	Introduction	110
2	Install library	110
3	Program implementation	111
4	Review questions	113

Chapter 13. Python's Virtual Assistant - Building intelligence	115
1 Introduction	116
2 Retrieve current date	116
3 Complete the program	116
4 Review questions	118
Chapter 14. Python Interface Programming	119
1 Introduction	120
2 Interface design	120
3 Add an interface object to the window	121
4 Review questions	124
Chapter 15. Create button handler	125
1 Introduction	126
2 Declaring the handler function	126
3 Implement the handler function for the button	127
4 Exercise	127
5 Review questions	128
Chapter 16. Interface error handling	129
1 Introduction	130
2 Handle input errors	130
3 Review questions	132

Part I

Basic Python Commands

CHAPTER 1



Python and Pycharm Installation Instructions

pythonTM
Package
Index

1 Introduction

Python is a high-level programming language and is more accessible to people who are just starting to learn programming languages. If comparing Python with traditional languages like Pascal or C, then Python's level of complexity is lower. Some of the advantages of the Python programming language can be listed as follows:

- Python has a very simple syntax. It is much easier to read and write when compared to other programming languages like Pascal or C. Although sometimes this simplicity can cause some problems in program management. However, Python makes programming simpler, especially input and output tasks, allowing to focus on solutions and not the syntax of the programming language. In an abstract comparison, Python makes it easier for you to "communicate the language", rather than having to traditionally "learn the language".
- Widely open source: This is very important for beginners. Because it is open source, we can not only use software and programs written in Python, but also change its source code. Python has a huge community that is constantly improving it with each update. A lot of open source code related to artificial intelligence or speech recognition is available on Python, so that users can develop very advanced applications.
- Cross-Platform Compatibility: Many systems from computers to embedded boards, even low-resource boards like MicroBit, also support the Python programming language. Therefore, we can imagine that the software can only be developed once and can be used for many different platforms. With only very small changes, a piece of software running on a computer can also run on a mini board.

Thus, it can be seen that the Python programming language is a very broad concept. Depending on the application development platform, appropriate accompanying software will be used, as illustrated in Figure 1.1. The software here not only plays the role of a program for us to write programs (also known as programming), but it also plays the role of compiling from a programming language to an executable language.



Figure 1.1: Programming Languages and Compilers

As illustrated in Figure 1.1, when programming Python on a computer, we will need a program that translates from Python language into a language that the computer can understand. And this program, the most popular currently is Python3. However, if we want

to program in Python but for MicroBit circuit, we will need another program, such as Mu. This program will translate from Python language into a language that the MicroBit circuit can understand. In the figure Figure 1.1 we call the MicroBit Computer or circuit the execution platform.

Finally, there is a method of translating from the Python language into a language that the execution platform can understand, called an interpreter. As an interpreter, the software translates the first python statement, and then executes it. Then, translate the second statement again, and execute until the end of the program. This is a very different point compared to Pascal language, the compiler will translate the program and then execute it. Also for this reason, Python is compatible with many execution platforms, because the Python program is essentially just a text file. Only when executed will it be translated into a language understood by the hardware platform.

The instructions below will detail the steps to install the Python3 interpreter. Actually, after installing this interpreter, we already have an interface for programming in Python language. However, the interface is quite poor, so we encourage you to install PyCharm, a software that is highly recommended by the Python community.

2 Installing Python and Pycharm on a 32-bit machine

Currently, for 32-bit models, we find a general installation package to install as follows:

Step 1: Download and install the Visual Studio add-in package by downloading the installation file from the following path:

<https://ubc.sgp1.cdn.digitaloceanspaces.com/DARIU/PyCharm32/VC.exe>

Extract and install the file as usual.

Step 2: Download and install the JAVA virtual machine add-in package at the following path:

<https://ubc.sgp1.cdn.digitaloceanspaces.com/DARIU/PyCharm32/jdk.exe>

Just like step 1, we also extract and install as usual.

Step 3: Install Python version 3.8.5 at the following path:

<https://ubc.sgp1.cdn.digitaloceanspaces.com/DARIU/PyCharm32/python32.exe>

During Python installation, check **Install for all user** and **Add Python to PATH** if you have this message.

Step 4: Install the 32-bit PyCharm software by downloading it from the following link:

<https://ubc.sgp1.cdn.digitaloceanspaces.com/DARIU/PyCharm32/pycharm32.exe>

During the installation process, if the following message appears, please select the options marked as follows:

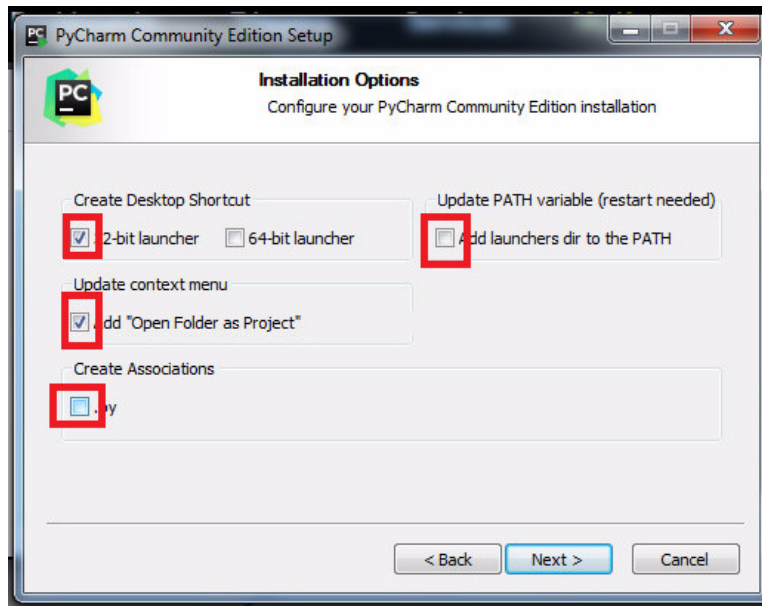


Figure 1.2: Check options for PyCharm

3 Installing Python on 64-bit Window

- Step 1: Download Python version 3.8.5 from the following path:

https://ubc.sgp1.cdn.digitaloceanspaces.com/DARIU/python_3_8_5.exe

In case we want to update to a newer version, we can go to Python's homepage <http://www.python.org/>, then select **Downloads** to select a session. Python version we want to install. **However, we recommend using version 3.8.5 in this tutorial.**

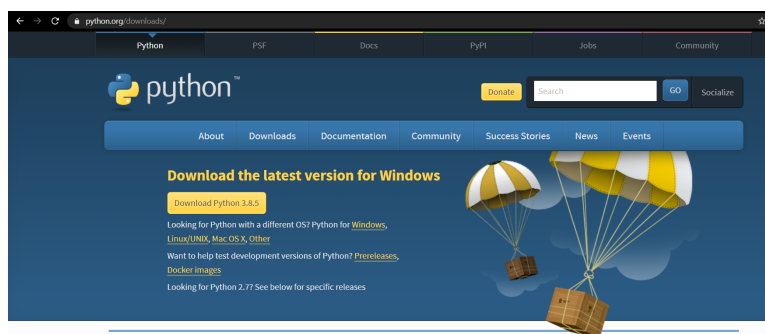


Figure 1.3: Download Python software from homepage <http://www.python.org/>

- Step 2: When the download is complete, proceed to run the installation file with the .exe extension just downloaded in step 1. Click Install Now to install. **We need to select the last 2 options** (Install for all users and Add Python to PATH), as shown in the image below.
- Step 3: We can see the Python installation progress.
- Step 4: When the installation is complete, we will see a message screen like below. Then click "Close" (in the lower right corner) to complete the installation.

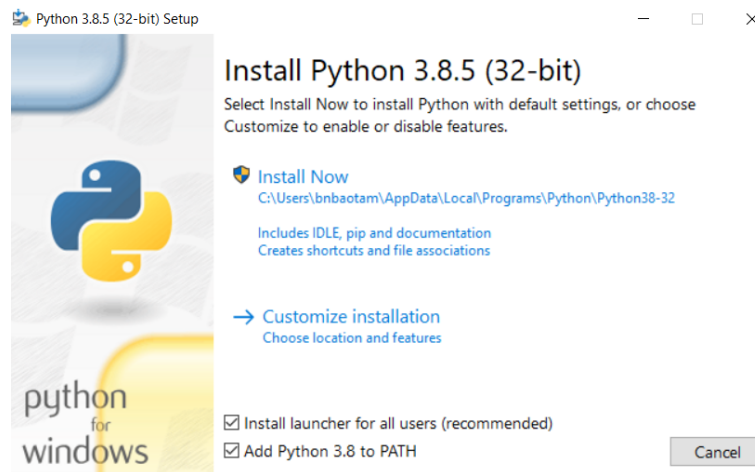


Figure 1.4: Select both options, before clicking Install Now to install

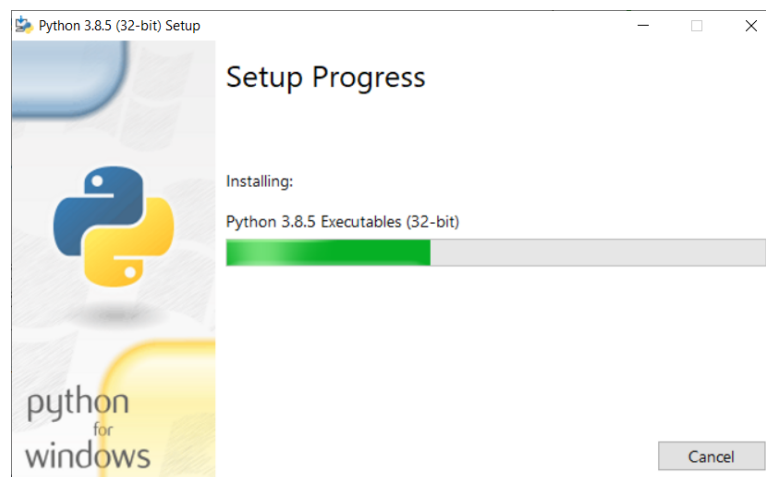


Figure 1.5: Waiting for the program to install Python

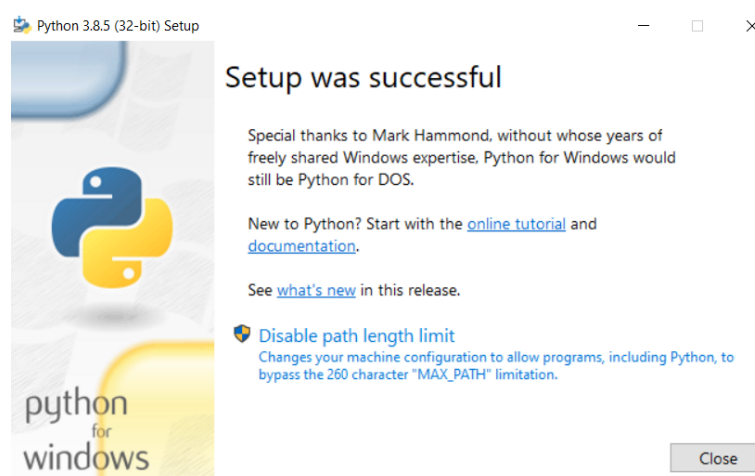


Figure 1.6: Python installation is successful

4 Python GUI programming interface

As soon as install Python, we can start programming. From the Windows start button, we search with the keyword Python GUI to find this application, as shown in Figure 14.1.

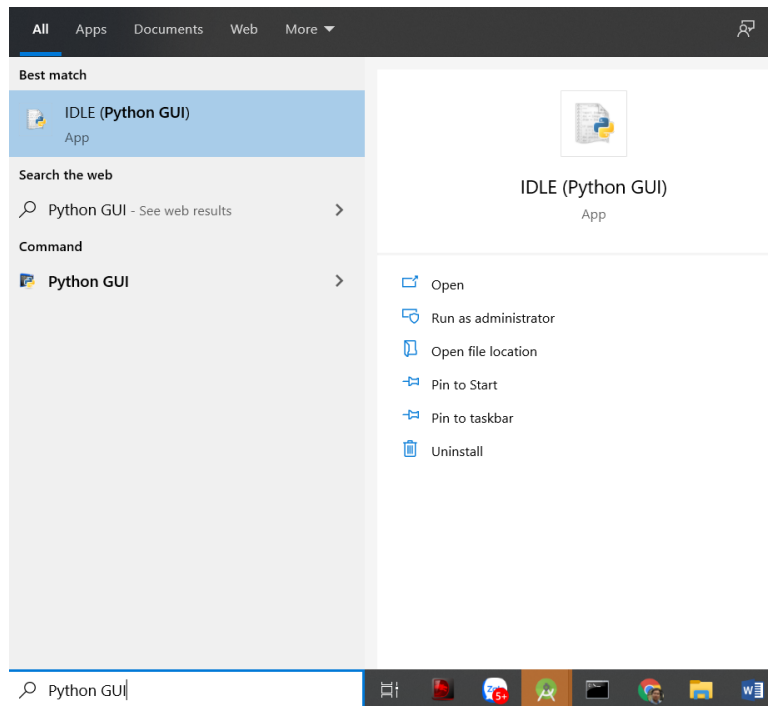


Figure 1.7: Python GUI Program

The Python Shell interface appears, from the File menu, select New Window, as illustrated in Figure 14.3.

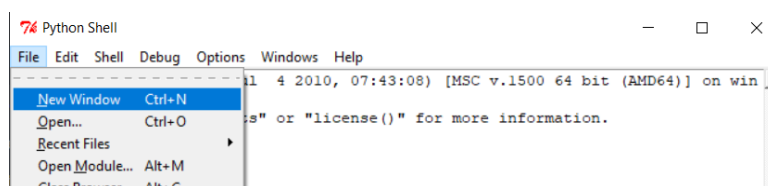


Figure 1.8: Open a window to program in the Python IDE

The first program is very simple, only one statement is **print("Hello")**. To run the program, select Run, select Run mode or press F5. The results of the program will be displayed on the Python shell. Obviously, this programming interface is quite poor and doesn't have much support when writing commands, which we will see clearly in PyCharm software. However, this software can also be used temporarily when you have not successfully installed PyCharm. In the next tutorial, we will show you how to install and write programs on PyCharm.

5 Installing PyCharm on Windows 64 bit

- Step 1: From the homepage www.jetbrains.com/pycharm/download/, click the “Download” button, and select the free version **Community**, as shown in the image below.

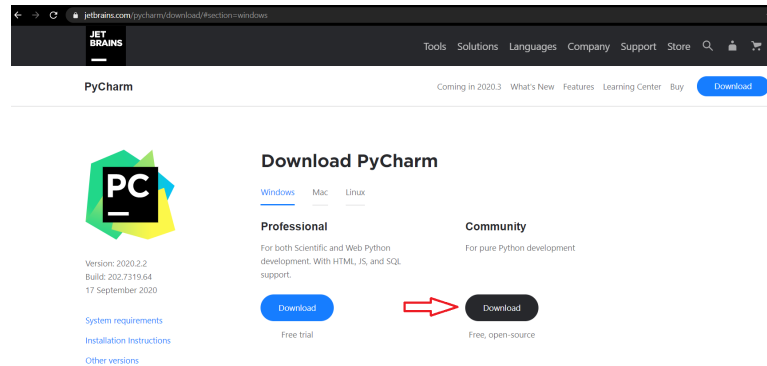


Figure 1.9: Download Pycharm from the homepage

- Step 2: When the download is complete, run to install PyCharm. An installation window will appear, click “Next”.

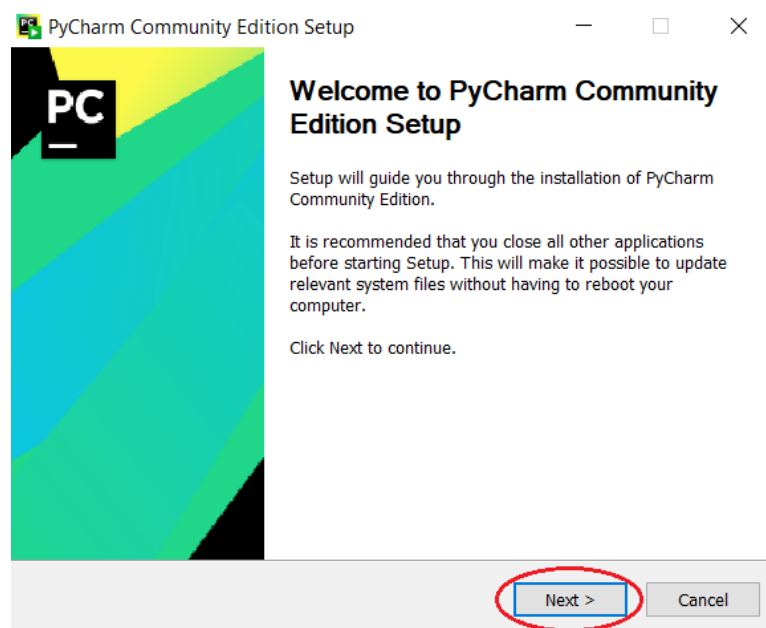


Figure 1.10: Press "Next" to continue

- Step 3: On the next screen, we can change the path to install PyCharm or leave it as default. Then click “Next”.

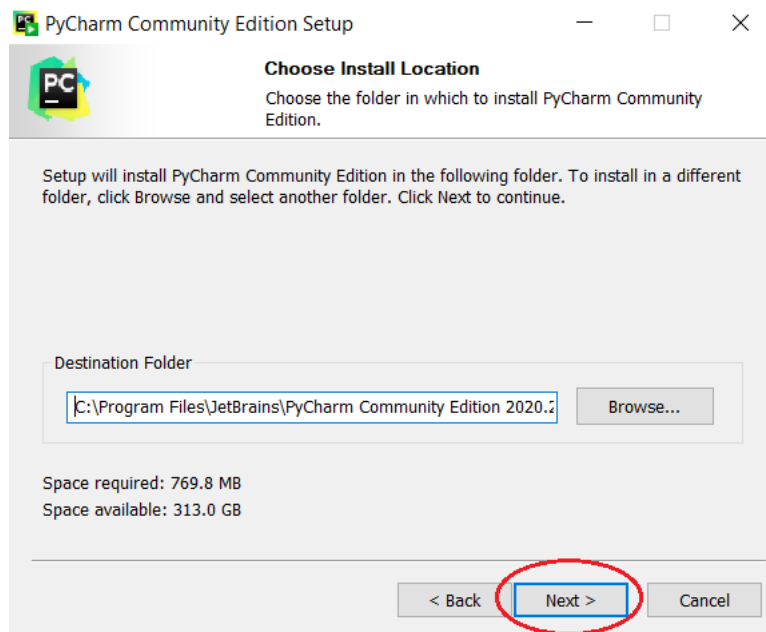


Figure 1.11: Press "Next" to continue

- Step 4: We will see a screen with a few options, we select the boxes as shown (select the same if machine only shows 32-bit instead of 64-bit). Then click “Next”.

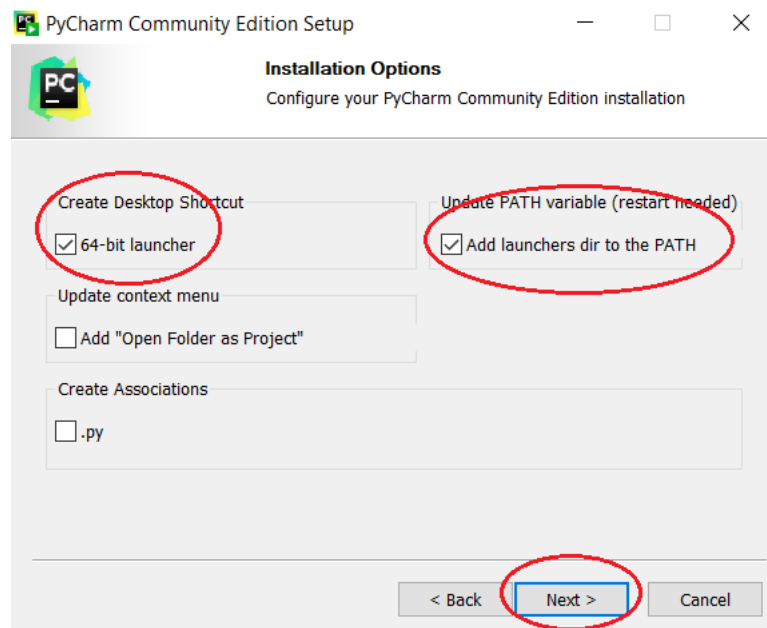


Figure 1.12: Press "Next" to continue

- Besides, if in this step, in the settings frame we have more options as shown below. That means we have not installed the Java library, please choose to install the Java library automatically.

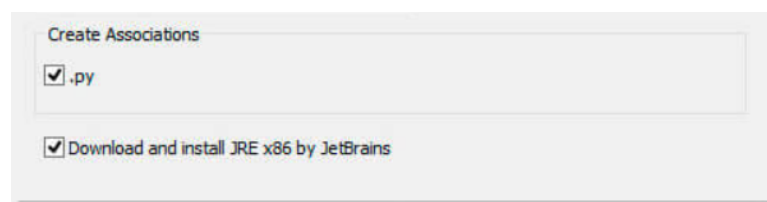


Figure 1.13: Tick for 2 options

- Step 5: Select the folder in the Windows start menu. Leave the default as the JetBrains folder and click “Install”.

