Teaching materials

15016406: COMPUTER PROGRAMMING

By

Patcharin Kamsing (Ph.D.)

Department of Aeronautical Engineering and

Commercial Pilot

International Academy of Aviation Industry

King Mongkut's Institute of Technology Ladkrabang

Abstract

Nowadays, information technology plays a vital role in daily human life. Computer programming becomes a common thing for the new generation of engineers. These teaching materials are relevant to the fundamentals of python programming, which includes examples for readers to practice. Also, it gives you the knowledge of using a host for software development version control using Git like GitHub to allow readers to have more understanding about the software development ecosystem.

The author hopes that this teaching material will be an advantage for all readers. The author also wants to thank all the staff and faculty members for their contribution, as well as all the students assist in pushing this teaching material into completion.

Contents

Course Syllabus

International Academy of Aviation Industry

King Mongkut's Institute of Technology Ladkrabang

Aeronautical Engineering and Commercial Pilot

1/2562

---

1. **Course Number**: 15016406
2. **Course Title**: COMPUTER PROGRAMMING
3. **Course Credit**: 3 (2-2-5)
4. **Instructor**: Patcharin Kamsing (Ph.D.), E-mail: patcharin.ka@kmitl.ac.th
5. **Condition**

   5.1 Prerequisite: None

   5.2 Corequisite: None

   5.3 Concurrent: None

   5.4 Status (Required/Elective): Required
6. **Hours/Week**: 4 Hours
7. **Course Description**:

   แนวคิดของระบบคอมพิวเตอร์องค์ประกอบของระบบคอมพิวเตอร์การปฏิสัมพันธ์ ระหว่างฮาร์ดแวร์และซอฟต์แวร์ แนวคิดของการประมวลผลข้อมูลแบบอิเล็กทรอนิกส์การออกแบบและขั้นตอนการพัฒนาโปรแกรมการเขียนโปรแกรมด้วยภาษาคอมพิวเตอร์ระดับสูง

   Computer concept, computer components, hardware and software interaction, EDP concepts, program design, and development methodology, high-level language programming.
8. **Course Outline**

   8.1 Course Learning Outcome (CLO)

   CLO-1: To be able to develop Python programs.

   CLO-2: Illustrate the understanding of computer programming language concepts.

   CLO-3: To have the ability to design and develop Computer programs, analyze, and interprets the concept of pointers, declarations, initialization, operations on pointers, and their usage.

CLO-4: To have the ability to define data types and use them in simple data processing applications, also the student must be able to use the concept of array of structures. A student must be able to define union and enumeration user-defined data types.

CLO-5: Develop confidence for self-education and the ability for life-long learning needed for Computer language.

8.2 Learning Contents

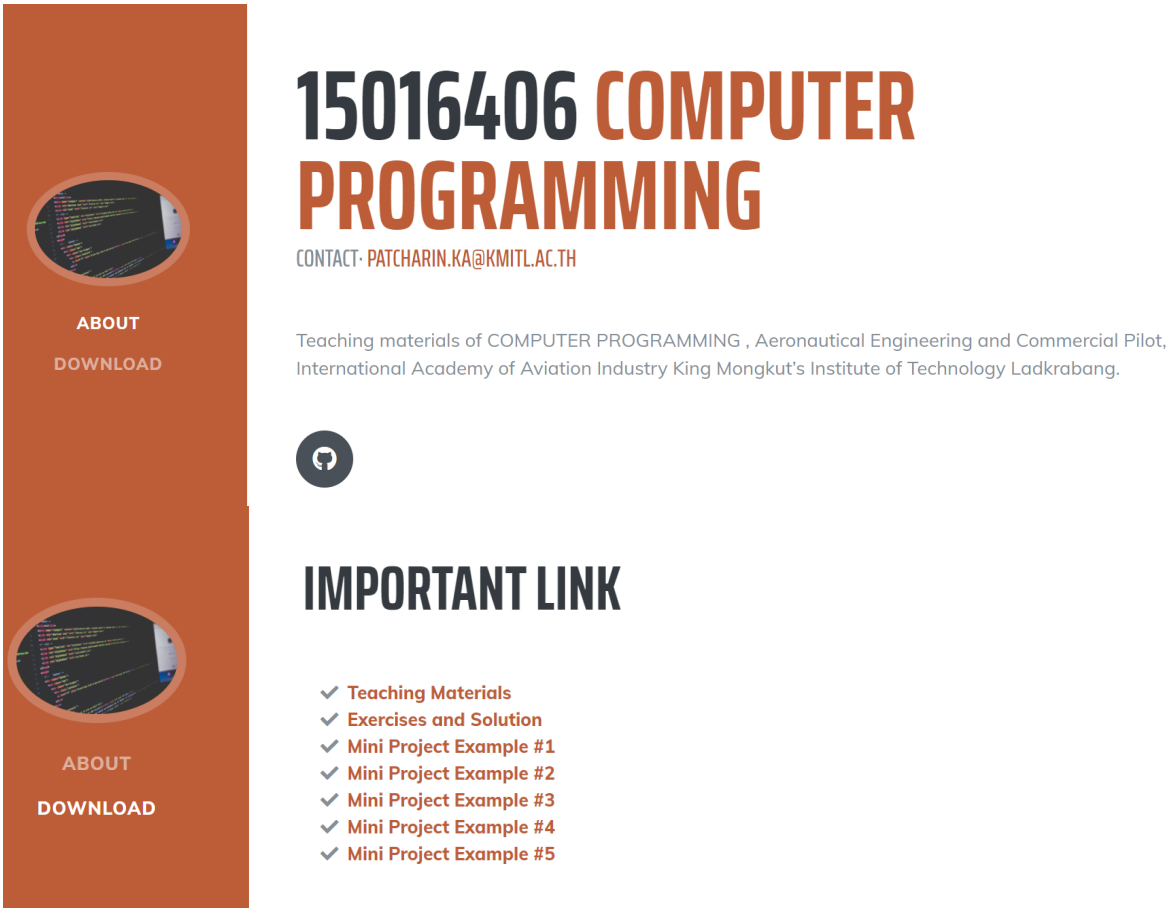
Week	Learning content	Remark
1	Chapter 1 Important of Programming - Installing and Using Python	Explanation/Laboratory
2	Chapter 1 Important of Programming - Installing and Using GitHub	Explanation/Laboratory
3	Chapter 2 Variables, expressions, and statements	Explanation/Laboratory
4	Chapter 3 Conditionals	Explanation/Laboratory
5	Chapter 3 Conditionals	Explanation/Report
6	Chapter 4 Loop and Iteration (for loop)	Explanation/Laboratory
7	Chapter 4 Loop and Iteration (While loop)	Explanation/Report
8	Midterm Examination	-
9	Chapter 5 Arrays	Explanation/Laboratory
10	Chapter 5 Arrays	Explanation/Laboratory
11	Chapter 6 Functions	Explanation/Laboratory
12	Chapter 7 Files Handling	Explanation/Laboratory
13	Mini-project Python programming	Explanation/mini project
14	Mini-project Python programming	Explanation/mini project
15	Final Examination	-

8.3 Learning Method

- Explanation
- Using a computer for coding a program

8.4 Learning Media

- Kenneth A. Lambert. 2018. Fundamentals of Python: First Programs (2nd. ed.). Course Technology Press, Boston, MA, USA.
- Internet
- Website: https://patcharinka.github.io/15016406/



8.5 Evaluation

- Midterm Exam 30%
- Final Exam 40%
- Mini-project 10%
- In-class activities (including Homework, class attendance, etc.) 20%

9. Reading List and Supplementary Texts

Kenneth A. Lambert. 2018. Fundamentals of Python: First Programs (2nd. ed.). Course Technology Press, Boston, MA, USA.

10. Teacher Evaluation

10.1 Method for evaluation

- Post-Test
- Learning outcome

10.2 An improvement from the last teacher evaluation

None

11. Recommending to students about this course and other topics

- Describe the date and time of this course, date, and time to have a consult with the teacher face to face. Explain the method to submit homework and mini-project and explain the learning plan and condition to give a score in the first class.
- Allow a student to send an email to get a suggestion from the teacher in case, not in-office hours.

12. The teacher gives this Course Syllabus to the student in the first class.

- In the first day (.....), the teacher already gives the course syllabus to the student, as shown in the attached document.

Chapter 1 Important of Programming

Computer programming considers being a vital thing to create and produce new things or innovation technology in recent days. The advantage of computer programming has both in term of technical and economic. For the technical, a developer can use programming as they prefer language to do a simulation about their experiment; trial and error can be done many times on a computer. This also allows the developer to adjust the parameters and modify their method to obtain the suitable way for focused application. In the economy side, a developer can save the number of resources such as man or money to invest in the experiment, which could succeed or fail as several factors.

The important thing that needs to have an efficiency programming skill is logical thinking and practicing as consistently. The logical thinking is the skill that can develop by asking ourselves about the next stage of concentrate situation, which affects output previously and may have an impact from current input as in Figure 1. Besides, practice thinking as logical as continuously resulting in developing programming skills so much.

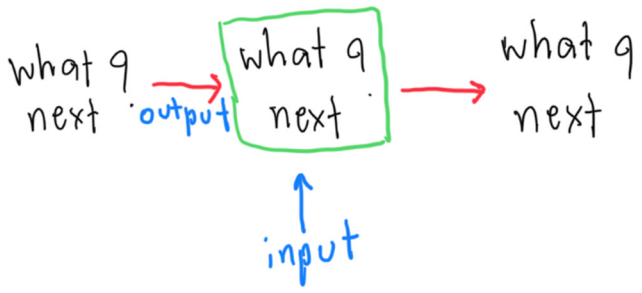


Figure 1 Thinking process to develop logical skill

Computers consist of both hardware and software, and all of them have been progressing compared to the past. Hardware includes a chip, RAM, or event power supply develop to have more robustness in various environments and have the ability to working continuously in an extended period. There is much software to serve humanity as they propose. For example, to type a report in Microsoft Word. However, software development is still required to support daily human activity, such as software for detecting the type of object or product which can deploy in a shopping store. Moreover, it is a trend to combine hardware

and software to accomplish the human propose, such as software for detecting heart rate during exercise. Therefore, the Internet of Things (IoT) became attractive right now.