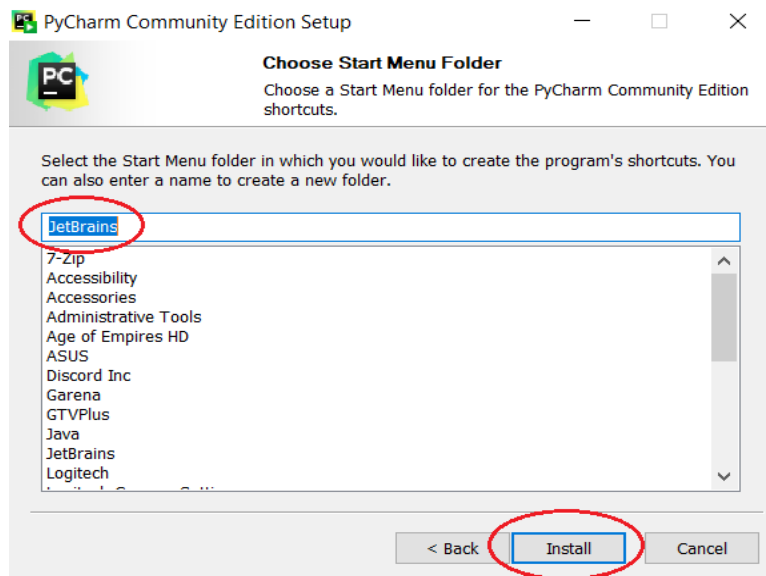


Figure 1.14: Click "Install" to proceed with the installation

- Step 6: Wait for the installation to complete.

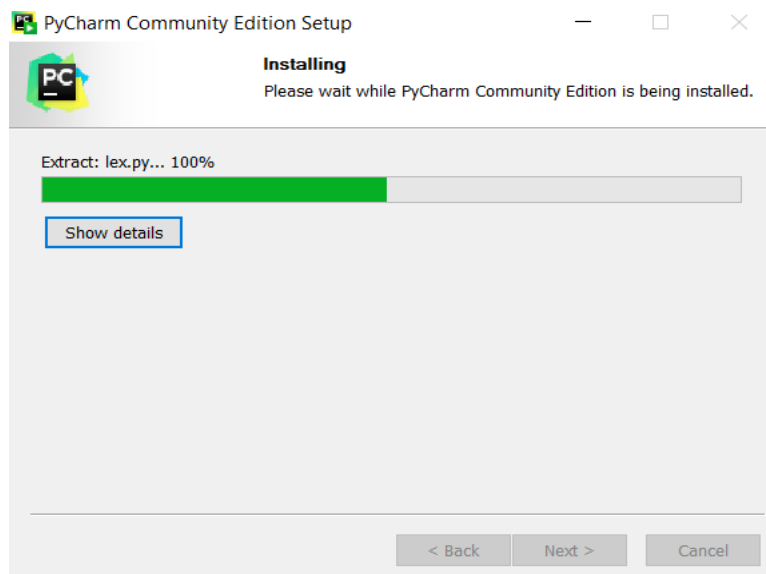


Figure 1.15: Waiting for the program to install Pycharm

- Step 7: When the installation is complete, we will see the message screen as below. PyCharm asks if we want to restart the machine or not. We can choose Reboot Now to restart the computer to complete the installation process.

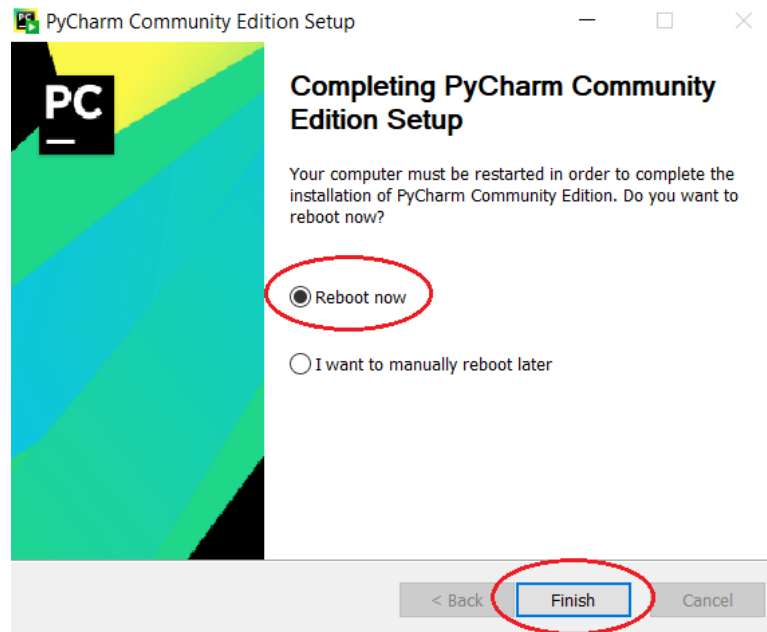


Figure 1.16: Press "Finish" to complete the installation

6 Instructions for setting up and running the program with PyCharm

So we have basically installed PyCharm.

But to know if you have installed it correctly, we need to try to see if python program can run with the PyCharm just installed.

- Step 1: Open the successfully installed PyCharm program on computer.
- Step 2: We will be asked "Do you want to import existing settings?". If the installation is completely new, we select the item Do not import settings, then click OK, as instructed in Figure 1.17
In the privacy policy section (if any), click confirm and click Continue to continue.
- Step 3: In the PyCharm customization screen, select Skip Remaining and Set Defaults to choose the default settings.

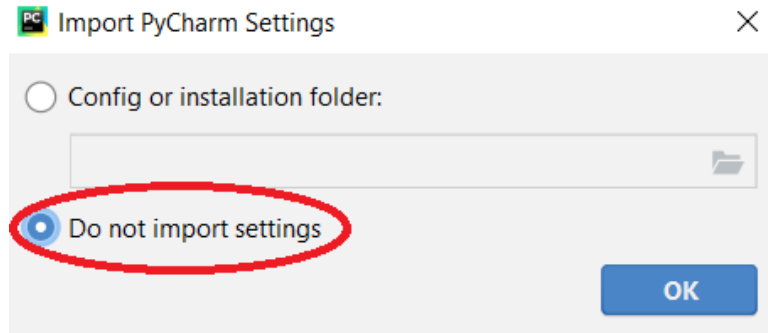


Figure 1.17: Do not import old settings when installing new



Figure 1.18: Agree with PyCharm's policy

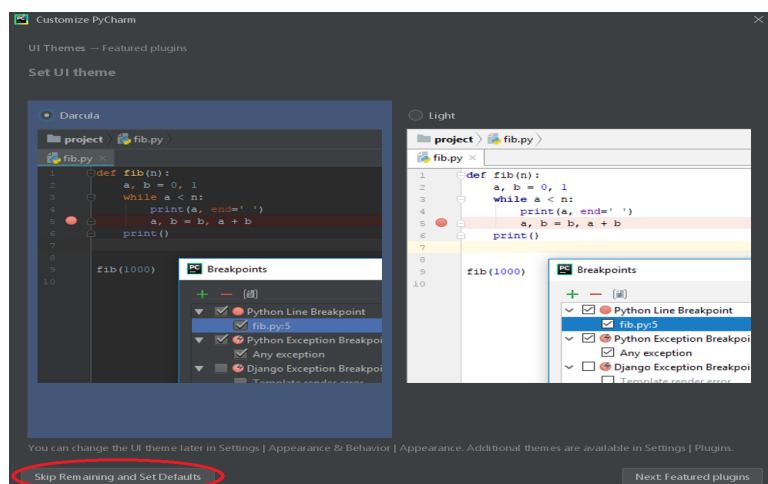


Figure 1.19: Select default configuration for PyCharm software

- Step 4: After that, PyCharm's welcome screen, we select Create New Project to create a new Project.

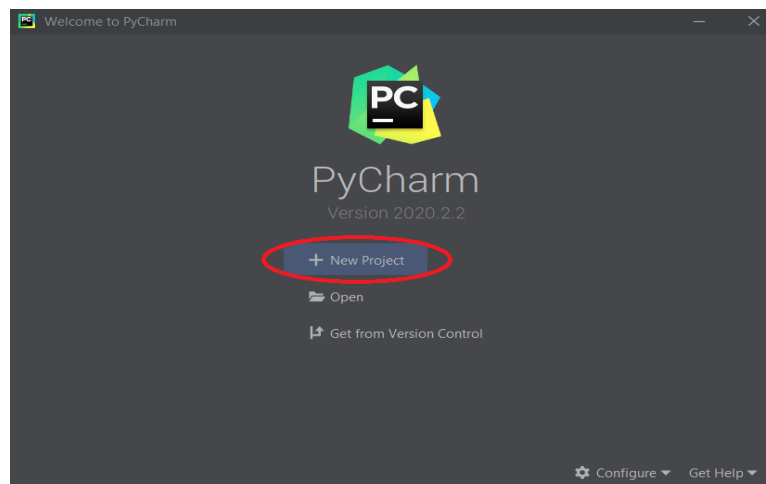


Figure 1.20: Create a new project by clicking New Project

- Step 5: We choose the folder to save the project at **Location**. At the same time, we check the environmental information below, see if it is the same as the guide image or not. Usually the default values will be correct and we do not have to edit anything. Finally, we click on the **Create** button.

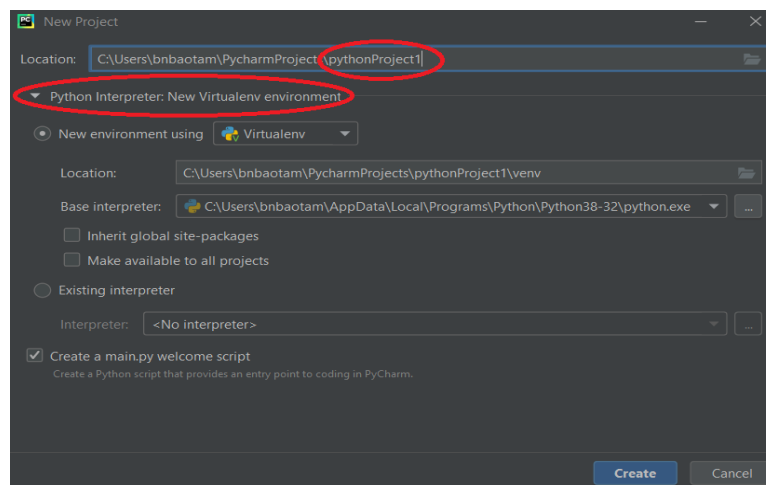


Figure 1.21: Select the path for the project and check the environment information

- Step 6: After the above process is completed, a dialog box as shown below may appear. If do not like it appear again. We can check the box that does not show tips in the lower left corner and select close to exit.

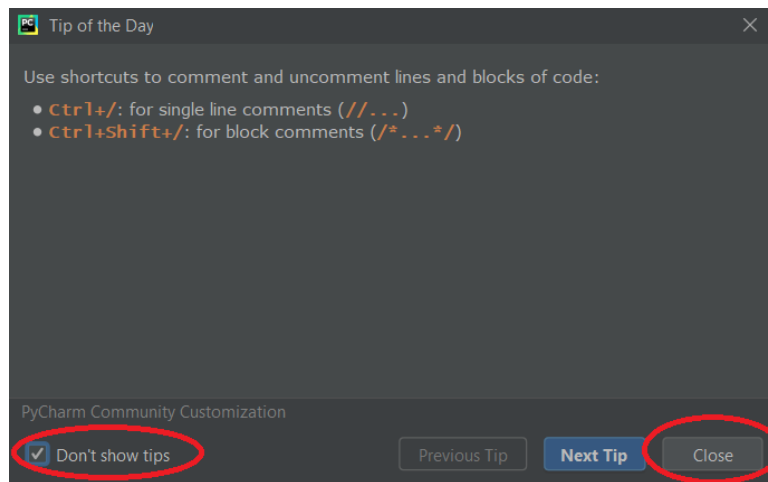


Figure 1.22: Turn off dialog with instructions for programming tips

The new project will be created on PyCharm as shown below.

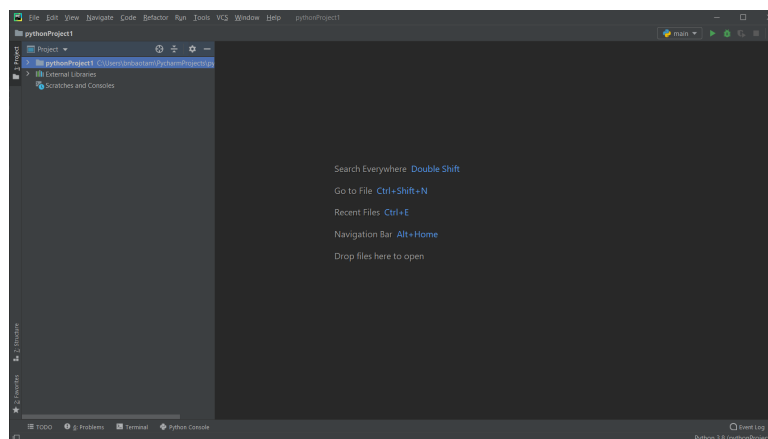


Figure 1.23: New project created successfully on PyCharm

- Step 7: To see the content of the sample code when we create the project, click on the project section (circled in red) we just created to see all the files in the project and select the **main.py**. Then we get the result as shown below.
- Step 8: To display more useful buttons like running the program, we go to View -> Appearance -> select Toolbar.

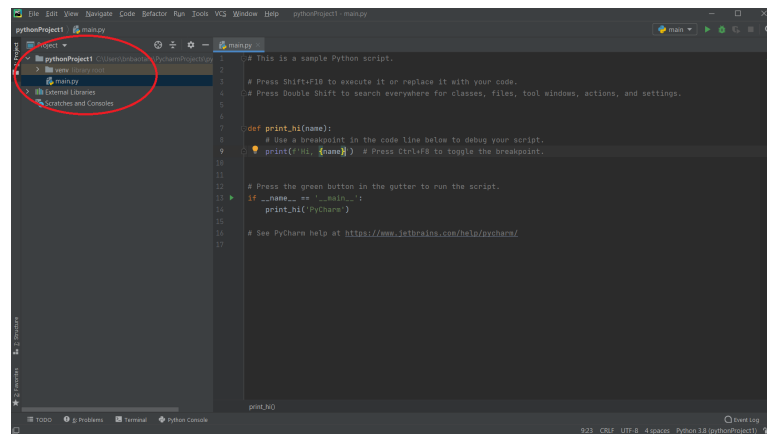


Figure 1.24: Click on the Project section in the red circled area

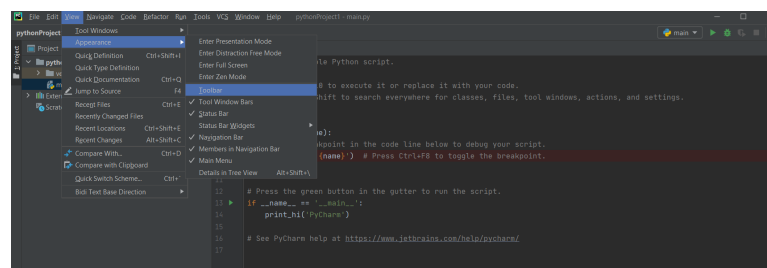


Figure 1.25: Show Toolbar on PyCharm

- Step 9: After displaying the toolbar, we select the green arrow. Then the program will be executed and the results will be displayed below as shown.

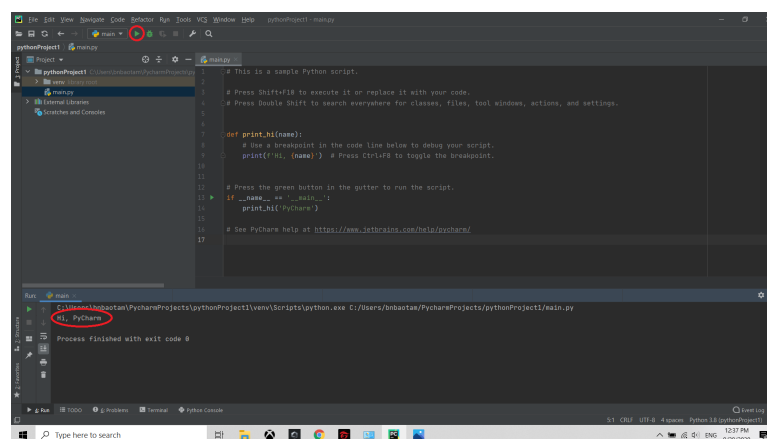


Figure 1.26: Press the Run button to test the default program on PyCharm

So we have installed and successfully tested a program on PyCharm.

7 The first program on PyCharm

When creating a new project on PyCharm, the software automatically creates for us the default file **main.py** along with some sample program snippets. We can double-click this

file, delete all the content and start with our program.

In this first program, we also simply print a sentence to the screen. Our program has only one instruction, **print("Hello World")**. And here is the output after running the program:

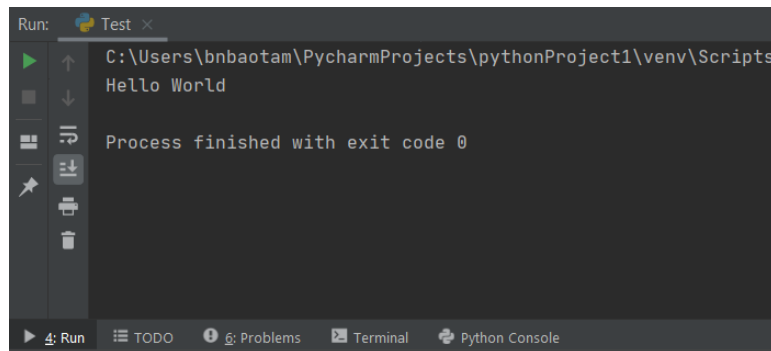


Figure 1.27: Print output for the first program on PyCharm

8 Some other operations on PyCharm

PyCharm's default color tone is dark (black background with white text). In case we want to change the display color tone, we can go to File -> Settings. With the new window that appears, continue to select Appearance and choose your preferred color tone, as instructed in Figure 1.28

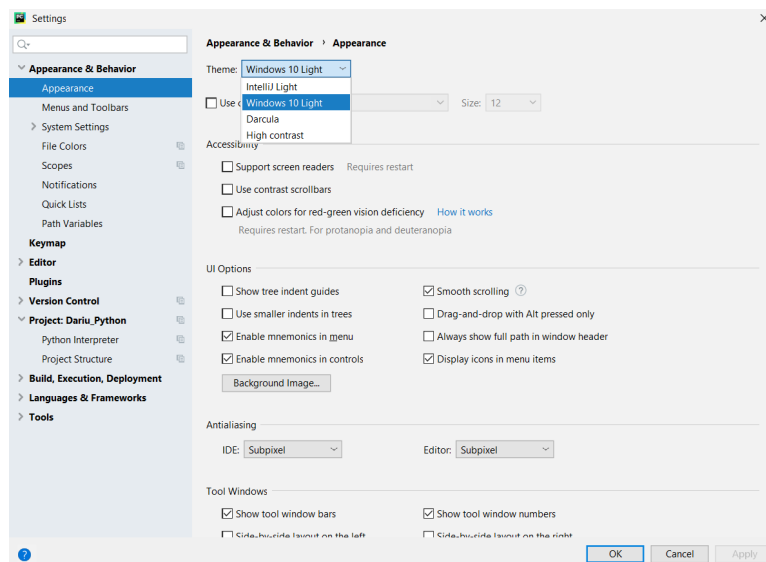


Figure 1.28: Change the color tone displayed on PyCharm

In case we want to change the size of the editor, so that the programming is easy to see, from the Settings interface in Figure 1.28, we continue through the Editor and select Font and change its Size, as illustrated in Figure 1.29 is 30.