Computer programming is a skill that support developer to develop their software. Programming Language is the language to convey the command to order a computer to have processing. Although different language has different syntax, the fundamental is still similar. A programmer should know the basics of the focus language, such as how to initialize parameters, type of parameter, the syntax for operation, etc. In 2019, TIOBE was an organization that has created an index for programming languages report the top programming language and its changing rate as Figure 2 and Figure 3. Python Language has a changing rate at +3.03%, which is the highest and attractive to study for a new programmer, and Python programming is not so difficult when compared to other computer languages.

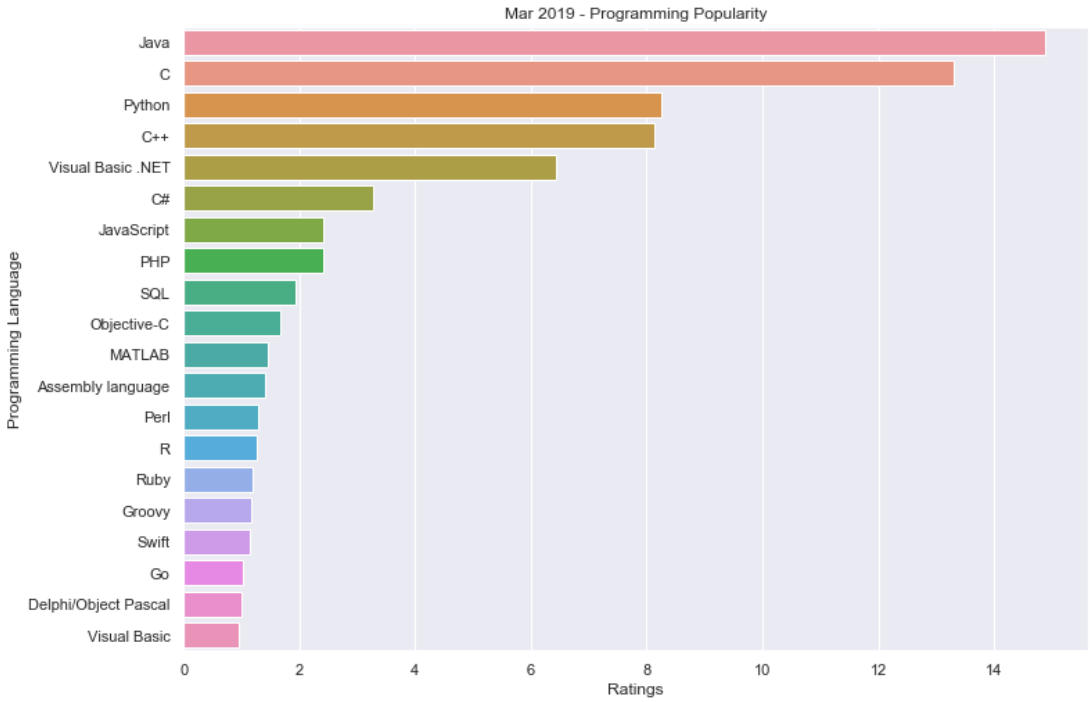


Figure 2 Top programming language report by TIOBE (https://towardsdatascience.com/visualize-programming-language-popularity-using-tiobeindexpy-f82c5a96400d)

Programming Language	Ratings	Change
Java	16.028%	-0.85%
C	15.154%	+0.19%
Python	10.020%	+3.03%
C++	6.057%	-1.41%
C#	3.842%	+0.30%
Visual Basic .NET	3.695%	-1.07%
JavaScript	2.258%	-0.15%
PHP	2.075%	-0.85%
Objective-C	1.690%	+0.33%
SQL	1.625%	-0.69%
Ruby	1.316%	+0.13%
MATLAB	1.274%	-0.09%
Groovy	1.225%	+1.04%
Delphi/Object Pascal	1.194%	-0.18%
Assembly language	1.114%	-0.30%
Visual Basic	1.025%	+0.10%
Go	0.973%	-0.02%
Swift	0.890%	-0.49%
Perl	0.860%	-0.31%
R	0.822%	-0.14%

Figure 3 Top programming language and its change rate report by TIOBE (https://stackify.com/popular-programming-languages-2018/)

1.1 Installing and Using Python

This teaching material is for basic Python programming, which mainly concentrates on Python desktop programming. Therefore, the student needs to install a Python environment

for themselves. Python installation has two parts, namely, 1) Python Editor and 2) Python executes(compiler).

1.1.1 Python IDEs and Code Editor

Python is the high-level programming language that was developed in 1991 by Guido van Rossum. Python has been used for several applications such as web development, development of software, math, scripting, and artificial intelligence. Python can process on multiple platforms like Windows, Mac, Linux, Raspberry Pi, etc.

Figure 4 Guido van Rossum(https://en.wikipedia.org/wiki/Guido_van_Rossum)

Before learning how to program by Python language, we should know about Python IDE and code editor, which it benefits for writing a program.

IDE stands for Integrated Development Environment. IDE is generally packing all equipment for a developer to write and test a program. Python IDE has a library or a secure method typically to install packages. Python IDE also helps to write a Python program. For example, a suggestion for command. However, some developers also prefer Code editors. Code Editor is a text editor such as Notepad, where a developer can write the code for developing any software. The code editor also allows the developer to save small text files for the code. Code editor can operate with a small size of software and need to comply with the source code by command line manually.