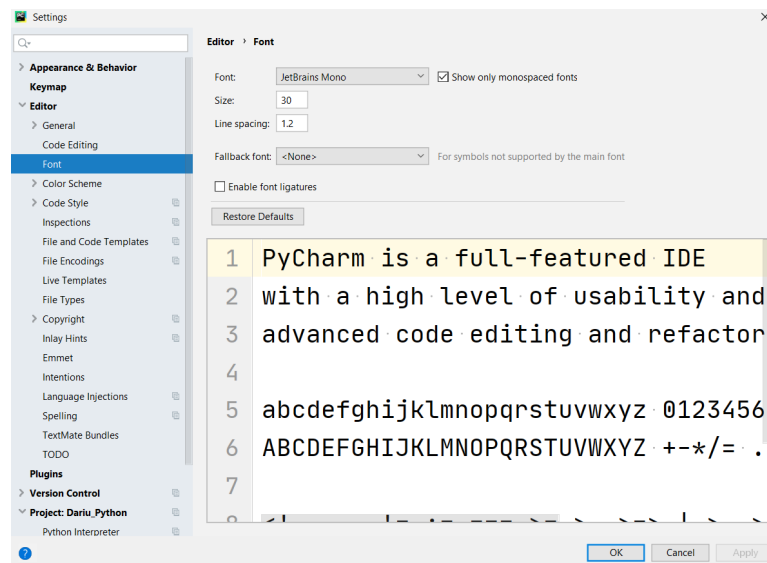


Figure 1.29: Change font size when programming

9 Review questions

1. Which statement is true about the Python language?
 - A. Low Level Programming Language
 - B. High Level Programming Language
 - C. Machine language
 - D. Language MicroBit
2. What are the advantages of the Python language over Pascal?
 - A. Simple and accessible
 - B. Cross-platform compatibility
 - C. Open source with many support libraries
 - D. All correct
3. To convert from Python to a computer-understandable language, we need:
 - A. Python3 Compiler
 - B. Python3 interpreter
 - C. Both programs above
 - D. All wrong
4. Which characteristic is true for the interpreter?
 - A. Translate each command and execute it
 - B. Translate the entire program and execute it
 - C. Depending on the case, one of the two methods above will be selected
 - D. All correct
5. Which is true for the compiler?
 - A. Translate each command and execute it
 - B. Translate the entire program and execute it
 - C. Depending on the case, one of the two methods above will be selected
 - D. All correct
6. The Python programming language is:
 - A. Compiled language
 - B. Interpretation language
 - C. All correct
 - D. All wrong
7. The Pascal programming language is:
 - A. Compiled language
 - B. Interpreted language
 - C. All correct
 - D. All wrong

Answer

1. B 2. D 3. B 4. A 5. B 6. B 7. A

CHAPTER 2



Show results in Python

pythonTM
Package
Index

1 Introduction

To learn a programming language in general and python in particular, usually we will start with the display statement to print the results to the screen. In fact, in a computer system with three characteristic components: Input - Processing - Output, the last step is the simplest step, followed by the input and final data. For the same reason, we often learn the display statement first, the simplest statement belonging to the group of output results (Output).

In the previous tutorial, we actually used the display command **print** to print the word "Hello" to the screen. In this article, we will cover the options of the print statement in more detail.

The commands in the instructions below will be imported directly into the default file **main.py**. Then, when running the program, the results will be printed in the window directly below the program.

2 Show more information

In addition to printing a string as in the previous lesson, the print function can print multiple information, each separated by **comma** (,), like the example in the following program:

```
1 print("Hello", 10, 12.1, "Goodbye")
```

Program 2.1: Print on Python

Besides strings, the print function can also print integers and decimals. Python will automatically **add a space** between the information to be printed. A variable is also considered an information, and can be inserted into the print statement in the same way as above. An example is as follows:

```
1 x = 20
2 print("Hello", 10, 12.1, "Goodbye", x)
```

Program 2.2: Print the value of variable

In the above example, *x* is a variable. However, on Python, the declaration of a variable has been greatly simplified. We don't even need to specify the data type for the variable. The details of declaring variables and data types will be presented in detail in the following articles.

3 Display with separators

Instead of each printed information being separated by a space as above, we can also customize it with another string of characters, based on the **sep** option as in the example below:

```
1 print("Hello", "Goodbye", sep="---")
```

Program 2.3: Print results with separators

4 Display with end character

Finally, we can also add the option **end** to define the end character of the string to be printed, like the following example:

```
1 print("Hello", "Goodbye", sep="---", end="!!!")
```

Program 2.4: Print results with end option

Of course, as optional, `sep` and `end` may or may not appear together, or even appear together in the print statement. The only requirement is that these options be separated by commas and **no order constraints**. Program ?? will also print the same result as Program ??

```
1 print("Hello", "Goodbye", end="!!!", sep="---")
```

Program 2.5: Options are unordered

5 Display with decimals

One of the important operations when calculating, the result can be a decimal number with many digits after the comma. In some cases, we only need to print 2 digits after the comma. The support print statement in this case will look like the example below:

```
1 a = 3.14165
2 print("Result is %.2f" %a)
```

Program 2.6: Customize number of digits after decimal point

Here, `%.2f` is the format of the data to be printed. The result printed to the screen will only take 2 decimal places of the variable *a*. An important note is that **result will be rounded**.

6 Review questions

1. What is the command to print the output to the screen in Python?
A. print
B. Log
C. Out
D. All correct
2. What is the output of `print("Hello", "123")`?
A. Hello
B. 123
C. Hello 123
D. Wrong syntax
3. What is the output of the command `print("Hello", "123", sep="*")`?
A. Hello 123 *
B. Hello * 123
C. Hello * 123 *
D. Wrong syntax
4. What is the output of the command `print("Hello", "123", end="*")`?
A. Hello 123 *
B. Hello * 123
C. Hello * 123 *
D. Wrong syntax
5. What is the output of the command `print("Hello", "123", sep = "*", end="*")`?
A. Hello 123 *
B. Hello * 123
C. Hello * 123 *
D. Wrong syntax
6. What is the output of the command `print("Hello", "123", end = "*", sep="*")`?
A. Hello 123 *
B. Hello * 123
C. Hello * 123 *
D. Wrong syntax
7. What is the output of the command `print("Vi du %.2f", % 3.14999)`?
A. to go 3.14
B. For example 3.15
C. Travel 3.14999
D. Wrong syntax

Answer

1. A 2. C 3. B 4. A 5. C 6. C 7. B

CHAPTER 3



Data Input and Data Type

pythonTM
Package
Index

1 Introduction

Before starting to process the algorithm, we have to go through the step of entering the input data for the problem. Although it is the first process of a system, its complexity is much higher than data export. One of the difficulties for this process, is that the user can enter the wrong data, or do not follow the data syntax of the program. This can lead to faulty data processing, which can even crash our program. Therefore, when doing large programs, it is always accompanied by the input process, which is the input data error handling process.

In this tutorial, we focus on teaching input data from the keyboard, and focus only on numeric data, as well as operations for numeric data. A simple but effective technique for error handling of input data will also be introduced. More complex data structures will be presented in the following tutorials.

2 Input statement

To initialize the variable, we can use the assignment statement to assign a value to the variable. Input programs allow input from the keyboard or from the disk to be assigned to variables, making the program flexible, able to calculate with many different input data sets.

The command to receive data from the keyboard is **input**, with the following syntax:

```
<variable name> = input(<Notice>)
```

Note, the input value from the keyboard is a string value, so if we want to return a value of any type, we must cast it to that value. Usually, we also add a message to instruct the user with the necessary information for input, such as the example below:

- Integer type: `a = int(input("Enter integer a: "))`
- Real type: `b = float(input("Enter real number b: "))`

In this tutorial, we will learn the two most common numeric data types when starting out with programming languages, integer types and real numbers, which will be presented in the next 2 sections.

3 Integer type on Python

3.1 Declare an integer variable

With its flexibility, Python allows us to neither declare a variable nor define its data type. Simply by an assignment, a variable will be created and Python automatically chooses its data type, depending on the value of the assignment. For Python programmers, declaring an integer variable will often use the following assignment statement:


```

1 #Declaring a, b as integer
2 a = 5
3 b = int(6)
4 print(a, b)

```

Program 3.1: Declare a variable of type integer

More traditionally, like the example above, we can specify the type of the variable, by adding the keyword **int**. However this isn't really of much use with variable declaration statements. The statement **b = int(6)** actually has a larger meaning, is to convert everything in brackets to integers. It will have more use in the next tutorial, which is to read variables from the keyboard.

3.2 Enter integer from the keyboard

A small program, to get integer from keyboard input is as follows:

```

1 a = int(input("Input integer a: "))
2 print(a)

```

Program 3.2: Get integer from keyboard

With this function, we must have the keyword **int**, not omit it like the variable declaration. The reason is that what is entered from the keyboard, is a string type. The **int** keyword in front will do the conversion from string to numeric. Therefore, we can enter the integer 6 or the decimal number 6.1, the value of the variable **a** will still only have the value 6.

3.3 Integer operations

Before experimenting with integer operations, we will execute a small program, which is to enter two numbers **n1** and **n2** from the keyboard, and print the sum of the two numbers. The example program is as follows:

```

1 n1 = int(input("First integer: "))
2 n2 = int(input("Second integer: "))
3
4 print("Total of 2 numbers: ", n1+n2)

```

Program 3.3: Simple calculation program on 2 integers

Common operations on integers are listed as follows:

- **+** : Addition
- **-** : Subtraction
- ***** : Multiplication
- **//** : Round division
- **/** : Divide real numbers
- **%** : Divide by remainder
- ****** : Exponential math

Now we can try changing the math in Program 3.3 to better understand the maths listed above.

4 Real number on Python

4.1 Declaring a variable real number

Similar to when working with integers, Python allows us to neither declare a variable nor define its data type. Simply by an assignment, a variable will be created and Python automatically chooses its data type, depending on the value of the assignment. For Python programmers, declaring a variable of type real will often use the following assignment statement:

```
1 #Declaring a, b as real number
2 a = 5.2
3 b = float(6.7)
4 print(a, b)
```

Program 3.4: Declare a variable as a real number

4.2 Enter real numbers from the keyboard

A small program, to get real numbers from keyboard input is as follows:

```
1 a = float(input("Enter real number a: "))
2 print(a)
```

Program 3.5: Get real numbers from keyboard

With this function, we must have the keyword **float**, not omit it like the variable declaration. The reason is that what is entered from the keyboard, is a string type. The float keyword in front will do the conversion from string to numeric. Therefore, you can enter the integer 6, it will still be a valid number to convert to real number.

4.3 Operations on real numbers

Before experimenting with integer operations, we will execute a small program, which is to enter two numbers n1 and n2 from the keyboard, and print the sum of the two numbers. The example program is as follows:

```
1 n1 = float(input("First number: "))
2 n2 = float(input("Second number: "))
3
4 print("Total of 2 numbers: ", n1+n2)
```

Program 3.6: Simple calculation program on 2 real numbers

Common operations on real numbers, which are similar to integers, are listed as follows:

- + : Addition
- - : Subtraction
- * : Multiplication
- / : Divide real numbers
- **: Exponential math

Now we can try to change the math in Program 3.7 to better understand the maths listed above.

5 Handle input errors

This will be the most important part of input data processing. When giving a message to enter an integer, the user can completely enter a string. As a result, the type conversion statement may misbehave and the program must stop. Although Python will ignore integer input errors, the input is real numbers and vice versa, because these 2 pieces of information are also numbers.

To handle the case where the user enters a string, or collectively, **invalid numeric value**, the simplest technique is to use a try except statement, like the following example program:

```
1 try:
2     a = int(input("Input integer: "))
3 except:
4     a = 0
5 print(a)
```

Program 3.7: Input error handler

In the above program, Python will try to convert the input string value to an integer and assign it to the variable a. In case this process fails (due to invalid data), the except part will be executed and a has the value 0. Otherwise, if the conversion succeeds, the except part will not be executed.

6 Review questions

1. In Python, it is necessary to declare a variable before assigning a value to it. Is the above statement true or false?
 - A. True
 - B. False
2. The way to assign an integer value of 100 to variable a is:
 - A. `a := 100`
 - B. `a <- 100`
 - C. `a = 100`
 - D. `a « 100`
3. In Python, a variable a is assigned a value of type int, and then can continue to assign a value of type float to variable a. Is the above statement true or false?
 - A. False
 - B. True

4. Given the following code:

```
1 employeeNumber = 4398
2 EmployeeNumber = 4398
3 employeeNumber = 4398
```

Which of the following statements is correct:

- A. The above commands all initialize the same variable
 - B. The above commands initialize different variables
5. What is the data type of the variable N in the following command after the user enters the value:

```
1 N = input("Input N") # Let's say user enters 5
2 print(N)
```

- A. str
 - B. int
 - C. float
 - D. No data type
6. Given the following code:

```
1 a = 4 / 2
```

What is the result and data type of variable a:

- A. `a = 2.0`, type float
- B. `a = 2.0`, type int
- C. `a = 2`, type float
- D. `a = 2`, type int

7. Given the following code:

```
1 a = 5
2 b = 5.0
3 c = a + b
```

What the result and data type of the variable c is:

- A. c = 10.0, type float
- B. c = 10, type int
- C. Error because two numbers of different types

Answer

1. B 2. C 3. B 4. B 5. A 6. A 7. A

CHAPTER 4



IF Conditional Statement

pythonTM
Package
Index

1 Introduction

In any programming language, there will be conditional statements. Thanks to this statement, the mapping of tasks to be executed when certain conditions occur, can be performed on the computer. When we say "If it rains, I will bring an umbrella", is an example of the behavior **bring an umbrella** that is only performed when there is **it is raining**. In programming languages **it rains** is considered a condition. When this condition is true, the computer will perform a function, such as **carrying an umbrella**. Also from this article, we officially start the concept of **processing** on computers. Math problems are getting more and more complex and require a combination of complex commands.

In the Python programming language, a simple condition is usually expressed through a comparison. And a comparison expression will return results with data type of Boolean, i.e. True or False. The comparison operators can be mentioned as follows:

- Compare equals: $a == b$
- Different: $a != b$
- Less than: $a < b$
- Less than or equal to: $a <= b$
- Greater than: $a > b$
- Greater than or equal to: $a >= b$

Here a and b are examples for 2 variables. However, we can also compare a variable and a constant, similar to other programming languages. In addition, logical operators such as **and** and **or** can also be used when considering the association condition of multiple logical propositions.

In fact, the conditional statement **if** has many different uses, through its many **variants**. In this tutorial, we distinguish three statements, including the simple **if** statement, the **if else** statement, and the **if elif else** statement. The usage and meaning of each statement in Python in turn are presented below.

2 Command if

This is the simplest statement, with the following conditional statement structure:

```
1 if <Condition>:  
2     <Statements>
```

In this structure, if the condition is true, then the statements are executed. Otherwise, if the condition is false, then the statements will not be executed. It is imperative that we have at least one statement following the **if** statement, otherwise the program will give an error.

Another extremely important note, after the **if** statement, the statements contained within it must be indented **a tab character**. When working on PyCharm, just type up to a colon

(:), and then press Enter, the program will automatically indent. After executing the statements inside **if**, press Delete to get out of the scope of the conditional statement.

In the Pascal language, the construction of a sub-scope for the conditional statement is usually placed between the keywords **begin end**, or like the C language, the statements will be placed between the two opening and closing brackets ({}). However, in Python, a tab character means that we have just opened up a new area. Therefore, the programmer must be very careful in managing the scope.

```
1 a = 100
2 b = 10
3 if b == a:
4     print("b equal a")
```

Program 4.1: conditional statement if

In the Program 4.1 example above, only when *a* and *b* have equal values will the program print the sentence **b equal a**. In addition, the program does not print any sentences.

3 Statement if else

In this statement, there will be a second group, which, if the conditional statement is not true, the program will execute the statements in this group. In other languages, this is the **If then ... more ...**. The structure of the full conditional is as follows:

```
1 if <Condition>:
2     <Statements>
3 else:
4     <Statements>
```

An example for using this statement is as follows:

```
1 a = 100
2 b = 10
3 if b == a:
4     print("b equal a")
5 else:
6     print("a is different from b")
```

Program 4.2: If else statements

With Program 4.2, one of the two messages will be printed to the screen. In the example above, the sentence "a is different from b" will be printed. In this case, the condition **b == a** is false, so the command in **else** will be executed.

4 If elif else statement

With this statement, multiple conditions will be considered **in turn** before the last part else will be executed, if all previous conditions are false. The structure of this command is as follows:

```

1 if <Condition_1>:
2     <Statements>
3 elif <Condition_2>:
4     <Statements>
5 elif <Condition_3>:
6     <Statements>
7 else:
8     <Statements>

```

One mistake that learners will easily encounter when using this structure, is that from the top down, when a condition is true, the statements in that condition will be executed and all the rest of the conditions will not be considered again. For example, the first and second conditions are true, then the program only executes the statements in group 1. In other words, only **first condition is true** is executed in the **if** statement.

An example for this command is as follows:

```

1 a=200
2 b=10
3 if a<b:
4     print("a less than b")
5 elif a==b:
6     print("a equal b")
7 else:
8     print("a greater than b")

```

Program 4.3: If...elif...else example

Note that the **else** part is optional, maybe yes or no.

5 Nested statements

Actually **if**, **if else** or even **if elif else** is just one statement. Therefore, it can appear inside a condition of the **if** statement, creating nested conditional statements.

This presentation just wants to re-emphasize to the reader, that using this nested structure can be confusing, if do not understand the principle of **if** command. An example program using nested conditional statements would be as follows:

```

1 a = 2
2 b = 3
3 if a > 1:
4     if b > 4:
5         a = a + 1
6         b = b + 1
7 else:
8     a = a + 2
9     b = b + 2
10 print(a, b, sep=" - ")

```

Program 4.4: Nested if statement example

Without mastering the principle of the conditional statement, many people will mistakenly believe that the final result of *a* and *b* is 4 and 5 respectively. However, when the

condition $a > 1$ is true, then its **else** part will no longer be executed. Inside the first **if** section, the condition $b > 4$ is false and no further program segments are executed. In the end, the values of a and b do not change, remaining 2 and 3.

6 Exercise

1. Enter a number from the keyboard, print the result is even or odd number.

```
1 num = float(input("Enter a number "))
2 if num % 2 == 0:
3     print("Even number")
4 else:
5     print("Odd number")
```

Program 4.5: Suggested Answers for Exercise 1

2. Enter a number from the keyboard, print whether the result is positive, negative or zero (0).

```
1 num = float(input("Enter a number "))
2 if num >= 0:
3     if num == 0:
4         print("Zero")
5     else:
6         print("Positive number")
7 else:
8     print("Negative number")
```

Program 4.6: Suggested Answers for Exercise 2

3. Write a program to simulate solving a first-order equation with 1 unknown number: Allow the user to enter the numbers a and b , then print the result of the equation.

```
1 print("First-order equation: ax + b = 0")
2 a = int(input("a = "))
3 b = int(input("b = "))
4 if (a == 0):
5     print("Equation has no solution")
6 else:
7     x = -b/a
8     print("Solution of equation: x = ", x)
```

Program 4.7: Suggested Answers for Exercise 3

4. Write an equation to simulate solving a quadratic equation 1 unknown: Allow the user to input a , b and c , then print the result of the program. In this tutorial, we need to add the `sqrt` library to calculate the square root using the command **from math import sqrt**.

```
1 from math import sqrt
2
3 print("Quadratic equation : ax^2 + bx + c = 0")
```

```

4 a = float(input("a = "))
5 b = float(input("b = "))
6 c = float(input("c = "))
7
8 if a == 0:
9     if b == 0:
10         if c == 0:
11             print("Equation has many solutions!")
12         else:
13             print("Equation has no solution!")
14     else:
15         if c == 0:
16             print("Equation has 1 solution x = 0")
17         else:
18             x = -c / b
19             print("Equation has 1 solution x = ", x)
20 else:
21     delta = b ** 2 - 4 * a * c
22     if delta < 0:
23         print("Equation has no solution!")
24     elif delta == 0:
25         x = -b / (2 * a)
26         print("Equation has 1 solution x = ", x)
27     else:
28         print("Equation has 2 distinct solutions!")
29         x1 = float((-b - sqrt(delta)) / (2 * a))
30         x2 = float((-b + sqrt(delta)) / (2 * a))
31         print("x1 = ", x1)
32         print("x2 = ", x2)

```

Program 4.8: Suggested Answers for Exercise 4

7 Review questions

1. Which if statement is syntactically correct in Python?

- A. `if a>=2:`
- B. `if (a >= 2)`
- C. `if a >=2`
- D. All correct

2. What is the output of the following program?

```
1 x = True
2 y = False
3 z = False
4 if not x or y:
5     print (1)
6 elif not x or not y and z:
7     print (2)
8 elif not x or y or not y and x:
9     print (3)
10 else:
11     print (4)
```

- A. 1
- B. 2
- C. 3
- D. 4

3. Which of the following commands will fail?

A.

```
1 if (1, 2): print('a')
```

B.

```
1 if (1, 2):
2     print('a')
```

C.

```
1     if (1, 2):
2         print('a')
```

D.

```
1 if (1, 2):
2
3     print('a')
```

4. To end a block of statements in the body of an if statement, we use:

- A. Semicolon
- B. Close brackets
- C. `end`
- D. A statement is indented less than the previous statement

5. Does the following code have an error?

```
1 d = {'a': 0, 'b': 1, 'c': 0}
2 if d['a'] > 0:
3     print('ok')
4 elif d['b'] > 0:
5     print('ok')
6 elif d['c'] > 0:
7     print('ok')
8 elif d['d'] > 0:
9     print('ok')
10 else:
11     print('not ok')
```

- A. No error
- B. There was an error

6. Which of the following statements will execute successfully, assuming x and y have been initialized before?

A.

```
1 if x < y: if x > 10: print('foo')
```

B.

```
1 if x < y: print('foo') else: print('bar')
```

C.

```
1 if x < y: print('foo'); print('a'); print('b')
```

D.

```
1 if x < y: print('foo')
2 elif y < x: print('bar')
3 else: print('baz')
```

7. What is the following command used for:

```
1 a = int(input("a = "))
2 if a % 2 == 0:
3     print(a)
4 else:
5     print(0)
```

- A. Print the value of a
- B. If the number is even, print the value you just entered, otherwise print 0
- C. Prints the value of a and prints 0
- D. If it is an odd number, print the value you just entered, otherwise print 0

Answer

1. A 2. C 3. C 4. D 5. A 6. C 6. D 7. B

CHAPTER 5



One-dimensional array - FOR loop structure

pythonTM
Package
Index

1 Introduction

Arrays are a fundamental data structure of all programming languages. Unlike a single variable, which is only used to store individual values, an array is a collection of many elements of the same data type, such as an integer array or a string array.

However, on Python, the pure array concept like other programming languages like Pascal, C or C++ is quite complicated and not very beginner friendly. Therefore, in this tutorial, we will use the concept of list instead of array. At first glance, we will see that it is no different from arrays in other programming languages. We still use the term "array" in this tutorial, for the convenience of the reader.

2 Declare and traverse a one-dimensional array

Arrays are declared in Python with the character string [], in the middle are the elements of the array, separated by commas (,), like the example below:

```
1 a = [1, 4, 5, 7]
2 print("First number", a[0])
3 print("Second number", a[1])
```

Program 5.1: Declare and traverse an array

As the example in Program 5.1, array **a** has all **4 elements**. Accessing each element in **a** is done via the operation **a[0]** or **a[1]**. This means, the first element will have the index of 0, and the last element will have the index of 3. For some reason, if we access the element **a[4]**, the program will report "**list index out of range**" error, i.e. the access exceeds the limit of the array.

The flexibility in the Python language allows we to access elements with a negative index, for example **a[-1]** for the last element and back to the first element, which is **a[-4]**. The same limit overflow error as above also appears when you access **a[-5]**. However, we really do not encourage you to follow this rule.

3 For loop structure

When working with arrays, the most commonly used task is called **array traversal**. We need a loop to process through all the elements of the array. In this tutorial, we will use the FOR.

In this structure, we need a variable, named **i**, to hold the current element index of the array. This variable will automatically increment by 1 after each iteration. In Python, the simplest way to implement this function is to use the statement **range(lower bound, upper bound)**, presented as shown below:

```
1 a = [1, 4, 5, 7]
2 N = len(a)
3 print("Array size", N)
4 for i in range(0, N):
```



```
5 print(i)
```

Program 5.2: Python range statement

In the example in Program 5.2, we get the size of the array through the operator **len** and then assign the result to the variable **N**. The variable **N** will then have the value 4, because the array **a** there are 4 elements. However, in the **for** loop, the values are printed from 0 to 3 only. That's why, the range statement will go into range suitable for array traversal in Python, because the array's index is numbered from 0 to **N** - 1, where **N** is the size of the array.

We will modify the program above to print out each value of the array, like this:

```
1 a = [1, 4, 5, 7]
2 N = len(a)
3 print("Array size", N)
4 for i in range(0, N):
5     print("No.", i + 1, a[i])
```

Program 5.3: Print array values

Again, in natural language we usually position the first number as the first number (the number 1), while in the Python programming language the first position is the 0. Hence, in the statement print to the screen, to make the information more user-friendly, we will add the variable **i** by 1 unit.

The FOR iteration structure presented in this tutorial is also known as the finite iterative structure. This means that before the loop starts, the values of the variable **i** have been set to the value of **N**. From there, the loop is executed through the **i** variable and is completely independent of **N**. Therefore, if we change the **N** value inside the loop, the result remains unchanged.

```
1 a = [1, 4, 5, 7]
2 N = len(a)
3 print("Array size", N)
4 for i in range(0, N):
5     print("No.", i + 1, a[i])
6     N = N - 1
```

Program 5.4: Finite loop for regardless of value **N**

The program 5.4 is an example to illustrate the fixed number of iterations of the FOR structure. At the initial setup time, **N** = 4 and the program will be looped 4 times, regardless of whether we change the value of **N** inside the loop. Therefore, this structure is often used for cases with a fixed number of loops.

At this point, we can pause to switch to exercises (from 1 to 5) to practice with arrays on Python. The next section of this tutorial will show how to enter an array from the keyboard, instead of assigning a fixed array like the examples above.

4 Enter array from keyboard

According to the ideas presented in this tutorial, we give priority to the output of the output first, and the input processing to be done later. The reason is that the processing of

input data is often more complex. For example, in this request, we expect the user to enter a number. However, for some reason, the user enters a string of characters. Handling these errors is also not simple, we will ask the user to re-enter, or we will default to the wrong input value 0.

In this tutorial we will focus only on a simple application: The program asks for the size of the array, a loop will ask the user to enter each element for the array.

With the above requirement, we will initialize the array `a = []`, which has no elements yet. After asking the user to enter the number `N`, the for loop will ask to enter each element. The suggested program is as follows:

```
1 a = []
2 N = int(input("Enter array size: "))
3 for i in range(0,N):
4     temp = int(input("Enter element no. " + str(i + 1)))
5     a.append(temp)
6 print(a)
```

Program 5.5: Enter array values from the keyboard

To make the input statement more user-friendly, we will add information about the sequence number of the array when asking the user for input. Since `i + 1` is a numeric type, it must be converted to a string type before concatenating with another string. Finally, the **append** operator is to add the entered element to the `a` array. At the end of the program, we will use the **print(a)** command to check the entered values of the array.

It should be noted that, for programming languages, there is no operator to add an element to an array like Program 5.5. Here, essentially, we are using a list to replace the function of an array, so this operator is available.

5 Exercise

1. Write a program to calculate the sum of the elements in an array. Arrays are entered at the beginning of the program.

```
1 a = [1,2,3,4,5]
2 N = len(a)
3 Sum = 0
4 for i in range(0,N):
5     Sum = Sum + a[i]
6 print("Sum of array values:", Sum)
```

Program 5.6: Suggested Answers for Exercise 1

2. Write a program to count the number of even numbers in an array. Arrays are entered at the beginning of the program.

```
1 a = [1,2,3,4,5]
2 N = len(a)
3 even = 0
4 for i in range(0,N):
5     if a[i] % 2 == 0:
```

```

6         even = even + 1
7 print("Count of even numbers:", even)

```

Program 5.7: Suggested Answers for Exercise 2

3. Write a program to count the number of odd numbers in an array. Arrays are entered at the beginning of the program.

```

1 a = [1,2,3,4,5]
2 N = len(a)
3 odd = 0
4 for i in range(0,N):
5     if a[i] % 2 != 0:
6         odd = odd + 1
7 print("Count of odd numbers:", odd)

```

Program 5.8: Suggested Answers for Exercise 3

4. Write a program to find the largest number in an array. Arrays are entered at the beginning of the program.

```

1 a = [1,2,3,4,5]
2 N = len(a)
3 max = a[0]
4 for i in range(1,N):
5     if max < a[i]:
6         max = a[i]
7 print("Max value is:", max)

```

Program 5.9: Suggested Answers for Exercise 4

5. Write a program to find the smallest number in an array. Arrays are entered at the beginning of the program.

```

1 a = [1,2,3,4,5]
2 N = len(a)
3 min = a[0]
4 for i in range(1,N):
5     if min > a[i]:
6         min = a[i]
7 print("Min value is:", min)

```

Program 5.10: Suggested Answers for Exercise 5

6. Perform all the above requirements again, with the array entered from the keyboard.

```

1 a = []
2 N = int(input("Enter array size: "))
3 for i in range(0, N):
4     temp = int(input("Enter element no. " + str(i+1)))
5     a.append(temp)
6
7 #Calculate Sum, find even, odd numbers
8 sum = even = odd = 0
9 for i in range(0,N):
10     sum = sum + a[i]
11     if a[i] % 2 == 0:

```

```

12         even = even + 1
13     else:
14         odd = odd + 1
15
16 #Find min, max
17 max1 = min1 = a[0]
18 for i in range(1,N):
19     if a[i] < min1:
20         min1 = a[i]
21     if a[i] > max1:
22         max1 = a[i]
23
24 print(sum, even, odd, max1, min1)

```

Program 5.11: Suggested Answers for Exercise 6

6 Review questions

1. What is the output of the following command?

```
1 numbers = [1, 2, 3, 4]
2 numbers.append([5,6,7,8])
3 print(len(numbers))
```

- A. 4
- B. 5
- C. 8
- D. Error

2. What is the output of the following command?

```
1 list1 = [1, 2, 3, 4]
2 list2 = [5, 6, 7, 8]
3 print(len(list1 + list2))
```

- A. 2
- B. 4
- C. 8
- D. Error

3. What is the output of the following command?

```
1 list1 = [1, 2, 3, 4]
2 print(list1[4])
```

- A. 3
- B. 4
- C. 1
- D. Error

4. What is the output of the following command?

```
1 x = sum(range(5))
2 print(x)
```

- A. 4
- B. 5
- C. 10
- D. 15

5. Which of the following statements is used to print the elements of the array list to the screen:

```
1 list1 = [1,2,3,4]
```

A.

```
1 for i in list1:
2     print(i)
```

B.

```
1 for i in range(len(list1))
2     print(a[i])
```

- C. Both of the above commands are correct
D. Both of the above commands are wrong

6. What is the result of the following code, given that the break statement is used to exit the loop containing it:

```
1 for i in range(10):
2     if i == 4:
3         break
4     else:
5         print(i)
6 else:
7     print("Python")
```

- A. 0 1 2 3 4
B. 0 1 2 3 Python
C. 0 1 2 3
D. 0 1 2 3 4 Python

7. What is the value of a after the loop ends:

```
1 a = 0
2 for i in range(10, 14):
3     a = a + 3
```

- A. 9
B. 12
C. 15
D. 18

Answer

1. C 2. C 3. D 4. C 5. C 6. C 7. C

CHAPTER 6



Multidimensional array - nested FOR

pythonTM
Package
Index

1 Introduction

As introduced in the previous post, a one-dimensional array contains a collection of many elements with the same data type. In some cases, we will have each element as an array. This creates a new concept called multidimensional arrays. One of the best examples of multidimensional arrays is matrices, a data structure commonly found for image processing or mathematical processing.

Just like in the previous tutorial, we can use lists to implement multidimensional arrays: Each element of a list is another list. When using lists to implement instead of arrays, the operation will be very similar to the traditional array structure in other languages such as Pascal or C. Accessing an element in an array will require 2 indexes for 2-dimensional array, similar to the concept of rows and columns in other languages.

However, before going into the details of the multidimensional array, we will create a new python file in the project. This first project, we can save it as a sample program library, and use it for future projects.

2 Add a Python file

By default, when we first create a project, we only have one python file, which is **main.py**. When you click the **Run** button on the toolbar (or press the **Shift F10** hotkey), Pycharm will default to running the program in the main.py file. In this tutorial, we will add a new file, name it **mang_2_chieu.py**, and reconfigure PyCharm so that it can run programs in this file. The step-by-step instructions are detailed below.

Step 1: From the File menu, select **New...** (avoid mistakenly selecting New Project...). An interface appears and we continue to choose Python File, as illustrated in Figure 6.1.

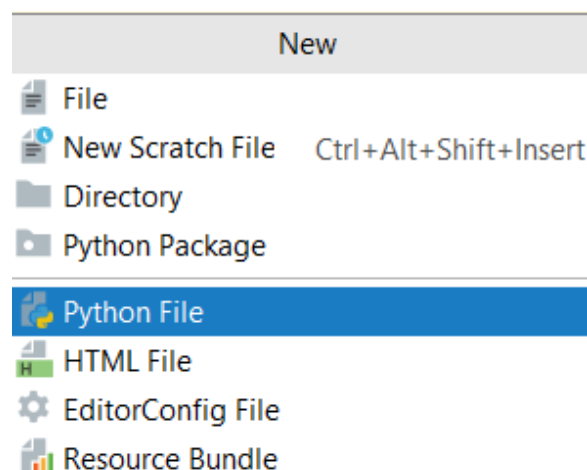


Figure 6.1: Add a new python file

Step 2: Name the file **give_2_chieu** and then press **Enter**, as shown in Figure 6.2. The **.py** extension for the file is automatically added.

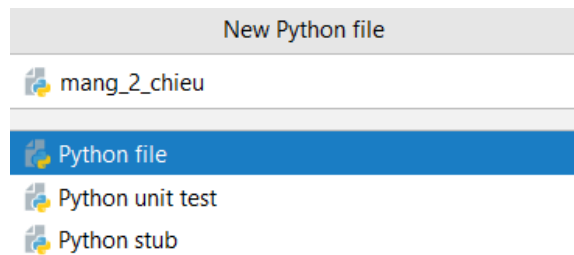


Figure 6.2: Name for python file

After successfully added, we will see the newly added file appear in the Project window. We open this file by double clicking on it, and print to the screen a simple sentence, as shown in Figure 6.3.

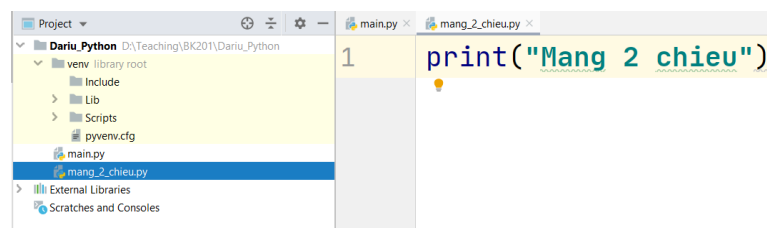


Figure 6.3: Newly added file on PyCharm

Step 3: From the Project window, right-click, and select **Run with_2_chieu**, as shown in Figure 6.4.

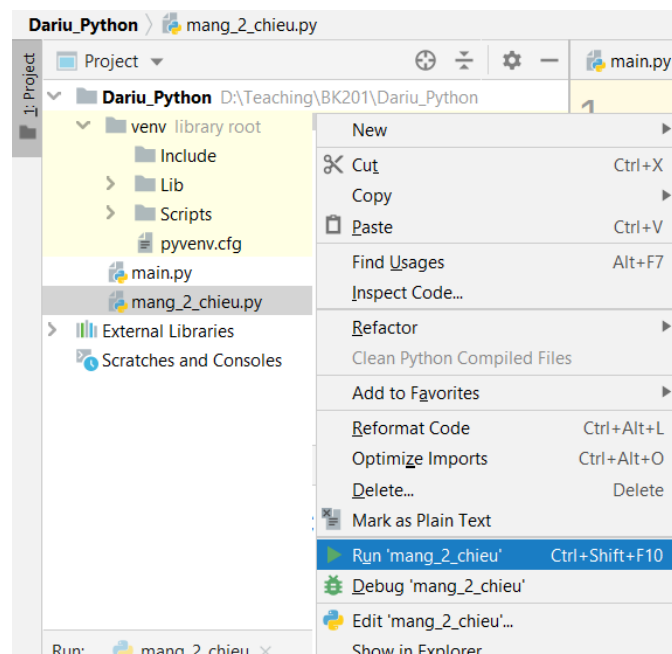


Figure 6.4: Run program of newly added file

Now if we look at the option before the Run button on the toolbar, we will see that PyCharm automatically adds another option (see Figure 6.5). We have 2 options, **main** and **carry_2_chieu**. We can change the execution of the python file by selecting here. Also

because in our project there are many files, let's print something before executing, as a way to manage the program more easily.

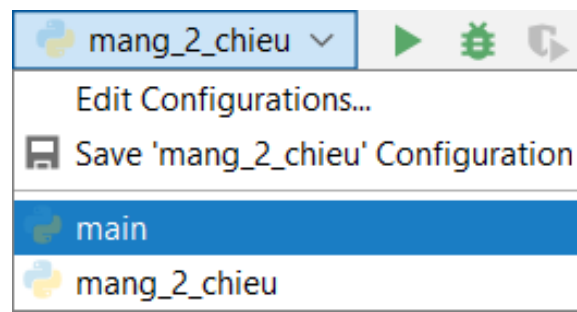


Figure 6.5: Option to change the executable on the toolbar

3 Declare and traverse multidimensional arrays

Each dimension of the array will be accessed through an index. In the program below, the array **a** is initialized to include 2 dimensions: The first dimension has 2 elements, the second dimension, each dimension has 3 elements.

```
1 a = [[1, 2, 3], [4, 5, 6]]
2 print(a[0][0])
3 print(a[0][1])
4 print(a[1][1])
```

Program 6.1: Example for 2D array

Like a one-dimensional array, **the first element in each dimension is indexed 0**. This is very important when working with arrays in Python. Through the example above, we can also map a 2-dimensional array as a matrix: the first dimension is the row and the second dimension is the column. We can also state that the matrix **a** above has 2 rows and 3 columns.

4 Traverse a multidimensional array

Through the declaration of a multi-dimensional array, we can realize that it is two nested one-dimensional array structures (the specialized word is nested list). Therefore, the most common way to traverse a multidimensional array is to use two nested for loops. The following program, will print information about the row and column position along with the value of the element there, as the example in Program 6.2.

```
1 a = [[1, 2, 3], [4, 5, 6]]
2 ROW = len(a)
3 COL = len(a[0])
4 for i in range(0, ROW):
5     for j in range(0, COL):
6         print("Row", i + 1, "Column", j + 1, ":", a[i][j])
```

Program 6.2: Traverse 2D array

In fact, the above array traversal is only correct assuming that all rows have the same number of columns. Therefore, the above program only needs to calculate the number of columns of the first row (the statement **COL = len(a[0])**) This is also the principle of arrays in programming languages that have structured like Pascal or C. However, in Python, we are allowed to **change the number of columns of each row** in a multidimensional matrix. The program 6.2 needs to change the loop inside to be able to function properly, like the example in Program 6.3.

```
1 a = [[1, 2, 3], [4, 5, 6], [7, 8]]
2 ROW = len(a)
3 for i in range(0, ROW):
4     COL = len(a[i])
5     for j in range(0, COL):
6         print("Row", i + 1, "Column", j + 1, ":", a[i][j])
```

Program 6.3: Traverse 2D array

In the example in Program 6.3, the third row of the array **a** has only 2 columns. Therefore, before entering the column loop, we recalculate the number of columns with the statement **COL = len(a[i])**.

5 Enter a multidimensional array from the keyboard

In fact, the need to enter a multidimensional array from the keyboard is not much because the number of elements to enter is quite a lot. For the function that reads data from a multi-dimensional array, which is mainly a 2-dimensional array, we will read the data up from an existing file. However, in this tutorial, we also suggest to the learners the most basic processing steps when performing reading data from the keyboard for a 2-dimensional array. The main idea here is to input each row (1 dimensional array), and then add each row to the 2 dimensional array. The suggested program is as follows:

```
1 a = []
2 ROW = int(input("Number of rows "))
3 COL = int(input("Number of columns "))
4 for i in range(0, ROW):
5     temp_row = []
6     for j in range(0, COL):
7         temp = int(input("Element "+str(i)+"-"+str(j)))
8         temp_row.append(temp)
9     a.append(temp_row)
10 print(a)
```

Program 6.4: Enter 2D array from keyboard

In Program 6.4, variable **a** is used to store information of 2-dimensional array, variable **temp_row** is used to store each row of 2-dimensional array. The loop with the variable **j** is actually a loop to input a 1-dimensional array from the keyboard in the previous post. Finally, the variable **a** is printed to check the input. This program is allowing input of integers only. The case of entering real numbers will be considered as an advanced exercise. The 6.4 program is a great practice for scope discrimination in Python. We have 2 nested for loops and reassigning values to variables needs to be done in the right scope to get the

desired result.

In this program, we implicitly specify that the rows have the same number of columns. If we want to enter a matrix with an unknown number, for example, enter until we reach zero, we will have to use another technique, which will be covered in the following tutorial.