Figure 5 Example program of Python IDE

There is some code editor program that the Python developers world-wide are preferred. Code editor may be integrated or stand-alone application and speedy processing different from IDE that frequently needs to update the environment.

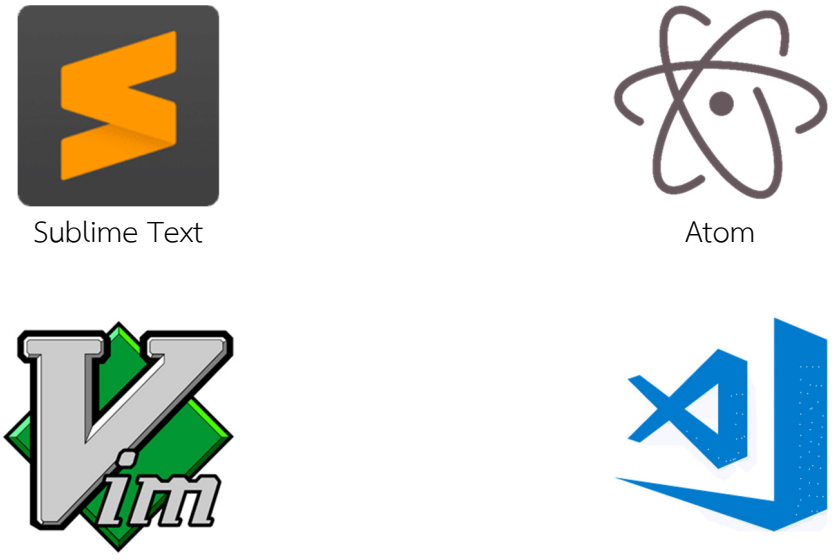


Figure 6 Example program of code editor

In conclusion, the code editor is a program or platform that allows a developer only to modify code. Whereas IDE is an integrated environment and has many features, namely coding, compiling, debugging, executing, autocomplete, libraries, in one place for the developer.

PyCharm is a popular Python IDE that was created by Jet Brains. PyCharm is a specialized IDE for Python. The programmer can write and maintain code with smart assistance for a developer. Also, PyCharm can take care of the routine works. The following is the best feature, pros, and cons of PyCharm.

Best Features:

1. It comes with an intelligent code editor, smart code navigation

2.PyCharm is integrated with features like debugging, testing, profiling, deployments, remote development, and tools of the database.

3. PyCharm provides support to python web development frameworks, JavaScript, HTML, CSS, Angular JS, and Live edit features.

4. It has a robust integration with IPython Notebook, python console, and scientific stack.

Pros:

1. It provides a smart platform to the developers who help them when it comes to auto code completion, error detection, quick fixing, etc.

2. It provides multiple framework support by increasing a lot of cost-saving factors.

3. It supports a rich feature like cross-platform development so that the developers can write a script on different platforms as well.

4.PyCharm also comes with an excellent feature of the customizable interface, which in turn increases productivity.

Cons:

1.PyCharm is an expensive tool while considering the features and the tools it provides to the client. However, there is a free version for an ordinary developer (community version).

2. The initial installation is complicated and may hang up in between sometimes but can uninstall and install again to complete it.

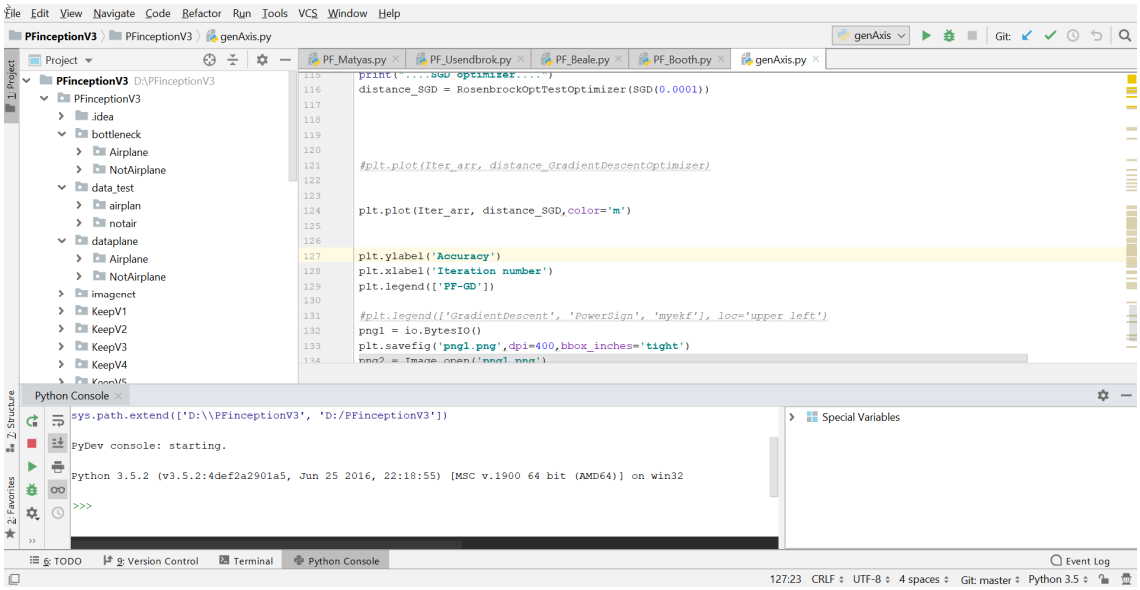


Figure 7 Screen of PyCharm IDE

The official website for downloading PyCharm is https://www.jetbrains.com/pycharm/. There are many versions suitable for a different types of users [1].