6 Exercise

1. Write a program to calculate the sum of the elements in a 2-dimensional array. Arrays are permanently entered at the beginning of the program.

```
1 a = [[1, 2, 3], [4, 5, 6]]
2 ROW = len(a)
3 COL = len(a[0])
4 sum = 0
5 for i in range(0, ROW):
6     for j in range(0, COL):
7         sum = sum + a[i][j]
8 print("Sum of array values: ", sum)
```

Program 6.5: Suggested Answers for Exercise 1

2. Write a program to count the number of even numbers in a 2-dimensional array. Arrays are permanently entered at the beginning of the program.

```
1 a = [[1, 2, 3], [4, 5, 6]]
2 ROW = len(a)
3 COL = len(a[0])
4 even = 0
5 for i in range(0, ROW):
6     for j in range(0, COL):
7         if a[i][j] % 2 == 0:
8             even = even + 1
9 print("Count of even numbers: ", even)
```

Program 6.6: Suggested Answers for Exercise 2

3. Write a program to count the number of odd numbers in a 2-dimensional array. Arrays are permanently entered at the beginning of the program.

```
1 a = [[1, 2, 3], [4, 5, 6]]
2 ROW = len(a)
3 COL = len(a[0])
4 odd = 0
5 for i in range(0, ROW):
6     for j in range(0, COL):
7         if a[i][j] % 2 != 0:
8             odd = odd + 1
9 print("Count of odd numbers: ", odd)
```

Program 6.7: Suggested Answers for Exercise 3

4. Write a program to find the largest number in a 2-dimensional array. Arrays are permanently entered at the beginning of the program.

```

1 a = [[1, 2, 3], [4, 5, 6]]
2 ROW = len(a)
3 COL = len(a[0])
4 max1 = a[0][0]
5 for i in range(0, ROW):
6     for j in range(0, COL):
7         if a[i][j] > max1:
8             max1 = a[i][j]
9 print("Max value is: ", max1)

```

Program 6.8: Suggested Answers for Exercise 4

5. Write a program to find the smallest number in a 2-dimensional array. Arrays are permanently entered at the beginning of the program.

```

1 a = [[1, 2, 3], [4, 5, 6]]
2 ROW = len(a)
3 COL = len(a[0])
4 min1 = a[0][0]
5 for i in range(0, ROW):
6     for j in range(0, COL):
7         if a[i][j] < min1:
8             min1 = a[i][j]
9 print("Min value is: ", min1)

```

Program 6.9: Suggested Answers for Exercise 5

6. Perform all the above requirements again, with the array entered from the keyboard.

```

1 a=[]
2 ROW = int(input("Number of rows "))
3 COL = int(input("Number of columns "))
4 for i in range (0, ROW):
5     temp_row = []
6     for j in range(0, COL):
7         temp = int(input("Element "+str(i)+ "-" + str(j) +
8 ")))
9         temp_row.append(temp)
10    a.append(temp_row)
11 for i in range(0, ROW):
12     for j in range(0, COL):
13         print(a[i][j])
14
15 # Calculate sum, find even, odd
16 sum = even = odd = 0
17 for i in range(0, ROW):
18     for j in range(0, COL):
19         sum = sum + a[i][j]
20         if a[i][j] % 2 == 0:
21             even = even + 1
22         else:
23             odd = odd + 1

```

```
24
25 # Find min, max
26 max1 = min1 = a[0][0]
27 for i in range(0, ROW):
28     for j in range(0, COL):
29         if a[i][j] < min1:
30             min1 = a[i][j]
31         if a[i][j] > max1:
32             max1 = a[i][j]
33
34 print(sum, even, odd, max1, min1)
```

Program 6.10: Suggested Answers for Exercise 6

7 Review questions

1. Which of the following is used to declare a two-dimensional array with two rows and three columns:
A. `a = [[1,2,3], [4,5,6]]`
B. `a = [1,2,3,4,5,6]`
C. `a = [1,2,3];[4,5,6]`
2. Which of the following is used to assign the value of the first element in the second row of a two-dimensional array `a` to the variable `x`:
A. `x = a[0,1]`
B. `x = a[2][1]`
C. `x = a[1,0]`
D. `x = a[1][0]`
3. Given the following code:

```
1 for i in range (0,2):  
2     for j in range (0,2):  
3         print(1)
```

How many times the print command is executed:

- A. 2
 - B. 3
 - C. 4
 - D. 7
4. Given the following code:

```
1 a = [[1,2,3], [4,5,6], [7,8,9]]  
2 for i in range (0,2):  
3     for j in range (0,2):  
4         print(a[i][j])
```

What does the above command do:

- A. Prints all values in a two-dimensional array `a`
- B. Print all the values of the top 2 in the two-dimensional array `a`
- C. Prints the first 2 of the first 2 rows in a two-dimensional array `a`

5. Given the following code:

```
1 x = 0  
2 a = [[1,2,3], [4,5,6]]  
3 for i in range (0,2):  
4     for j in range (0,3):  
5         if a[i][j] % 2 == 0:  
6             x = x + a[i][j]
```

Value of the variable `x` is:

- A. 9
 - B. 12
 - C. 21
6. Given the following code:

```

1 x = 0
2 for i in range (0,1):
3     for j in range (0,2):
4         for j in k (0,3):
5             x = x + 1

```

Value of the variable x is:

- A. 6
- B. 8
- C. 9
- D. 12

7. Given the following code:

```

1 x = 0
2 a = [[1,2,3],[4,5,6]]
3 for i in range (0,2):
4     for j in range (0,3):
5         if a[i][j] % 2 == 0:
6             break;
7         else:
8             x = x + a[i][j]

```

What is the value of the variable x, knowing that the break statement is used to exit the loop containing it:

- A. 0
- B. 1
- C. 4
- D. 9

Answer

1. A 2. D 3. C 4. C 5. B 6. A 7. B

CHAPTER 7



While loop structure

pythonTM
Package
Index

1 Introduction

What is loop? Loops allow us to repeat the execution of code until a certain condition is satisfied. Loops are used quite commonly in programming. Unlike other programming languages with different loops like for, while, do...while..., Python only supports for loop and while loop.

In this tutorial we will learn about the while loop in Python, this is a loop that is used quite a lot when you do applications in practice, not only in Python but also in other languages.

2 Loop syntax while

Python is a simple language so its syntax is also simple. Following is the general syntax of the while loop.

```
1 while expression:
2     statement(s)
```

In there:

- **statement(s):** is a single command or a set of instructions consisting of many single commands. If there is only one command we may not need spaces, but it is recommended that we use spaces to make the program clearer and easier to maintain.
- **expression:** can be a variable or an expression, but its value must be TRUE or FALSE.

For example:

```
1 count = 0
2 while (count <= 6):
3     print("Count:", count)
4     count = count + 1
5
6 print("Good bye!")
```

Print results:

```
1 Count: 0
2 Count: 1
3 Count: 2
4 Count: 3
5 Count: 4
6 Count: 5
7 Count: 6
8 Good bye!
```

Line Good bye! is not repeated because it is outside the loop, and the count will be repeated 7 times the variable count has an initial value of 0 (count = 0), after each loop it increases by 1 unit (count = count + 1) and the stop condition is count less than or equal to 6 (count <= 6).

2.1 break and continue statements within while

2.1.1 Command break

Used to end the loop. As long as it is in the block of any loop, the loop will terminate when running this statement.

In case loop a contains loop b. In loop b running break statement, only loop b ends, while loop a does not.

2.1.2 Command continue

This statement is used to continue the loop. Suppose a loop has the following structure:

```
1 while expression:
2     #while-block-1
3     continue
4     #while-block-2
```

When the while-block-1 is done, the continue statement will continue the loop, ignoring the statements below continue and so it skips the while-block-2.

2.2 Use while on a line

If the block of code has only one line of code, we can write your while loop on just one line:

```
1 while True: print("hello")
```

Above is an infinite loop, with the condition always being True, so the loop will run until you end the program.

3 Loop syntax while-else

There is a novelty to the while loop in Python that you can incorporate the ELSE keyword to handle the loop that is not executed when the loop condition is false.

```
1 while expression:
2     statement(s)
3 else:
4     statement(s)
```

For example:

```
1 count = 0
2 while count < 5:
3     print(count, " is less than 5")
4     count = count + 1
5 else:
6     print(count, " is not less than 5")
```

Running this example results in the following:

```
1 0 is less than 5
2 1 is less than 5
3 2 is less than 5
4 3 is less than 5
5 4 is less than 5
6 5 is not less than 5
```

The 6th iteration will not happen so the code in the else statement will be executed.

4 Exercise

1. Write a program to calculate the sum of the elements in a 2-dimensional array. Arrays are permanently entered at the beginning of the program.

```
1 a = [1,2,3,4,5]
2 N = len(a)
3 Sum = 0
4 i = 0
5 while (i < N):
6     Sum = Sum + a[i]
7     i = i + 1;
8 print("Sum of array values:", Sum)
```

Program 7.1: Suggested Answers for Exercise 1

2. Write a program to count the number of even numbers in an array. Arrays are entered at the beginning of the program.

```
1 a = [1,2,3,4,5]
2 N = len(a)
3 even = 0
4 i = 0
5 while (i < N):
6     if a[i] % 2 == 0:
7         even = even + 1
8     i = i + 1
9 print("Count of even numbers:", even)
```

Program 7.2: Suggested Answers for Exercise 2

3. Write a program to count the number of odd numbers in a 2-dimensional array. Arrays are permanently entered at the beginning of the program.

```
1 a = [1,2,3,4,5]
2 N = len(a)
3 odd = 0
4 i = 0
5 while (i < N):
6     if a[i] % 2 != 0:
7         odd = odd + 1
8     i = i + 1
9 print("Count of odd numbers:", odd)
```

Program 7.3: Suggested Answers for Exercise 3

4. Write a program to find the largest number in a 2-dimensional array. Arrays are permanently entered at the beginning of the program.

```
1 a = [1,2,3,4,5]
2 N = len(a)
3 max = a[0]
4 i = 0
5 while (i < N):
6     if max < a[i]:
7         max = a[i]
8     i = i + 1
9 print("Max value is:", max)
```

Program 7.4: Suggested Answers for Exercise 4

5. Write a program to find the smallest number in a 2-dimensional array. Arrays are permanently entered at the beginning of the program.

```
1 a = [1,2,3,4,5]
2 N = len(a)
3 min = a[0]
4 i = 0
5 while (i < N):
6     if min > a[i]:
7         min = a[i]
8     i = i + 1
9 print("Min value is:", min)
```

Program 7.5: Suggested Answers for Exercise 5

6. Perform all the above requirements again, with the array entered from the keyboard.

```
1 a = []
2 N = int(input("Enter array size: "))
3 i = 0
4 while (i < N):
5     temp = int(input("Element " + str(i+1)))
6     a.append(temp)
7     i = i + 1
8
9 #Calculate sum, find even, odd
10 sum = even = odd = 0
11 i = 0
12 while (i < N):
13     sum = sum + a[i]
14     if a[i] % 2 == 0:
15         even = even + 1
16     else:
17         odd = odd + 1
18     i = i + 1
19 #Find min, max
20 max1 = min1 = a[0]
21 i = 0
22 while (i < N):
```

```

23     if a[i] < min1:
24         min1 = a[i]
25     if a[i] > max1:
26         max1 = a[i]
27     i = i + 1
28 print(sum, even, odd, max1, min1)

```

Program 7.6: Suggested Answers for Exercise 6

5 Review questions

1. What is the value of x?

```

1 x = 0
2 while (x > 10):
3     x+=2
4 print(x)

```

- A. 0
- B. 10
- C. 12
- D. Don't know the value of x because of the endless loop

2. What is the value of x?

```

1 x = 0
2 while (x < 10):
3     x+=2
4 print(x)

```

- A. 8
- B. 100
- C. 101
- D. Don't know the value of x because of the endless loop

3. What is the value of x?

```

1 x = 0
2 while (x < 50):
3     x-=2
4 print(x)

```

- A. 49
- B. 50
- C. 51
- D. Don't know the value of x because of the endless loop

4. Which of the while statements below will create an endless loop:

A.

```

1 while True:
2     . . .

```

B.

```
1 a = [1,2,3]
2 while a:
3     ...
```

C.

```
1 while "0":
2     ...
```

D. Depends on the code inside the while statement.

5. How many times does the following command print the character "a" and the character "b" ?

```
1 x = 0
2 while x < 5:
3     print("a")
4     x = x + 2
5 else:
6     print("b")
```

- A. 1 time "a" and 1 time "b"
- B. 2 times "a" and 1 time "b"
- C. 3 times "a" and 2 times "b"
- D. 3 times "a" and 2 times "b"

6. What is the result of s?

```
1 s = ""
2 n = 5
3 while n > 0:
4     n -= 1
5     if (n % 2) == 0:
6         continue
7     a = ['foo', 'bar', 'baz']
8     while a:
9         s += str(n) + a.pop(0)
10        if len(a) < 2:
11            break
```

- A. 3bar3bar1for1for
- B. 3foo3bar1foolbar
- C. 3foo3bar1bar1for
- D. All wrong

Answer

1. A 2. A 3. D 4. D 5. B 6. B

CHAPTER 8



File operations

pythonTM
Package
Index

1 Introduction

In some cases, entering data from the keyboard is not feasible and takes a lot of time, such as reading data from a multidimensional array in the previous tutorial. Or as another example, we need to save the result of program execution for review. In such cases, we will read the content from the file or save the execution result to file.

The process of working with files usually has 3 basic steps as follows: Open the file, read or write data, and finally close the file. If the file forgets to open, obviously we will not be able to do anything. Conversely, if the file forgets to close, the next time it is opened, the program may report an error, depending on the support level of the editing program. However, to be on the safe side, always remember to close the file at the end of the program.

2 Write data to File

We will start with the instructions for writing data to a file first. This can be seen as an output process. And usually, this process will be simpler than reading data from the file. An example program for opening a file named **test.txt**, writes some content and closes the file as follows:

```
1 file = open("test.txt", "w")
2 file.write("Test write content file 1")
3 file.write("Test write content file 2")
4 file.close()
```

Program 8.1: The program opens file - writes contents and closes file

In Program 8.1, **file** is a variable, referring to the file to be worked on. The file variable will take effect after the first statement, which is **open**. This open statement has two important parameters:

- File name: Name the file to save data. This file will be placed in the same directory as the working Python file. In case we want to save at a certain path in memory, we can specify the absolute path, for example **"D:/test.txt"**. Please note the separator character in the path, which is a right-to-left character, it is the opposite of the normal path character.
- File opening modes: There are many file opening modes in Python, here we only present the most commonly used modes, as shown below:
 - "r": Open file for reading only.
 - "w": Open file for writing only. If the file does not exist, create a new file. If the file already exists, delete the old file content and write the new content.
 - "a": Only open the file for further writing. If the file does not exist, create a new file. If the file already exists, write more content to the file.

In the example in Program 8.1, we open the file in **"w"** mode, i.e. open the file for writing. Initially, this file **test.txt** does not exist, so the system will automatically create this file and save it in the same directory as the python file we are working on. Each time the program is run, the old content of this file will be deleted and the new content will be recorded. In

case we want a newline, the character `"\n"` can be added, like the following example:

```
1 file = open("test.txt", "w")
2 file.write("Line 1\n")
3 file.write("Line 2" + "\n")
4 file.write("Line 3")
5 file.close()
```

Program 8.2: The program opens file - writes contents and closes file

The output of the file **test.txt** will look like this:

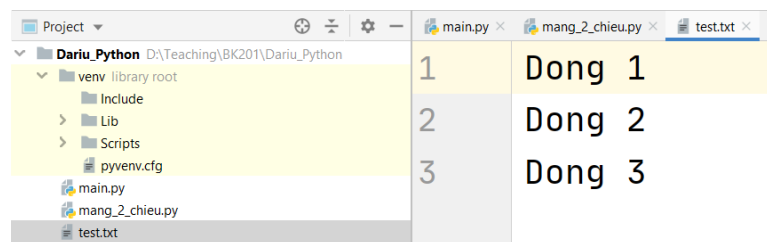


Figure 8.1: Write contents to file

3 Read data from File

To read data from the file, we will open it with `"r"` mode. A program that reads data from the file will be illustrated as below:

```
1 file = open("test.txt", "r")
2 a = file.read(3)
3 b = file.readline()
4 print(a, b, sep="--")
5 file.close()
```

Program 8.3: The program opens file - writes contents and closes file

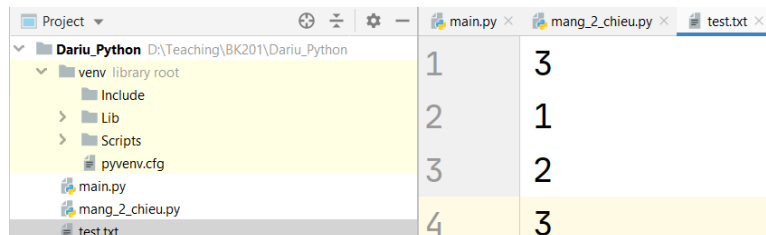
The two statements that we will often use to read values from the file, are listed as follows:

- `read(n)`: From the current position, read **n** next character, the new position increments by **n** characters.
- `readline()`: From the current position, read until the newline, the new position is on the next line.

Thus, we can read data from the file depends a lot on its current location. After performing a data read operation, the new location in the file changes. In Program 8.3, variable `a` has only 3 characters, "Don", and variable `b`, will be the remaining characters of line 1, i.e. "g 1".

4 Read 1-dimensional array from File

This is a frequently used requirement in languages like Pascal or C. To be able to read a 1-dimensional array from a file, we will usually have to specify the structure of that file. Suppose we are going to specify the first line of the file as the number of elements in the array, the subsequent lines as the elements of the array, each line containing only one element. The contents of such a file will look like this:



1	3
2	1
3	2
4	3

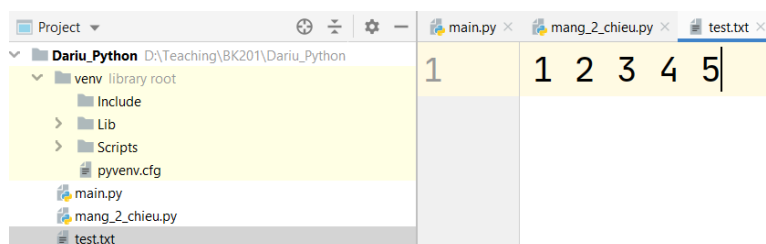
Figure 8.2: File contains data of a 1-dimensional 3-element array

Obviously, reading data from this file and storing it in an array is quite simple: read the number of elements and use a for loop to continue reading. The main command to use is `readline()`, as shown below:

```
1 file = open("test.txt", "r")
2 a = []
3 N = int(file.readline())
4 for i in range(0,N):
5     temp = int(file.readline())
6     a.append(temp)
7 print(a)
8 file.close()
```

Program 8.4: Read 1-dimensional from file

A more complicated problem is that in the file there is no information about the size. The elements of the array are separated by a space and written consecutively on a row as shown in Figure 8.3.



1	1	2	3	4	5
---	---	---	---	---	---

Figure 8.3: One-dimensional array without size

With the above requirements, reading input with other languages can be complicated. However, with the support of Python, this job is extremely simple. We just have to read the entire line of data, and cut it out with the command `split()`. The suggested program for this request is as follows:

```

1 file = open("test.txt", "r")
2 a = []
3 data = file.readline().split()
4 for i in data:
5     temp = int(i)
6     a.append(temp)
7 print(a)
8 file.close()

```

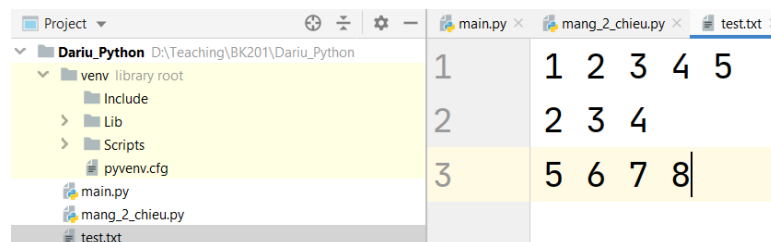
Program 8.5: Read array with no element number information

After using the **split()** operator, the **data** variable itself is a 1-dimensional array. However, its elements are being string literals. The next for loop simply converts the character type to the integer type, and then appends to our **a** array. An important error that users can encounter, is to manually press the Enter character to add the Enter character to the text file. This makes casting to integer an error.

Through this example, we also want to show another way to use the for loop. With this usage, we **not interested in the index of the element in the array**, but only interested in traversing all the elements in the array. In this usage, the variable **data** is treated as a set, and **i** will be the values in that set, respectively.

5 Read multidimensional array from File

In this request, we will read a 2-dimensional array from the file: each row of the file is a row of the matrix and there is no constraint that the number of elements in each row must be equal. The data of the test.txt file is as follows:



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

Figure 8.4: Multidimensional array data in file

The suggested program for this request would be as follows:

```

1 file = open("test.txt", "r")
2 a = []
3
4 for i in file:
5     data = i.split()
6     temp_row = []
7     for j in data:
8         temp = int(j)
9         temp_row.append(temp)
10    a.append(temp_row)
11 print(a)

```

```
12 file.close()
```

Program 8.6: Read multidimensional array from file

Again, the new for loop structure is also present for the file object. Because of the flexibility in Python, the program's statements have been greatly shortened. These tools are presented by us because we want to shorten the time for processing input data, so that learners can focus on handling more complex computational algorithms. In fact, a file is also a set structure, with each of its elements being a row. Therefore, we can write the **for i in file** loop as above. Once row-by-row is accessed, splitting each element and casting to an integer is similar to reading a one-dimensional array.