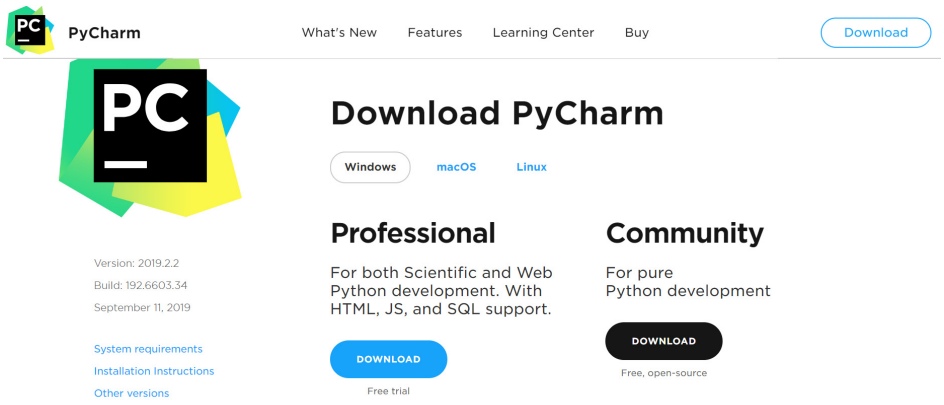


Figure 8 Page to download PyCharm IDE (community version)

For a beginner, the Community version is enough for learning Python Programming. PyCharm has three main areas, namely directory area, coding area, and result area, as Figure 9. The directory area will show the directory or folder that you want to see. The coding area is an area that you can be coding with Python syntax. The resulting area can demonstrate the result from running a program or debug a program.

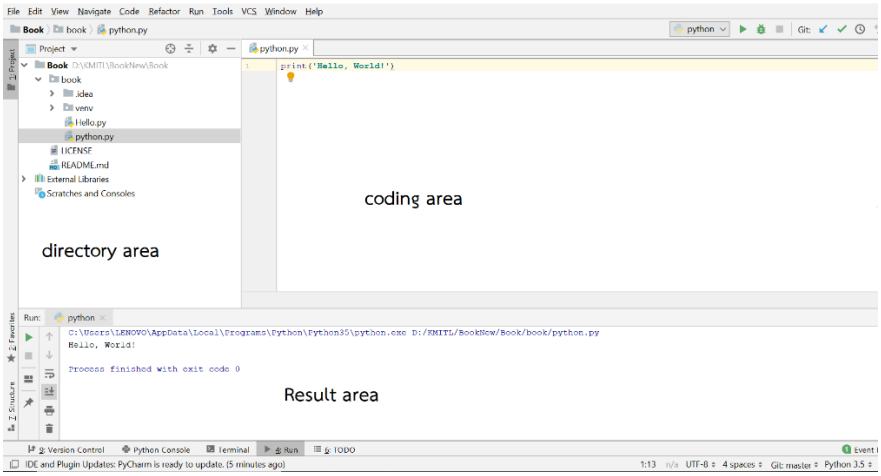


Figure 9 Three main areas of PyCharm

1.1.2 Python executes(compiler)

Same as another computer programming, Python has various versions, and different version has different feature and sometimes have different spalling of syntax. For all computer programming, Programmers must learn and remember the syntax to increase coding performance [2]. In Python programming, Python 2 and 3 are widely used depend on a different application. For example, deep learning applications (a kind of artificially intelligent,

AI) mostly implement with Python 3 because of a supporting library. Therefore, the first thing before writing a Python program, we should consider the feature of each Python version to ensure that all libraries that we need are included in that version. https://www.python.org/ is a website to download Python compiler.