6 Exercise

1. Read the value of a 1-dimensional array from the file "input.txt" and output the result to the screen. Suppose, the values in the file are all on one line, separated by a space.

```
1 # file input.txt put at same place with source code
2 f = open("input.txt", encoding = 'utf-8')
3 x = f.readline()
4 print(x)
5 f.close()
```

Program 8.7: Suggested Answers for Exercise 1

2. Write the value of each element of a given one on each line to the file "output.txt"

```
1 a = [1,2,3,4,5]
2 f = open("output.txt", "w")
3 for i in range(0, len(a)):
4     f.write(str(a[i]) + "\n")
5 f.close()
```

Program 8.8: Suggested Answers for Exercise 2

3. Read the value of a 1-dimensional array from the file "input.txt", calculate the average value of that array and output the result to the screen. Suppose, the values in the file are all on one line, separated by a space.

```
1 # file input.txt put at same place with source code
2 f = open("input.txt", encoding = 'utf-8')
3 x = f.readline()
4 a = x.split()
5 tb = 0
6 for i in range(0, len(a)):
7     tb = tb + int(a[i])
8 tb = tb / len(a)
9 print(tb)
10 f.close()
```

Program 8.9: Suggested Answers for Exercise 3

4. Similar to exercise 3 but the final value is written to the file "output.txt"

```
1 # file input.txt put at same place with source code
2 f = open("input.txt", encoding = 'utf-8')
```

```

3 x = f.readline()
4 a = x.split()
5 tb = 0
6 for i in range(0, len(a)):
7     tb = tb + int(a[i])
8 tb = tb / len(a)
9 fw = open("output.txt", "w")
10 fw.write("Average: " + str(tb))
11 f.close()
12 fw.close()

```

Program 8.10: Suggested Answers for Exercise 4

7 Review questions

1. What does opening file in 'wb' mode mean?
 - A. Open the file for writing.
 - B. Open the file for reading and writing.
 - C. Opens the file for writing to binary.
 - D. Open the file for reading and writing for binary.

2. What does the following code mean?

```

1 f = open("sample.txt")

```

- A. Open the sample.txt file that is allowed to read and write to the file.
- B. Opens the sample.txt file and is only allowed to read the file.
- C. Open the sample.txt file and have permission to overwrite the file
- D. Open the sample.txt file and allow writing to the file.

3. What does the following code mean?

```

1 f = open("sample.txt", "a")

```

- A. Open the sample.txt file that is allowed to read and write to the file.
- B. Opens the sample.txt file and is only allowed to read the file.
- C. Open the sample.txt file and have permission to overwrite the file
- D. Opens the sample.txt file and is allowed to write further to the file

4. What happens when I open a file that doesn't exist?
 - A. Python automatically creates a new file under the name you are calling.
 - B. Nothing happens because the file does not exist.
 - C. Report an error
 - D. There are no correct answers

5. Given the following directory tree:

```
University/
|
├── ClassA/
|   ├── student1.gif
|   └── student2.gif
|
├── ClassB/
|   └── student3.gif
|
└── University.csv
```

Which of the following is the absolute path to the student1.gif file? Let's say the path only starts from University.

- A. University/student1.gif
 - B. University/ClassA/student1.gif.
 - C. /student1.gif
 - D. University/
6. Which of the following statements will read the file from the current position until the next line, the new position is on the next line?
- A. read()
 - B. readline()
 - C. read(n)
 - D. All of the above answers are incorrect.
7. Which of the following statements will read the next n characters from the current position, increase the new position by n characters?
- A. read()
 - B. readline(n)
 - C. read(n)
 - D. All of the above answers are incorrect.

Answer

1. C 2. B 3. D 4. C 5. B 6. B 7. C

CHAPTER 9



Functions and function calls

pythonTM
Package
Index

1 Introduction

In the previous lessons we have used some functions in Python such as `print()`, `open()`, `write()`... So what is a function? In programming, a function is a group that includes one or more statements and is used to perform certain tasks.

Functions are divided into two groups:

- Built-in functions are functions provided by the Python language. For example the functions `print()`, `range()`, `max()`... We do not modify the logic inside these functions.
- User-defined functions are functions defined by programmers. Programmers will declare their own logic inside self-defined functions.

2 Function definition

In Python (self-defined) functions are defined using the following syntax:

```
1 def <function_name>(<parameters>):  
2     <statements>
```

In there:

- *<function_name>*: is the name of the defined function.
- *<parameters>*: (parameters) is a list of special variables used inside the function. The value of each parameter is determined when we use the function (or **function call**). A function is defined with one or more or no parameters.
- *<statement>*: contains the statements that will be executed when calling the function.

For example, a self-defined function `helloPython()` looks like this:

```
1 def helloPython():  
2     print("Hello, Python!")  
3  
4 helloPython() # Shown: Hello, Python!
```

The following `hello()` function example is defined with an input **parameter** named `language`:

```
1 def hello(language):  
2     print("Hello, %s!" %(language))
```

3 Function Call

Once defined, to use the function we will make a function call. Calling the function is done simply through the following syntax:

```
1 <function_name>(<arguments>)
```

In there:

- `<function_name>` is the name of the function being called.
- `<arguments>` (arguments) are the values passed to the parameters defined by the called function.

The following example will call the `hello()` function defined in the above example with the input Python argument:

```
1 hello("Python") # Shown: Hello, Python!
```

The number of arguments passed when calling the function needs to correspond to the number of parameters used when defining the function. If the number of arguments and arguments do not match, Python will give an error at runtime.

For example, if you call the `hello()` function defined above as follows:

```
1 hello()
```

You will see Python prints error:

```
1 Traceback (most recent call last):
2   File "function.py", line 4, in <module>
3     hello()
4 TypeError: hello() takes exactly 1 argument (0 given)
```

3.1 Default Arguments

When defining a function that uses parameters we can assign default values (or default arguments) to the parameters.

In the example below, the `hello()` function is defined with a language parameter with a default value of Python:

```
1 def hello(language = "Python"):
2     print("Hello, %s!" %(language))
```

When calling the `hello()` function in the above example, if we pass no arguments, the default argument will be used:

```
1 hello() # Shown: Hello, Python!
```

3.2 Return Value

In the previous examples, the `hello()` function is defined to perform the task of displaying the message on the screen. However, in many cases we don't want to display any message when calling the function, instead we need to get the return value from the function. To do this we will use the return keyword.

For example:

```
1 def sum(a, b):
2     return a + b;
```

In the above example, the function `sum()` will return the sum of two parameters `a` and `b`. We can use this return value to display a message or assign to another variable...

```

1 number_1 = 3
2 number_2 = 4
3
4 def sum(a, b):
5     return a + b
6
7 total = sum(number_1, number_2)
8 print("Total %s and %s is %s" %(number_1, number_2, total))

```

The return value when calling the above sum() function is assigned to the total variable. And so when running the above code, it will display the result: *Total 3 and 4 is 7.*

Functions that do not use return or use return (with no value following the keyword) will return None:

```

1 def hello_1():
2     print("Hello 1!")
3
4 def hello_2():
5     print("Hello 2!")
6     return
7
8 returnValue = hello_1()
9 print(returnValue) # Shown: None
10 print(type(returnValue)) # Shown: <type 'NoneType'>
11
12 return_value = hello_2()
13 print(returnValue) # Shown: None
14 print(type(returnValue)) # Shown: <type 'NoneType'>

```

Another note when using return in a function is that once the function has returned (returning a value), Python will terminate the execution of the function. For example:

```

1 def hello():
2     print("Hello!")
3     return
4     print("Nice to meet you!")
5
6 hello() # Shown: Hello!

```

In the above code, when calling hello(), only the first print("Hello!") statement is executed.

3.3 Command pass

A self-defined function cannot be left blank (i.e. cannot be without a function), however for some reason the self-defined function has no content. Then the statement **pass** will make the function definition non error. For example:

```

1 def myfunction():
2     pass

```

4 Write a function to calculate factorial

Based on what is known about the function, we will proceed with an example of how the function is defined. Example: Factorial function. And we will implement the definition of factorial function in two ways: for loop and recursion.

In mathematics, factorial is a unary operator on the set of natural numbers. Let n be a positive natural number, "n factorial", denoted $n!$ as the product of n of the first positive natural number.

$$n! = 1 \times 2 \times 3 \times \dots \times n$$

Example: $4! = 1 \times 2 \times 3 \times 4 = 24$

$8! = 1 \times 2 \times 3 \times \dots \times 7 \times 8 = 40320$

Especially, with $n = 0$, convention $0! = 1$.

4.1 Write function using for loop

```
1 def cal_factorial(n):
2     factorial = 1;
3     if (n == 0 or n == 1):
4         return factorial;
5     else:
6         for i in range(2, n + 1):
7             factorial = factorial * i;
8         return factorial;
9
10 n=10
11 print("Factorial of", n, "is", cal_factorial(n));
```

Result: *Factorial of 10 is 3628800*

4.2 Write function using recursion

```
1 def cal_factorial(n):
2     if n == 0:
3         return 1
4     return n * cal_factorial(n - 1)
5
6 n=9
7 print("Factorial of", n, "is", cal_factorial(n));
```

Result: *Factorial of 9 is 362880*

5 Exercise

1. Write a printMax function that takes two input values a and b. Then print to the screen larger numbers. If the two numbers are equal, just print one number to the screen.

```
1 def printMax(a, b):
2     if a > b:
3         print(a)
4     elif a == b:
5         print(b)
6     else:
7         print(b)
8 a = int(input("Enter first number: "))
9 b = int(input("Enter second number: "))
10 printMax(a, b)
```

Program 9.1: Suggested Answers for Exercise 1

2. Let the Fibonacci Sequence be calculated based on the following formula:

$f(n) = 0$ if $n = 0$

$f(n) = 1$ if $n = 1$

$f(n) = f(n-1) + f(n-2)$ if $n > 1$

Write a function that recursively computes the value of $f(n)$ where n is the number entered by the user and returns the corresponding value. Example: If n is entered as 7 then the program will return 13.

```
1 def f(n):
2     if n == 0: return 0
3     elif n == 1: return 1
4     else: return f(n-1)+f(n-2)
5
6 n = int(input("Enter n: "))
7 a = f(n)
8 print (a)
```

Program 9.2: Suggested Answers for Exercise 1

3. Write a function that calculates the power x and the y exponent (do not use the $**$ operator), with x, y entered from the keyboard and $x, y > 0$. If x or y is less than 0, return 0.

```
1 def power(x,y):
2     result = 1
3     if x < 0 or y < 0:
4         result = 0
5     else:
6         for i in range (0,y):
7             result = result * x
8     return result
9
10 x = int(input("Enter base: "))
11 y = int(input("Enter exponent: "))
12 a = power(x,y)
13 print(a)
```

6 Review questions

1. Choose the correct answer: Which statement is correct about Functions in Python?
 - A. The function can be reused in the program.
 - B. Using functions has no positive effect on the modules in the program.
 - C. The programmer's own functions cannot be created.
 - D. All of the above answers are correct.
2. What keyword is used to start the function?
 - A. Fun
 - B. Define
 - C. Def
 - D. Function

3. What is the output of the program below?

```
1 def sayHello():  
2     print('Hello World!')  
3 sayHello()  
4 sayHello()
```

A.
Hello World!
Hello World!

B.
'Hello World!'
'Hello World!'

C.
Hello
Hello

D. There is no correct answer

4. What is the output of the program below?

```
1 def printMax(a, b):  
2     if a > b:  
3         print(a, 'is maximum')  
4     elif a == b:  
5         print(a, 'is equal to', b)  
6     else:  
7         print(b, 'is maximum')  
8 printMax(3, 4)
```

A. 3
B. 4
C. 4 is maximum
D. There is no correct answer

5. What is the output of the program below?

```

1 x = 50
2     def func(x):
3         print('Value of x is', x)
4         x = 2
5         print('Value of x is changed to', x)
6 func(x)
7 print('Present value of x is', x)

```

- A. Present value of x is 50
- B. Present value of x is 100
- C. Present value of x is 2
- D. There is no correct answer

6. What is the output of the program below?

```

1 x = 50
2     def func():
3         global x
4         print('Value of x is', x)
5         x = 2
6         print('Value of x is changed to', x)
7 func()
8 print('Present value of x is', x)

```

- A.
Value of x is 50
Value of x is changed to 2
Present value of x is 50
- B.
Value of x is 50
Value of x is changed to 2
Present value of x is 2
- C.
Value of x is 50
Value of x is changed to 50
Present value of x is 50
- D. There is no correct answer

7. What is the output of the program below?

```

1 def maximum(x, y):
2     if x > y:
3         return x
4     elif x == y:
5         return 'Equal numbers'
6     else:
7         return y
8 print(maximum(2, 3))

```


- A. 2
- B. 3
- C. Equal numbers
- D. There is no correct answer

8. What is the output of the following program?

```
1 def outerFunction():
2     global a
3     a = 20
4     def innerFunction():
5         global a
6         a = 30
7         print('a =', a)
8 a = 10
9 outerFunction()
10 print('a =', a)
```

- A. a = 10 a = 30
- B. a = 10
- C. a = 20
- D. a = 30

9. What is the output of the program below?

```
1 def cube(x):
2     return x * x * x
3 x = cube(3)
4 print x
```

- A. 9
- B. 3
- C. 27
- D. 30

10. What is the output of the program below?

```
1 def C2F(c):
2     return c * 9/5 + 32
3 print C2F(100)
4 print C2F(0)
```

- A.
212
32
- B.
314
24
- C.
567

D. There is no correct answer

Answer

1. A 2. C 3. A 4. C 5. A 6. B 7. B 8. C 9. C 10. A

CHAPTER 10



Python Advanced Data Structures

pythonTM
Package
Index

1 Introduction

In the previous tutorial, we have learned through the basic data types such as integers, decimals or even lists. For these basic data types, we can already write simple applications with Python. However, for applications with a large number of variables, large and complex data, we need more advanced data structures to make writing concise and efficient code.

2 String (string)

String is a very common data type in Python, a word, a piece of text is a string type. Strings in Python are marked with single quotes ' or double quotes ", but if they start with any sign, they must end with that. For example:

```
1 a = "Hello"
2 b = 'Hello\n' # \n means new line
3
4 # c = 'Hi", this initialization is wrong
```

Program 10.1: Example for how to initialize string

Python supports the string data type that can contain paragraphs with multiple lines by starting and ending the string with 3 double quotes """. For example:

```
1 print("""\
2     Hi
3     Hello
4 """)
```

Program 10.2: Write string on multiple lines

Strings can be viewed as a list of characters, so the way to traverse and access the elements of the string is similar to how we do with lists (using indexes). As follows:

```
1 a = "Hello"
2 for i in range(0, len(a)):
3     print(a[i])
```

Program 10.3: Traverse and access the elements of the string

2.1 Concatenate string, change or delete string

Strings can be concatenated with the + operator and replaced with *:

```
1 a = "Hello"
2 b = "Viet Nam"
3 c = a + b # c = "Hello Viet Nam"
```

Program 10.4: Concatenate 2 strings

Python strings cannot be changed - they are fixed. So, if we try to assign a certain character to an indexed location, we will get an error message. So if a new string is needed, the best way is to create a new one.

```

1 a = "Hello"
2 # a[1] = a, this assigned is error

```

Program 10.5: Cannot change value in string

2.2 Methods for string variables

There are many built-in methods in Python to work with strings such as:

- `lower()`: Converts all letters in a string to lowercase
- `upper()`: Converts all letters in the string to uppercase
- `split(x)`: Returns a list by dividing the elements by the "x" character, which defaults to a space character if left blank.
- `find(x)`: Returns the number of times string x was found in the original string, returns -1 if not found.
- `replace(x,y)`: Replace the string x with the string y in the original string.
- ...

3 Tuple data structure

Python supports a data structure similar to List, named Tuple. However, unlike list, tuple is an immutable list. That is after initializing the tuple, we cannot change it.

3.1 Initialize and retrieve elements in a tuple

Tuples are declared in Python with a string (), in the middle are the elements of the array, separated by a comma (.). We can leave out the parenthesis if we want, but it's a good idea to add it for clearer code. In addition, tuples are not limited to the number of elements and can have many different data types such as integer, decimal, list, string,...

However, if it is not enough to create a tuple in the usual way, putting the element in a pair of () quotes, a comma is needed to indicate that this is a tuple. Take a look at the following example

```

1 a = (1, 2, 3, 4)
2 #tuple can be initialized without ()
3 b = 5,6,7,8
4 #note when initialize tuple
5 c = ("Hello") # type of c is string
6 d = ("Hello",) # type of d is tuple

```

Program 10.6: Initialize tuple

Similar to the list, to access the elements inside the tuple, we use the index [] operator with the index starting with 0.

```

1 a = (1, 2, 3, 4)
2 print("First element: ", a[0])
3 print("Second element: ", a[1])

```

Program 10.7: Access elements in a tuple

3.2 Tuple operations

We can use all the same techniques and functions as we use with list. However, a tuple is an immutable list that cannot be changed once created, so we will exclude functions that change the contents.

For example, we can use `len()` to traverse the elements in the tuple, or `index(x)` to return the index value of the first `x` element it encounters in the tuple:

```

1 a = (1, 2, 3, 4)
2 print(a.index(1)) # result is 0
3 N = len(a)
4 for i in range(0,N):
5     print(a[i])

```

Program 10.8: Traverse the elements in the tuple

For the rest, all methods like: `append()`, `insert()`, `pop()`, `extend()`, `remove()`, `sort()`, `reverse()`,... can't be used with tuples.

However, if the elements themselves are a mutable data type (like list for example) then the nested elements can be changed. Specifically, see the illustrative example below:

```

1 a = (1, 2, 3, [4,5])
2 # If changing the value of a tuple by assigning a[0] = 7 it
   # will error
3 # Only values of the elements in the list [4,5] can be
   # changed, because list is mutable
4 a[3][0] = 5;
5 print(a)

```

Program 10.9: Change values inside tuple

3.3 When to use Tuple

Tuples have certain limitations such as they cannot be changed when created, but there are also significant advantages to the following:

- Firstly, tuple is faster to process than list. Also, when we want to define a set of values as constant and then iterate through this set, we should choose tuple.
- Using tuples makes code safer, because tuple properties make data immutable. Therefore, tuples should be chosen for constant data, which do not change over time.

4 Set Data Structure

Set can contain many elements and these elements have no order, its position is chaotic in the set..

We can iterate through the elements in the set, can add or remove elements, and perform set operations such as union, intersection, difference, etc. Besides, the elements of the set must be immutable data such as a number (int), a string, or a tuple.

4.1 Initialize and retrieve elements of a set

We need to pay attention to the properties of set as follows:

- The elements in the set are unordered.
- These elements are unique, no repetitions are allowed.
- Sets are mutable (adding or removing elements) but the elements of the set must be immutable.

The elements in the set are separated by commas and enclosed in curly braces . And due to the nature of the set, we cannot use the index [] operator to access the elements, but have to access those elements directly. Specifically, like the example below:

```
1 a = {1, 2, 3, 4}
2 b = {{7.0, "Hello", (4, 5, 6)}}
3 c = set() # initialize an empty set
4
5 #traverse and print all elements in set
6 for x in b:
7     print(x)
```

Program 10.10: Initialize and traverse elements in set

4.2 Methods of set

Python supports a lot of methods to perform set changes, and note that the result may be different because the set does not arrange the elements in any order.

- add(): The method used to add an element to the set.
- remove(): Removes an element from the set and returns an error if the element does not exist.
- discard(): Same as remove(), but if the element does not exist, no error will be reported.
- pop(): Removes a random element from the set.
- clear(): Removes all elements in the set.
- update(): Used to add more elements to the collection.

4.3 Set operations

Set has an advantage over other data structures in that they can perform set operations such as union, difference, intersection, etc. And to better understand the operations commonly used in set, we consider the following examples:

4.3.1 Union

The union of two sets results in all the elements in the two sets, notice that any repeated element will appear only once in the result set.

```
1 a = {1, 2, 3}
2 b = {3, 4}
3 c = a.union(b)
4 print(c) # result is {1,2,3,4}
```

Program 10.11: Union in set

4.3.2 Intersection

The intersection of two sets results in elements that belong to both sets at the same time.

```
1 a = {1, 2, 3}
2 b = {3, 4}
3 c = a.intersection(b)
4 print(c) # result is {3}
```

Program 10.12: Intersection in set

4.3.3 Difference

The difference of a set A minus a set B results in all the elements that are in A but not in B.

```
1 a = {1, 2, 3}
2 b = {3, 4}
3 c = a.difference(b)
4 print(c) # result is {1, 2}
```

Program 10.13: Difference in set

4.3.4 Symmetric difference

The difference in symmetry of two sets A and B results in the set of elements that belong to both A and B but not to both A and B at the same time.

```
1 a = {1, 2, 3}
2 b = {3, 4}
3 c = a.symmetric_difference(b)
4 print(c) # result is {1, 2, 4}
```

Program 10.14: Symmetric difference in set

4.4 When to use set

A set is also a data structure like list and tuple, but it has built-in set operations, so if our program needs to perform operations like: `union()`, `intersection()`, ... then we can consider using set. In addition, it is also necessary to pay attention to the characteristics of set to apply to problems so that it is reasonable.

5 Dictionary Data Structure

Dictionary (also known as dict) consists of many elements that each follow a key-value pair. Dictionary is often used when we have a large amount of data and want to optimize them for data extraction provided that the key to get the value is known.