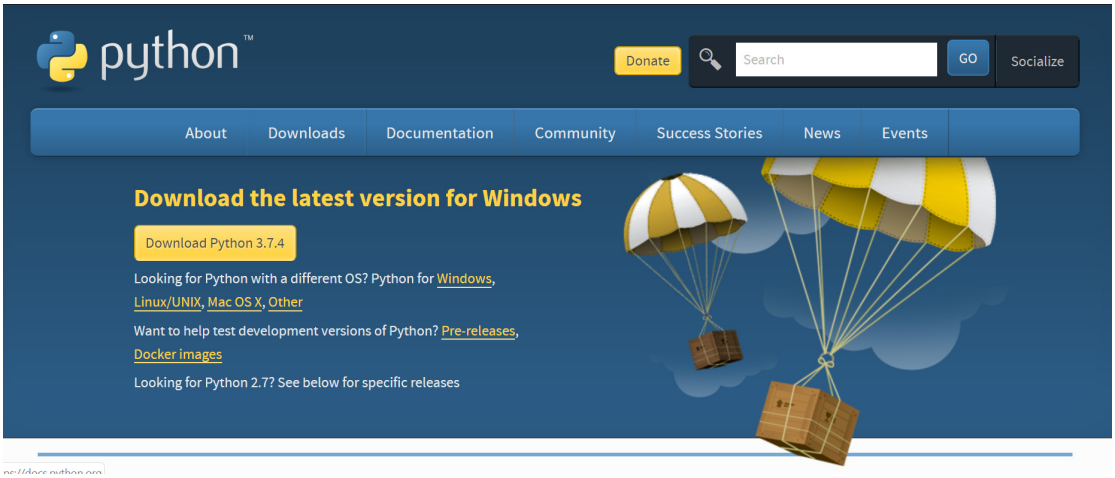


Figure 10 website https://www.python.org/

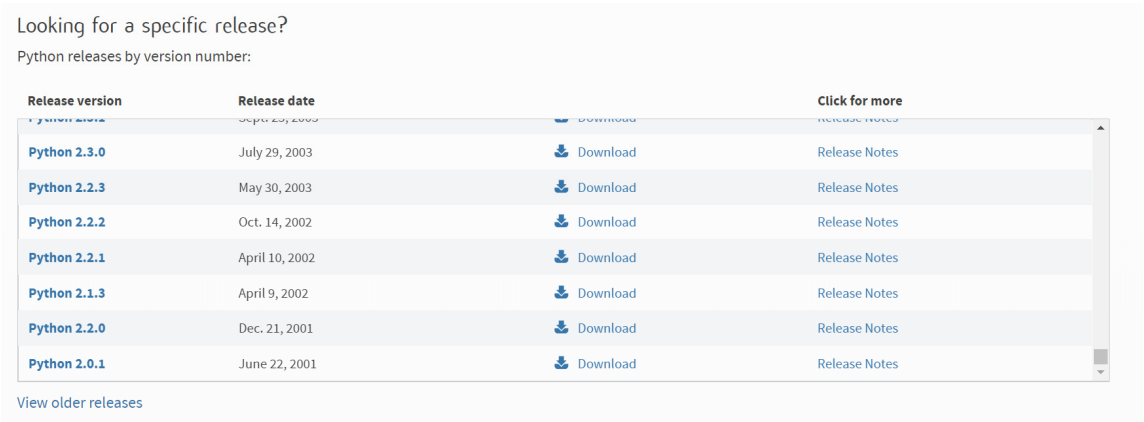


Figure 11 Different Python version

1.2 The first program

After we install Python IDE and Python compiler, the next task is trying to write a program to print the sentence “Hello World” as the following code and suggestion. First, we need to new Project (Figure 12), or it can compare to the new folder as we familiar with.

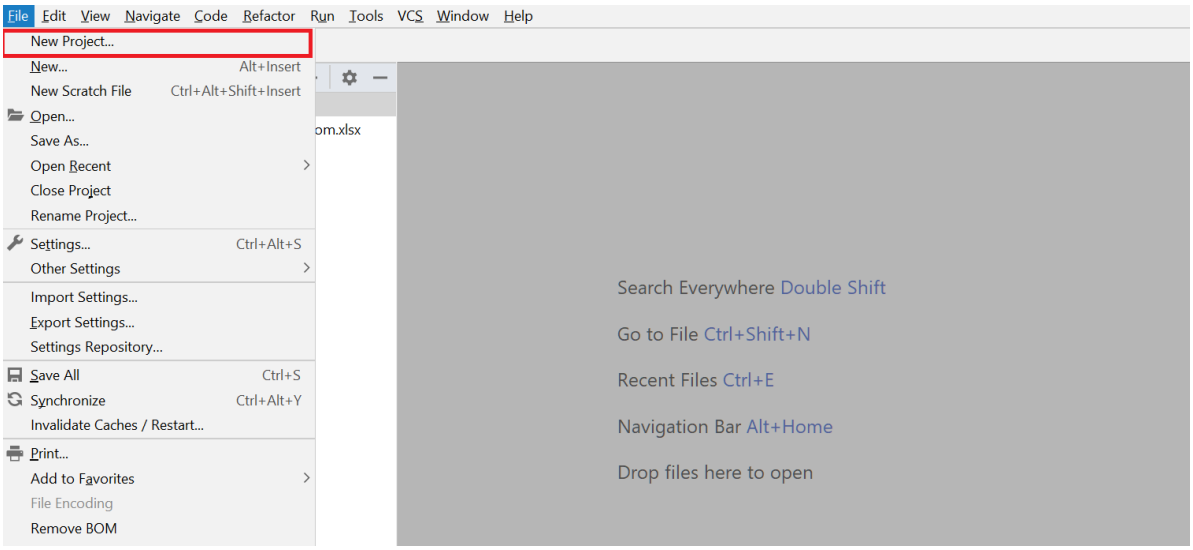


Figure 12 Method to the new project

Set location for the new project, then selects an interpreter by choosing the new environment or existing interpreter and create in the last step.