5.1 Initialize and retrieve dictionary elements

In python, dictionary is defined in curly braces similar to set. But inside are the elements in pairs (key and value) separated by commas, separating the key and value of each element by a colon. Example:

```
1 a = dict({"A": 24, "B": 30, "C": 27})
2 b = dict([("D", 24), ("E", 30), ("F", 27)])
3 # iterate through the elements in dictionary a
4 for key in a:
5     print(key)
```

Program 10.15: How to initialize a dictionary

Note:

- The dictionary key can be a string or a number, which is unique at each level.
- The value of a dictionary can be numbers, strings, or lists, tuples, sets, or even another dictionary.

We can access the elements in the dictionary through keys, specifically as follows:

```
1 a = dict({"A": 24, "B": 30, "C": {"D": 21}})
2 print(a[A]) # result is 24
3 #retrieve elements in nested dictionary
4 print(a["C"]["D"]) #result is 21
```

Program 10.16: Retrieve elements in a dictionary

5.2 Add and update dictionary elements

Dictionary is mutable, so we can add new or change the value of existing elements using the assignment operator. And because it is difficult for dictionaries to access the elements, when assigning a value to an element:

- The key exists then the element value is updated.
- The key does not exist, then the dictionary is added an element with the key and value in the statement.

```

1 dict = {"a": 1, "b": 2}
2 # add an element into a dict
3 dict["c"] = 2 # result: {'a': 1, 'b': 2, 'c': 2}
4 # update value of element in dict
5 dict["c"] = 3 # result: {'a': 1, 'b': 2, 'c': 3}

```

Program 10.17: Add and update element in dictionary

5.3 Remove an element from the dictionary

Like list, there are many methods available to remove an element from the dictionary like `pop()`, `popitem()`, `clear()`, `del`. The functions of the above methods are as follows:

- `pop()`: Removes the element with the given key and returns its value.
- `popitem()`: Removes and returns a random element as (key, value)
- `clear()`: Clear all the elements in the dictionary
- `del`: Delete an element or completely delete the variable containing the dictionary.

```

1 dict = {"a": 1, "b": 2, "c": 3}
2 a = dict.pop("a") # a = 1, dict = {"b": 2, "c": 3}
3 del dict["c"] #dict = {"b": 2}
4 dict.clear() # dict = {}
5 del dict
6 print(dict) #dict not exist

```

Program 10.18: Methods for removing elements from a dictionary

5.4 Some common dictionary methods

- `get()` method: Returns the value of an element in the dictionary with a given key.
- `key()` and `values()` method: Returns the dictionary's list of keys and values

```

1 dict = {"a": 1, "b": 2, "c": 3}
2 a = dict.get("a") # a = 1
3 b = dict.keys() # b = ['a', 'b', 'c']
4 c = dict.values() # c = [1, 2, 3]

```

Program 10.19: Some common dictionary methods

5.5 When to use dictionary

We can consider using a dictionary when the data is constantly changing, there is a link between the key and the value, or importantly that we want to increase efficiency when accessing data through the key.

6 Exercise

1. Enter any 2 strings from the keyboard, concatenate those 2 strings together. Print the result string and indicate how many characters it has.

```
1 a = input("Enter first string: ")
2 b = input("Enter second string: ")
3 result = a + b
4 print(result)
5 print("Length of above string: ", len(result))
```

Program 10.20: Suggested Answers for Exercise 1

2. Write a program to create another tuple, containing even values in a given tuple.
Hint: Use tuple() to create a tuple from a list.

```
1 a = (1,2,3,4,5,6,7,8)
2 tp = list()
3 for i in range(0, len(a)):
4     if a[i]%2==0:
5         tp.append(a[i])
6 tp1 = tuple(tp)
7 print(tp1)
```

Program 10.21: Suggested Answers for Exercise 2

3. Write a program to remove duplicates between two given sets:

```
1 a = {1,2,3,4,5,6,7,8}
2 b = {2,4,6,8}
3 tp = a.difference(b)
4 print(tp)
```

Program 10.22: Suggested Answers for Exercise 3

4. Enter integer N from the keyboard, write a program to generate a dictionary containing (i, i*i) as integers from 1 to N (including 1 and n) then print the dictionary this.

Example: Suppose n is 8 then the output will be: 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64.

```
1 n=int(input("Enter integer N:"))
2
3 d=dict()
4 for i in range(1, n+1):
5     d[i]=i*i
6 print(d)
```

Program 10.23: Suggested Answers for Exercise 4

7 Review questions

1. Given the following code:

```
1 a = "Hi Viet Nam"
2 print(a.find("i"))
```

What is the output of the above command?

- A. 1
- B. 2
- C. 3
- D. i

2. Which of the following is used to declare a tuple:

- A. `a = [1,2,3,4]`
- B. `a = (1,)`
- C. `a = (1,2,3,4)`
- D. `a = (1)`

3. Given the following code:

```
1 a = (1, 2, 3, 4)
2 a[1] = 3
3 print(a)
```

The output of the above command will be:

- A. (1,3,3,4)
- B. (3,2,3,4)
- C. (1,2,3,4)
- D. Report an error

4. The elements of a set can be accessed using the index [] operator. Is the above statement true or false?

- A. True
- B. False

5. `a = {1, 2, 3}`

`b = {3, 4}`

`a = a.union(b)`

The value of a after executing the above command is:

- A. `a = 1,2,3`
- B. `a = 1,2,3,4`
- C. `a = 4`
- D. Report an error because the set cannot be changed

6. In a dictionary, keys can be the same. Is the above statement true or false?

- A. True
- B. False

7. Given the following code:

```
1 dict = {"a": 1, "b": 2}
2 print(dict["a"])
```

What is the above command used for:

- A. To retrieve the element whose key is "a"
- B. To print an element whose key is "a"
- C. To print the character "a" to the screen.

Answer

1. A 2. B 2. C 3. D 4. A 5. B 6. B 7. B

Part II

Application Projects

CHAPTER 11



Python's Virtual Assistant - Text to Speech

pythonTM
Package
Index

1 Introduction

In this tutorial, we will show the most basic steps to make a simulation application for Virtual Assistant. In this application there will be 3 main parts: Speech recognition from the user, Intelligence of the computer, and finally Reply back to the user. In these 3 parts, we can see Speech Recognition as **input (Input)**, **Computer intelligence as Processing**, and finally, as **output (Output)**.

With a project, analyzing what is input data, output data is a very important step, before starting to program for processing. Usually, when comparing the complexity of input and output data, the output is simpler, and should be executed first.

In this tutorial, a simple program will be implemented so that the computer answers the results back to us in the form of voice. This technique is called "Text To Speech". That is, by programming, we give the computer a string of data, which it will say out loud through the computer's speakers. The steps below will detail how to implement the "Text To Speech" function, the output for the Virtual Assistant project.

2 Create a new application

For the previous tutorials, we were familiar with adding a new python file to the program and running it. However, when implementing a new project, we should recreate the application from scratch for ease of management. The sequence to create a new application is shown below.

Step 1: From the File menu, we choose **New Project...**, as instructed in Figure 11.1.

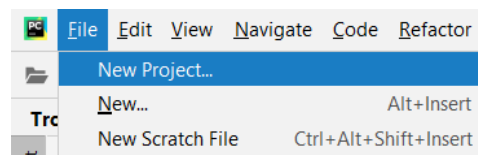


Figure 11.1: Create a new project

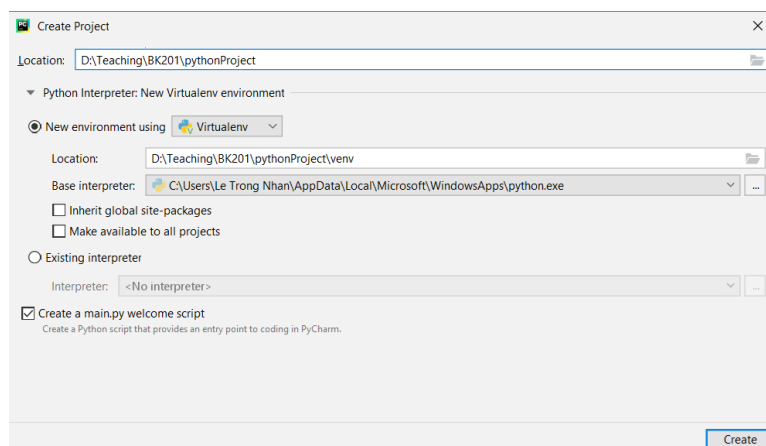


Figure 11.2: Choose path to save project

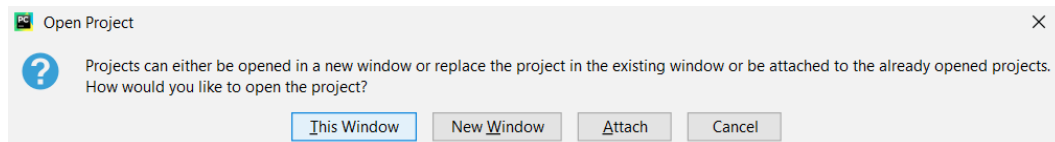


Figure 11.3: Choose This Window to close the old project and open a new one

Step 2: Next, we need to choose the path to save the new project, at **Location**, as instructed in Figure 11.2. Finally, click the **Create** button.

Step 3: Finally, the system will appear a message, meaning that we will close the current project, and open the new project in this window, or keep the old project open and a new window for the newly created project. Here, we recommend that you select **This Window** to close the old project and open only the newly created project, as shown in Figure 11.3. However, we can't start programming yet. For progressively larger projects, the first step is to install the software that supports the implementation of the program, which will be covered in the next section of this tutorial.

3 Install library

In fact, when implementing complex projects, it will be difficult to start everything from scratch. Usually, we will install additional external libraries to support our programming. This is also the strength of the open source programming community, and Python is an example. When referring to large projects, we will easily realize, installing libraries will be the first steps.

The library installation process will be complicated in that, if the library you need to use, for example, library A, it depends on another library, such as library B. This problem will appear when you are installing library A, and the system will give an error that library B is missing. You need to install library B first, then reinstall library A.

In this tutorial, we will install the following libraries one by one: **wheel**, **pipwin**, **pyaudio** and **pytsx3**. This is the best installation order we've found so that the installation won't fail.

Step 1: To start the installation, we need to switch to the **Terminal** interface, as shown in Figure 11.4

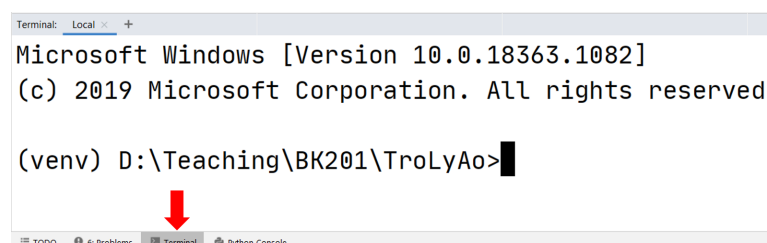
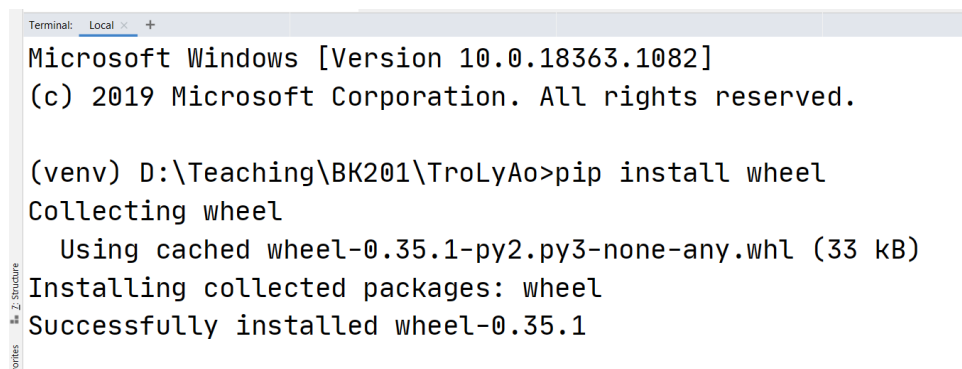


Figure 11.4: Terminal interface to install library

From here, type in **pip install wheel** and then press Enter. We also need an internet connection so that the program can download the library before installation. In case the network is too poor, the time allowed, the system will also automatically stop the installation

and need to do it again, until see the words **Successfully installed wheel**, as shown in the figure below. Figure 12.1. Here, pip is a tool to install the wheel library.



```
Terminal: Local x +
Microsoft Windows [Version 10.0.18363.1082]
(c) 2019 Microsoft Corporation. All rights reserved.

(venv) D:\Teaching\BK201\TroLyAo>pip install wheel
Collecting wheel
  Using cached wheel-0.35.1-py2.py3-none-any.whl (33 kB)
Installing collected packages: wheel
Successfully installed wheel-0.35.1
```

Figure 11.5: Install wheel library successfully

Step 2: Similarly, we will install the pipwin library again using the pip tool, with the command **pip install pipwin**.

Step 3: To install the **pyaudio** library, we cannot install using the pip tool. The reason is that it is not compatible with some audio drivers on Windows computers. We need to install this library using the pipwin tool, ie type **pipwin install pyaudio** on the Terminal window, and then press Enter.

Step 4: Finally, we install the pyttsx3 library using the pip tool, ie type **pip install pyttsx3** and then press Enter.

4 Program implementation

Back to the main.py file, we begin to implement the program below.

```
1 import pyttsx3
2
3 engine = pyttsx3.init()
4 engine.say("I am AI python")
5 engine.runAndWait()
```

Program 11.1: Text-to-speech program

With the support library pyttsx3 (which is taken into the program by the import statement on line 1), our job is quite simple: Initialize (line 3), pass parameters (line 4) and finally, give Allows the program to speak out. Please note that the pyttsx3 library currently only supports English pronunciation.

5 Review questions

1. What is the window used to install libraries on PyCharm called?
 - A. Terminal
 - B. Log
 - C. Console
 - D. All correct
2. Which of the following statements is used to install the **wheel** library?
 - A. install wheel
 - B. pip install wheel
 - C. wheel install pin
 - D. All correct
3. To install the pipwin library, which of the following statements is used?
 - A. install pipwin
 - B. pipwin install pip
 - C. pip install pipwin
 - D. All wrong
4. To install the pyaudio library, which of the following statements is used?
 - A. pipwin install pyaudio
 - B. pip install pyaudio
 - C. pip install pipwin pyaudio
 - D. All wrong
5. What is the main library for text-to-speech conversion?
 - A. pipwin
 - B. pyaudio
 - C. pips
 - D. pyttsx3
6. To be able to write a text-to-speech program, what library should be added at the beginning of the program? to voice?
 - A. import pipwin
 - B. import pyaudio
 - C. import pyttsx3
 - D. All correct
7. Which statement is used to **initialize** for speech output from text?
 - A. engine = pyttsx3.init()
 - B. engine = pyttsx3.say("Init")
 - C. engine.runAndWait()
 - D. All correct
8. What command is used to **pass text parameters** for voice output?
 - A. engine = pyttsx3.init()
 - B. engine = pyttsx3.say("Init")
 - C. engine.runAndWait()
 - D. All correct

9. What command is used to trigger **sound** when writing a text-to-speech program?
- A. `engine = pyttsx3.init()`
 - B. `engine = pyttsx3.say("Init")`
 - C. `engine.runAndWait()`
 - D. All correct
10. What is text-to-speech technique called?
- A. Saying Text
 - B. Speech To Text
 - C. Text To Speech
 - D. All wrong
11. What role does voice-to-text play in a virtual assistant system?
- A. Input
 - B. AI Processing
 - C. Output
 - D. All correct
12. What is the supported language for the pyttsx3 library?
- A. Vietnamese
 - B. English
 - C. Vietnamese and English
 - D. All languages

Answer

1. A 2. B 3. C 4. A 5. D 6. C 7. A 8. B 9. C 10. C 11. C 12. B

CHAPTER 12



Python's Virtual Assistant - Voice Recognition

pythonTM
Package
Index

1 Introduction

In the next part of the virtual assistant project, we will implement the voice recognition function from the user. This function is considered as the input part of the project, also known as Speech To Text. Just like the voice output function in the previous article, this is a very complicated task. However, with the support from available libraries, our programming work becomes simpler.

However, for the data entry function, we always have to come with error handling, so that the application can work more correctly. For example we expect the user to say a sentence, but the user does not speak at all, or for example, for other reasons, such as loss of life, and the system does not receive the result. All these functions make the input data processing always more complicated than the output data.

2 Install library

To perform speech recognition function, we need to install an additional library called SpeechRecognition to perform speech recognition function. The order to do it consists of 2 steps as follows:

Step 1: To start the installation, we need to switch to the **Terminal** interface, as shown in Figure 12.1.

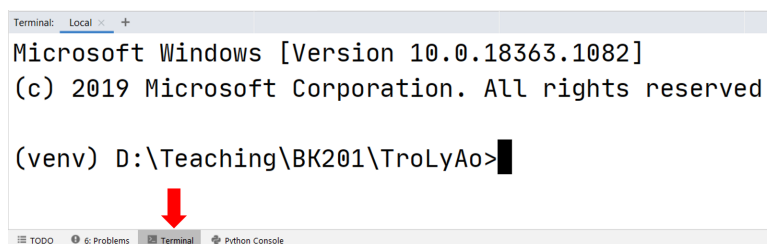


Figure 12.1: Terminal interface to install library

Step 2: Type **pip install speechrecognition** and press Enter. We also need an internet connection so that the program can download the library before installation. In case the network is too poor, the time allowed, the system will also automatically stop the installation and need to do it again, until see the words **Successfully installed speechrecognition**, as shown in the figure below. Figure 12.2.

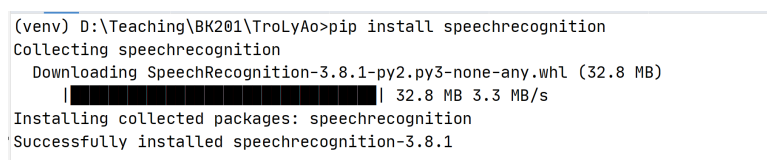


Figure 12.2: Install SpeechRecognition library successfully

Note: If you want to implement a program that only has speech recognition, you need to install wheel, pipwin, pyaudio and speechrecognition.

3 Program implementation

To implement the voice recognition function, we need the following basic commands:

```
1 import speech_recognition
2
3 voice = speech_recognition.Recognizer()
4 with speech_recognition.Microphone() as mic:
5     print("I am listening...")
6     audio = voice.listen(mic)
7 input_text = voice.recognize_google(audio)
8 print(input_text)
```

Program 12.1: Text-to-speech program

In the first line, we are adding the library to the program. Please note that the name of the library when importing, has an underscore (import speech_recognition), which is different from the conjoined name when installing the library (pip install speechrecognition). The next line is to initialize an object (named voice) for the main task of speech recognition. The command **with** is used to initialize our computer's microphone, and start recording, saving it to the audio variable. When the recording ends (we stop talking), the audio data will be recognized using the recognize_gooole method. With the result stored in the variable **input_text**, we can print it to the screen for monitoring.

In case the user says nothing, or the network fails, the command **text = voice.recognize_google(audio)** may cause an error. We will work around this with the **try except** statement. Cleverly combined with the program implemented in the previous lesson, we have the following program:

```
1 import speech_recognition
2 import pyttsx3
3
4 voice = speech_recognition.Recognizer()
5 with speech_recognition.Microphone() as mic:
6     print("I am listening...")
7     audio = voice.listen(mic)
8
9 try:
10     input_text = voice.recognize_google(audio)
11 except:
12     input_text = "I do not understand"
13 print(input_text)
14
15 engine = pyttsx3.init()
16 engine.say("I hear " + input_text)
17 engine.runAndWait()
```

Program 12.2: Recognize voice and response

In the program above, we will have the computer repeat what we just said. This is a very effective way to test input and output processing, before we begin to implement the processing, so-called "intelligence" of the assistant virtual logic. Note that the voice recognition system still only supports English. To be able to be a virtual assistant entirely in

Vietnamese, you must use more advanced libraries. Within the confines of this tutorial, we've only used the simplest libraries.

4 Review questions

1. The speech recognition process, what role does it play in the virtual assistant system?
 - A. Input processing
 - B. Processing output
 - C. Processing intelligence
 - D. All are correct
2. To handle errors in speech recognition, which command is used?
 - A. Regconizer()
 - B. try except
 - C. if else
 - D. All are correct
3. What is the main library for English speech recognition?
 - A. pyaudio
 - B. pyttsx3
 - C. speechrecognition
 - D. It's all wrong
4. What command is used to start the speech recognition program?
 - A. pyttsx3.init()
 - B. engine.say()
 - C. speech_recognition.Recognizer()
 - D. All commands above
5. What is the supported language for the speechrecognition library?
 - A. Vietnamese
 - B. English
 - C. Vietnamese and English
 - D. All languages
6. What is the command to add a speech recognition library to a Python program?
 - A. import pyttsx3
 - B. import speechrecognition
 - C. import speech_recognition
 - D. All are correct
7. What are the possible errors in speech recognition?
 - A. User says nothing
 - B. Lost internet connection
 - C. Speaks in an unrecognized system language
 - D. All of the above errors are possible

Answer

1. A 2. B 3. C 4. C 5. B 6. C 7. D

CHAPTER 13



Python's Virtual Assistant - Building intelligence

pythonTM
Package
Index

1 Introduction

This tutorial is the 3rd step, the last step when implementing a project: Processing input data, and outputting results. This must be the last step of implementation, because only when the input and output processing is stable, we will have a basis to make the algorithm more and more complete and stable.