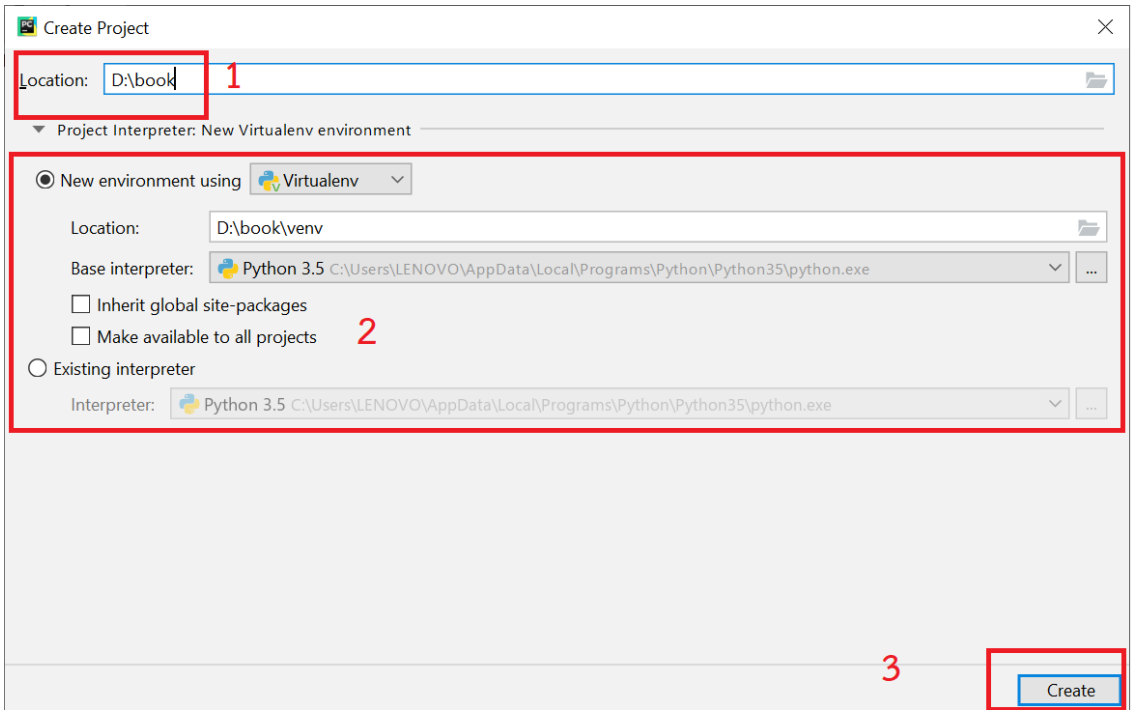


Figure 13 Setting the directory and environment for the project

Now, we have created a project (compare to a new folder as usual), the next step, we need to a new file which in here is new Python File as Figure 14.

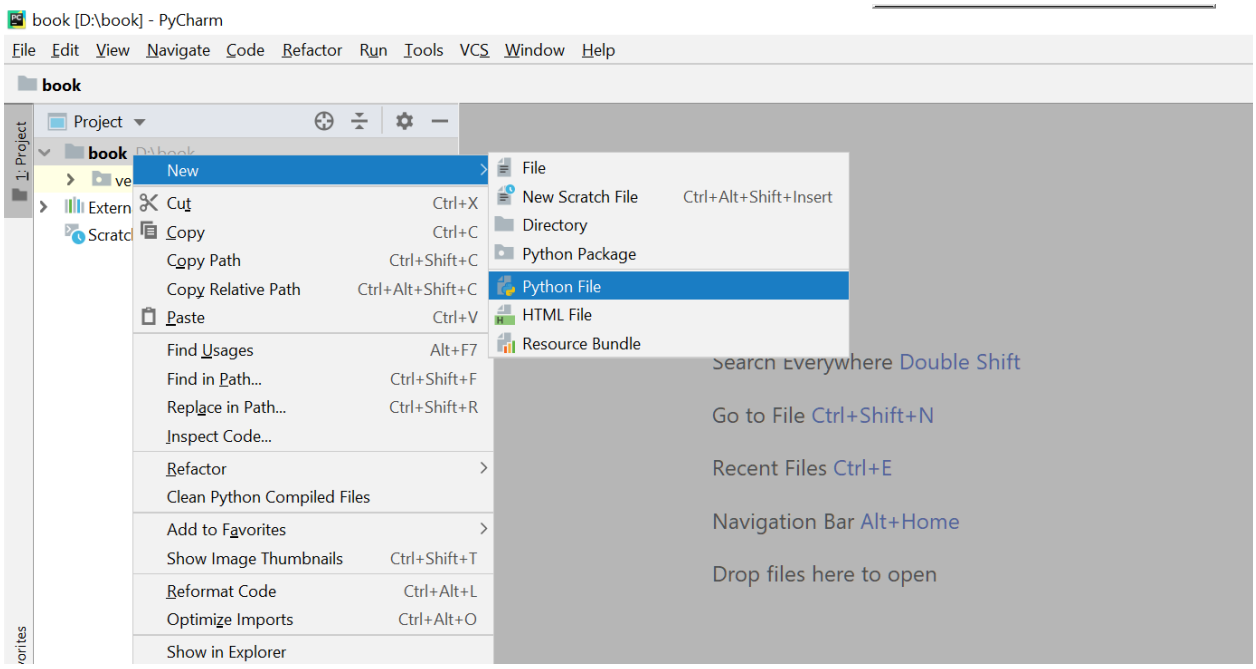


Figure 14 Method to new Python file

Type the file name and then hit "OK."