In this tutorial, we only give the most basic ideas so that readers can implement a simple virtual assistant program, so that it becomes smarter day by day, learners can develop themselves.

In this tutorial, our virtual assistant will answer 2 questions from users, "hello" (hello) and "what is today" (what is today).

2 Retrieve current date

There are many methods to get the current date information of the system, the program below is an example.

```
1 from datetime import date
2
3 today = date.today()
4 strTextToday = today.strftime("%B %d, %Y")
5 print(strTextToday)
```

Program 13.1: Get the current system date

In the above program, we need to add the date library, then format the data string describing the current date. Of course, the result will be in English, because we are implementing a virtual assistant that can recognize and speak in English.

3 Complete the program

The simplest way to implement auto response functionality for virtual assistants is to use the **if** statements. For each command from the user input, we will have the corresponding answer. The suggested program for this part is as follows:

```
1 import speech_recognition
2 import pyttsx3
3 from datetime import date
4
5 today = date.today()
6 strTextToday = today.strftime("%B %d, %Y")
7
8 voice = speech_recognition.Recognizer()
9 with speech_recognition.Microphone() as mic:
10     print("I am listening...")
11     audio = voice.listen(mic)
12
```

```

13 try:
14     input_text = voice.recognize_google(audio)
15 except:
16     input_text = "I do not understand"
17 print(input_text)
18
19 output_text = ""
20 if input_text == "hello":
21     output = "Hello, I am AI Python"
22 elif input_text == "what is today":
23     output_text = "Today is " + strTextToday
24 else:
25     output_text = "Sorry I do not understand"
26
27 engine = pyttsx3.init()
28 engine.say(output_text)
29 engine.runAndWait()

```

Program 13.2: Complete processing for virtual assistants

In fact, the current program still needs further development so that it can be almost like a virtual assistant. Extended directions for learners are listed as follows:

- Improved matching: Currently, in the **if** statement, matching is very unlikely in practice. For example, if the user says "hello hello", our assistant will not say hello back. We will compare by keyword type: In the user's sentence with the word hello, we can say hello back. The suggested statement for this function is **if "hello" in input_text**.
- Currently, the program only runs once and then stops. This is an opportunity for learners to manipulate the command **while**: The program will continue to work until the user says **bye bye**.
- The program currently only supports 2 commands, users can define more commands by themselves, so that the virtual assistant becomes smarter!!!

The expansion directions presented above, we do not guide with specific programs, but consider it as an idea for learners to create their own. Good luck with your virtual assistant.

4 Review questions

1. The process of realizing intelligence for virtual assistants, which of the following?
 - A. Input handling
 - B. Processing output
 - C. Algorithm processing
 - D. All correct
2. In order for the virtual assistant to work continuously until the user says "bye bye", what is the best command to execute?
 - A. if
 - B. while
 - C. for
 - D. All wrong
3. To implement intelligence for virtual assistants, which command can be used most effectively?
 - A. if
 - B. while
 - C. for
 - D. All wrong
4. To be able to retrieve the current system date, what library needs to be added to the program?
 - A. pyttsx3
 - B. date
 - C. speech_recognition.Recognizer()
 - D. All commands above
5. What are the limitations of the equality comparison statement between 2 strings when implementing intelligence for virtual assistants?
 - A. The probability of getting the correct command from the user will be low
 - B. When the user says "hello hello", the system will not understand if it just compares with "hello"
 - C. Very limited in practical implementation
 - D. All correct

Answer

1. C 2. B 3. A 4. B 5. D

CHAPTER 14



Python Interface Programming

pythonTM
Package
Index

1 Introduction

Like other high-level programming languages, Python gives us the opportunity to implement applications with a friendly interface. These applications are called Winform applications. The reason for such a name, because the interface of the application is really similar to the applications that we are using on the Windows operating system.

In this tutorial, we will design a simple application: User enters 2 numbers and presses the Sum button. The results will be printed on the program, as shown in the figure below:

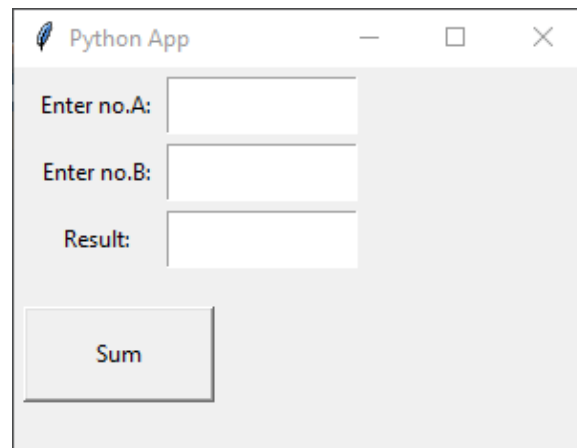


Figure 14.1: A GUI application on Python

Obviously, a program like the one shown in Figure 14.1 would be much more user-friendly, instead of just interacting from the console like traditional applications like Pascal or basic tutorials of Python. However, with such an application, we will need to be meticulous in the interface design, before we start programming. The special thing is that on Python, we will program to create interfaces, instead of dragging and dropping to design interfaces like other high-level languages. In this tutorial, we will present the simplest commands to design the application as shown in Figure 14.1.

2 Interface design

On Python, there is already support for the interface design library **tkinter**. The principle of interface programming is that we will create a window (Windows), then add interface objects (input frames, buttons, etc.). We will start the first step, which is to create the interface window, with the following program:

```
1 import tkinter as tk
2
3 window = tk.Tk()
4
5 window.title("Python App")
6 window.geometry("300x200")
7
8 window.mainloop()
```

Program 14.1: Create a window for the application

In Program 14.1, the first statement is used to add the interface design library **tkinter**. Since we will have to type a lot of tkinter keywords later on, we give it an abbreviation **tk**. Next, we create a window with the `tk.Tk()` statement, changing the window's parameters such as the title, the size of the window (width 300 and height 200 pixels). Finally, we will activate that window with the command **window.mainloop()**. When we run this program, we will have the following output:

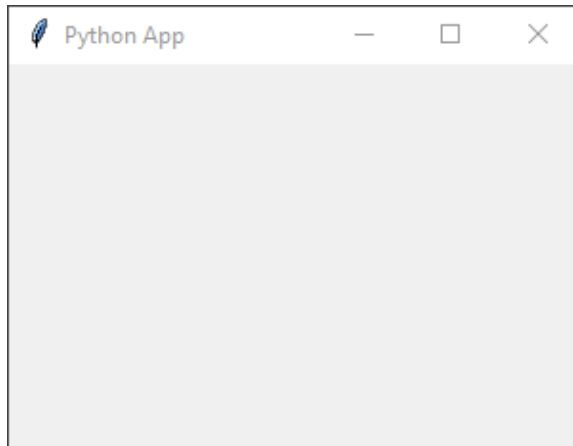


Figure 14.2: Application on Windows

The program output in Figure 14.2 is why the application is called a winform application, because it has the default structure of a window on Windows operating system: minimize, maximize, close the window as well as the title bar on the application.

3 Add an interface object to the window

After we have the standard window frame, we start adding interface objects. To be able to display the instruction to enter the number A, we need 2 interface objects: Label and Text. The Label object is only used for display, and the Text object is used for input. For each object, we have to use 2 commands: Create the object and position it in the window frame. The program for this section will be as follows:

```
1 import tkinter as tk
2
3 window = tk.Tk()
4
5 window.title("Python App")
6 window.geometry("300x200")
7
8 labelA = tk.Label(text="Enter no.A: ")
9 labelA.place(x=5, y=5, width = 80, height = 30)
10
11 txtA = tk.Text()
12 txtA.place(x = 80, y =5, width = 100, height = 30)
13
14 window.mainloop()
```

Program 14.2: Create window for application

In the Program 14.3, the parameters about x, y are the position of the object in the interface, width and height are the dimensions of the width and height of the object. Try changing it to see the effect of the change on the interface. With the above program, our output will be as follows:

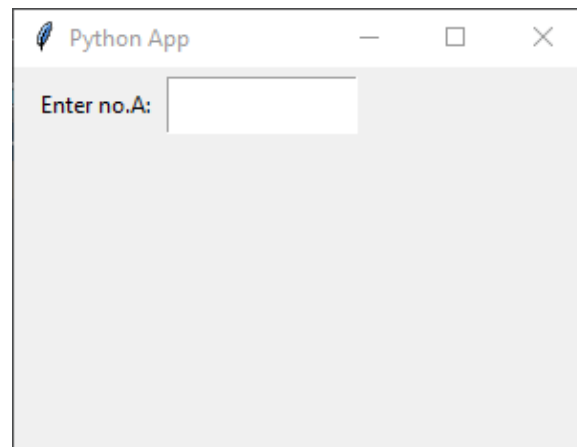


Figure 14.3: Add interface object to the application

Continue to do the same to add instructions for entering the number B, and the display of results. Notice that the height of the first line is 30, and the y-coordinate is 5. So the 2nd line must be at least 35. We set the y-coordinate of line 2 to 40 to get a moderate space. Likewise, line 3 y coordinates will be 80. The button will be added last. Our final program looks like this:

```
1 import tkinter as tk
2
3 window = tk.Tk()
4
5 window.title("Python App")
6 window.geometry("300x200")
7
8 labelA = tk.Label(text="Enter no.A: ")
9 labelA.place(x=5, y=5, width = 80, height = 30)
10
11 txtA = tk.Text()
12 txtA.place(x = 80, y =5, width = 100, height = 30)
13
14 labelB = tk.Label(text="Enter no.B: ")
15 labelB.place(x=5, y= 40, width = 80, height = 30)
16
17 txtB = tk.Text()
18 txtB.place(x = 80, y =40, width = 100, height = 30)
19
20 labelRes = tk.Label(text="Result: ")
21 labelRes.place(x=5, y= 75, width = 80, height = 30)
22
23 txtRes = tk.Text()
24 txtRes.place(x = 80, y =75, width = 100, height = 30)
```

```
25  
26 button = tk.Button(text = "Sum")  
27 button.place(x = 5, y = 125, width =100, height = 50)  
28  
29 window.mainloop()
```

Program 14.3: Complete interface design for the application

Finally, the interface of our application will be as shown in Figure 14.4.

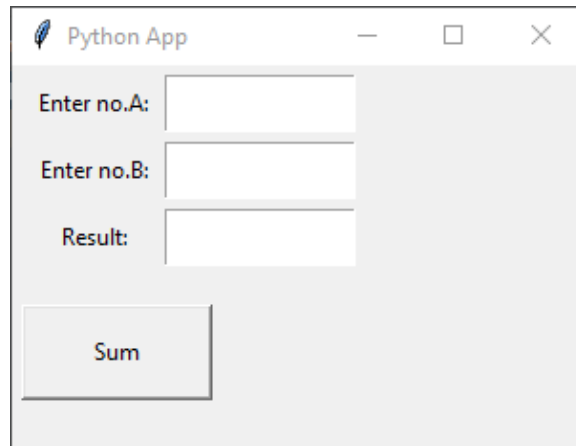


Figure 14.4: Complete interface design

4 Review questions

1. What is a Windows application called?
 - A. Winform
 - B. Console
 - C. Log
 - D. All correct
2. To program a Python interface on PyCharm you need to:
 - A. Drag and drop the interface object into the program
 - B. Write a program to generate the interface
 - C. Python does not support interface programming
 - D. All wrong
3. Normally, to print instructions on the application, which interface object will be used?
 - A. Label
 - B. Text
 - C. Button
 - D. Windows
4. Normally, for the user to enter information into the application, which interface object will be used?
 - A. Label
 - B. Text
 - C. Button
 - D. Windows
5. Normally, to receive user interactions, such as clicks, which interface object should be used?
 - A. Label
 - B. Text
 - C. Button
 - D. Windows
6. To adjust the position of the interface object on the application, which property will we interfere?
 - A. x coordinate and y coordinate
 - B. Parameter width and height
 - C. Both sentences above are correct
 - D. All wrong
7. To adjust the size of the interface object on the application, which property will we interfere?
 - A. x coordinate and y coordinate
 - B. Parameter width and height
 - C. Both sentences above are correct
 - D. All wrong

Answer

1. A 2. B 3. A 4. B 5. C 6. A 7. B

CHAPTER 15



Create button handler

pythonTM
Package
Index

1 Introduction

After designing the interface, the next step is to create a function that handles when the user clicks the Sum button. The button is almost the main object of user interaction on the application running on the computer. The function that we are about to create will be automatically called when the user clicks the button.

2 Declaring the handler function

Firstly, we will declare a function, named `onClick` on Python. This function only prints a short message at first, like the example below:

```
1 def onClick():  
2     print("Click Button")
```

Program 15.1: Declare function `onClick`

Next, we need to bind the Sum button click event to this `onClick` function. In the command to initialize the button in the previous tutorial, we edit it as follows:

```
1 button = tk.Button(text = "Sum", command = onClick)
```

Program 15.2: Declare function `onClick`

It is important to note that we must declare the `onClick` function before assigning it to the button click event. Our complete program will be as follows:

```
1 import tkinter as tk  
2  
3 def onClick():  
4     print("Click Button")  
5  
6 window = tk.Tk()  
7 window.title("Python App")  
8  
9 labelA = tk.Label(text="Enter no.A: ")  
10 labelA.place(x=5, y=5, width = 80, height = 30)  
11  
12 txtA = tk.Text()  
13 txtA.place(x = 80, y = 5, width = 100, height = 30)  
14  
15 labelB = tk.Label(text="Enter no.B: ")  
16 labelB.place(x=5, y= 40, width = 80, height = 30)  
17  
18 txtB = tk.Text()  
19 txtB.place(x = 80, y =40, width = 100, height = 30)  
20  
21 labelRes = tk.Label(text="Result: ")  
22 labelRes.place(x=5, y= 75, width = 80, height = 30)  
23  
24 txtRes = tk.Text()
```

```

25 txtRes.place(x = 80, y =75, width = 100, height = 30)
26
27 button = tk.Button(text = "Sum", command = onClick)
28 button.place(x = 5, y = 125, width =100, height = 50)
29
30 window.geometry("300x200")
31 window.mainloop()

```

Program 15.3: Complete program with button click event

3 Implement the handler function for the button

To implement this function, we need to get information from 2 objects txtA and txtB. The data from these two objects are of string type, so we have to convert to the desired data type, in this case numeric data. At this step, things have become much simpler, we calculate the sum of 2 number . And finally output the results to txtRes. The program instructions for this event handler function are as follows:

```

1 def onClick():
2     print("Click Button")
3     a = int(txtA.get("1.0", "end"))
4     b = int(txtB.get("1.0", "end"))
5
6     res = a + b
7
8     txtRes.delete(1.0)
9     txtRes.insert(1.0, res)

```

Program 15.4: Complete the program with a click button event

The program 15.4 is an example of basic processing operations from getting data from Text, processing and outputting the results back on a Text object. Before exporting new data, we need to delete the old data. The number "1.0" in the program means row 1, first position (position 0). For the get statement, it means reading the data at row 1, position 1 to the end (keyword "end"). For the delete statement, it means deleting the entire row 1. The insert statement adds new data in row 1, from the first position.

Now we can run the program, it is a simulation for calculating the sum of 2 numbers. The program will stay running until click the close window icon in the upper right corner.

4 Exercise

1. Write a program to calculate the solution of a first-order equation, with the coefficients a and b entered from the interface.
2. Write a program to solve quadratic equations, with the coefficients a, b and c entered from the interface.

5 Review questions

1. To create an event handler for a button click, what do we need to do?
 - A. Defines a function, for example `def onClick()`
 - B. Declare it in the button initialization command with `command = onClick`
 - C. Do both steps above
 - D. All wrong
2. Which statement is true about the button click event handler:
 - A. It must be declared before the button is pressed
 - B. Declared anywhere in the program
 - C. No need to declare, default Python calls
 - D. All wrong
3. What are the steps to implement button event handler in Python?
 - A. Get data from input interface
 - B. Data Processing
 - C. Output data to the interface
 - D. All correct
4. What is the meaning of `txtA.get("1.0")` statement?
 - A. Read all data of txtA
 - B. Export data to txtA from the first position
 - C. Python does not support this command
 - D. All wrong
5. What is the meaning of the statement `txtA.insert("1.0")`?
 - A. Read all data of txtA
 - B. Output more data to txtA from the first position
 - C. Python does not support this command
 - D. All wrong
6. What is the meaning of the statement `txtA.delete("1.0")`?
 - A. Delete all data of txtA
 - B. Delete txtA's data from row 1, first position to the end
 - C. Both of the above statements are correct
 - D. All wrong

Answer

1. A 2. A 3. D 4. A 5. B 6. C

CHAPTER 16



Interface error handling

pythonTM
Package
Index

1 Introduction

Through the project of implementing an application with an interface on Windows in Python language, we can see that our processing has only 1 command line, which is the sum of 2 numbers. The rest, all command lines are mainly used for interface design, get data from the Text on the interface and output the results back to the interface.

However, like many other applications, error handling for input data must always be taken into account, to make the application more realistic and easier to interact with the user. In this application, we are defaulting to an integer. However **user can enter a string** and our program will throw an error.

Besides using the **try except** statement, this tutorial will show us how to display an error message box (message box), as shown in the figure below:

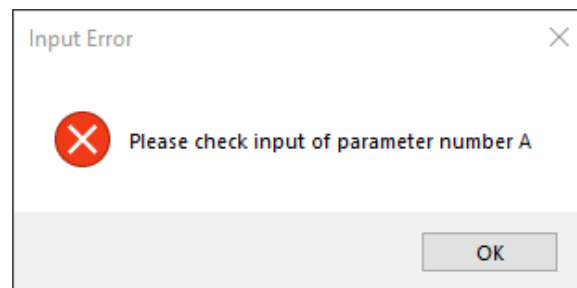


Figure 16.1: Error message from the program

Notifications like the above will make our program much more user-friendly, instead of just printing out the words on the Console interface.

2 Handle input errors

In principle, any process that receives data from the user should be checked for errors. Here, we expect the user to enter a number, so we write a statement that converts it to an integer. An extremely powerful statement that helps us in error handling is **try except**. Now, in the **except** section, we will display an additional error message.

First, we need to add the library for displaying the error message, with the import statement at the beginning of the program like this:

```
1 import tkinter as tk
2 from tkinter import messagebox
```

Program 16.1: Declare function onClick

The program for our button click event handler will look like this:

```
1
2 def onClick():
3
4     print("Click Button")
```

```

5     try:
6         a = int(txtA.get("1.0"))
7     except:
8         messagebox.showerror("Input Error", "Please check
input of parameter number A")
9         return
10    try:
11        b = int(txtB.get("1.0"))
12    except:
13        messagebox.showerror("Input Error", "Please check
input of parameter number B")
14        return
15
16    c = a + b
17
18    txtRes.delete(1.0)
19    txtRes.insert(1.0, c)

```

Program 16.2: Catch errors and notify when entering data from users

Please note how to use the **return** statement: When an input error occurs, we notify the user and at the same time have to stop the processing below. The return statement is used to exit the current event handler function, because simply, if the input data is not valid, we have not processed it.

Operations to restore the program's interface, such as When the user enters the wrong data, we notify but also delete the wrong input, it will be considered as an enhancement for the reader to self-improve. You can use Vietnamese in the announcement sentence. In this course, due to the font for Python language, we cannot use Vietnamese with accents.

At this point, we can already build larger applications yourself, such as the Calculator software that we are using on the computer, or the Math support software for middle and high school students.

3 Review questions

1. Why do we have to handle input errors?
 - A. The program will run incorrectly if the data is invalid
 - B. Python cannot perform the processing if the input data is invalid
 - C. The program will exit if the data is invalid
 - D. All correct
2. An effective tool for reporting errors to users is:
 - A. Show text on console screen
 - B. Show message box on app
 - C. No need to do anything, Python handles it by itself
 - D. All correct
3. What are the additional libraries to use the message box (messageBox)?
 - A. from tkinter import messagebox
 - B. from messagebox import tkinter
 - C. import messagebox
 - D. All correct
4. Which of the following statements is true for messageBox?
 - A. Message box can display Vietnamese
 - B. Close the message box by clicking the OK button
 - C. Can change the title of the message box
 - D. All correct
5. What is the meaning of the return statement?
 - A. Close current application
 - B. Exits the current function's thread
 - C. All correct
 - D. All wrong

Answer

1. D 2. B 3. A 3. D 4. D 5. B