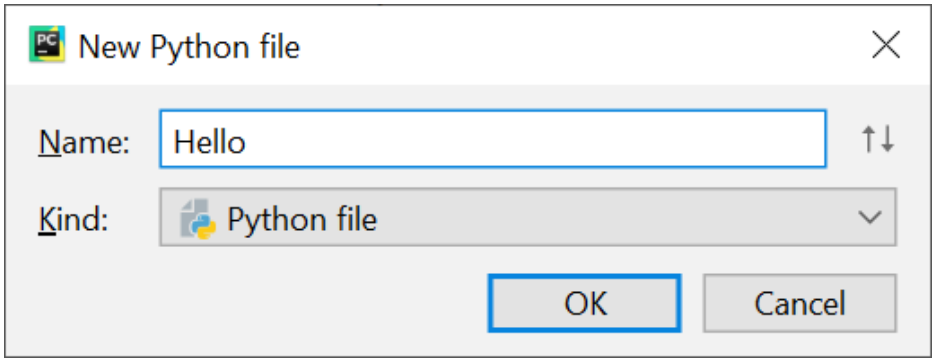


Figure 15 Set the name of the Python file.

In the coding area, it will have a cursor to allow us to begin to write a program. Type the first syntax by ***print("Hello World")*** as an image in Figure 16.

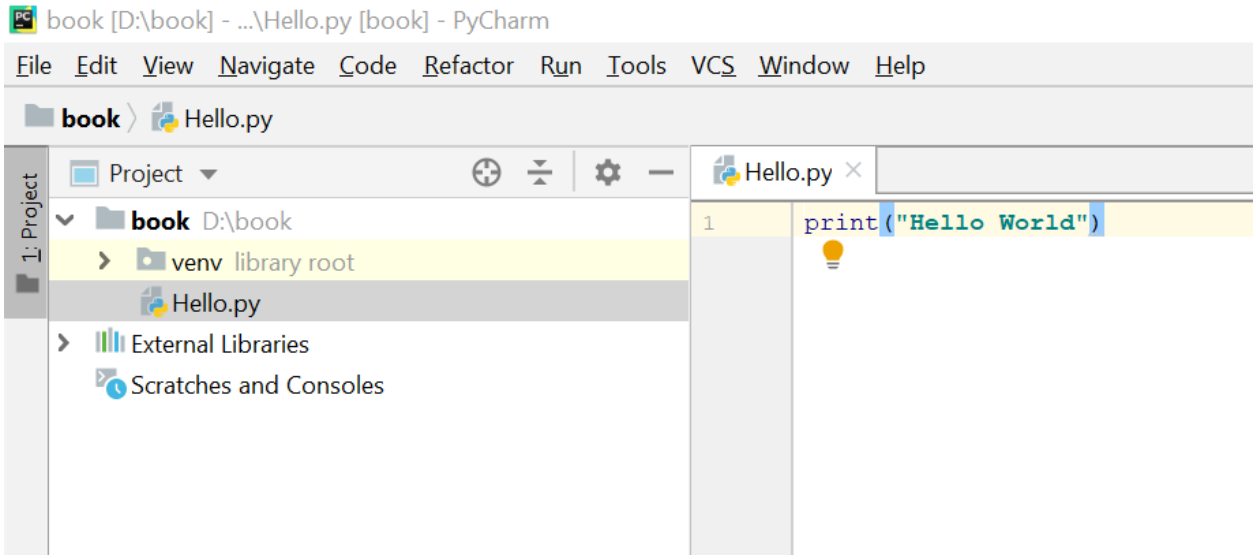


Figure 16 First Python program (1)

To see a result of the program, we have to run by right click on the screen and select Run "youfilename."

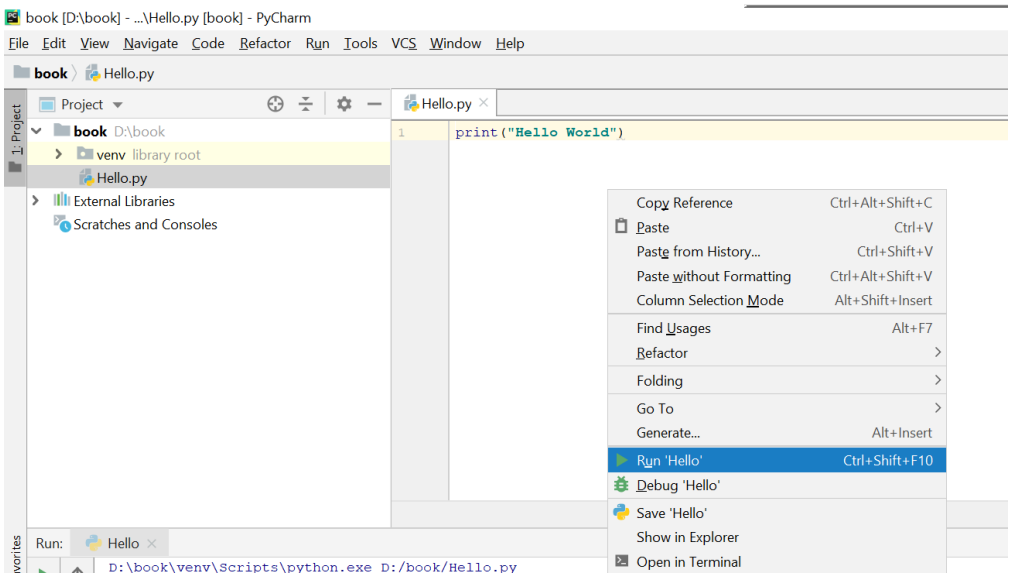


Figure 17 First Python program (2)

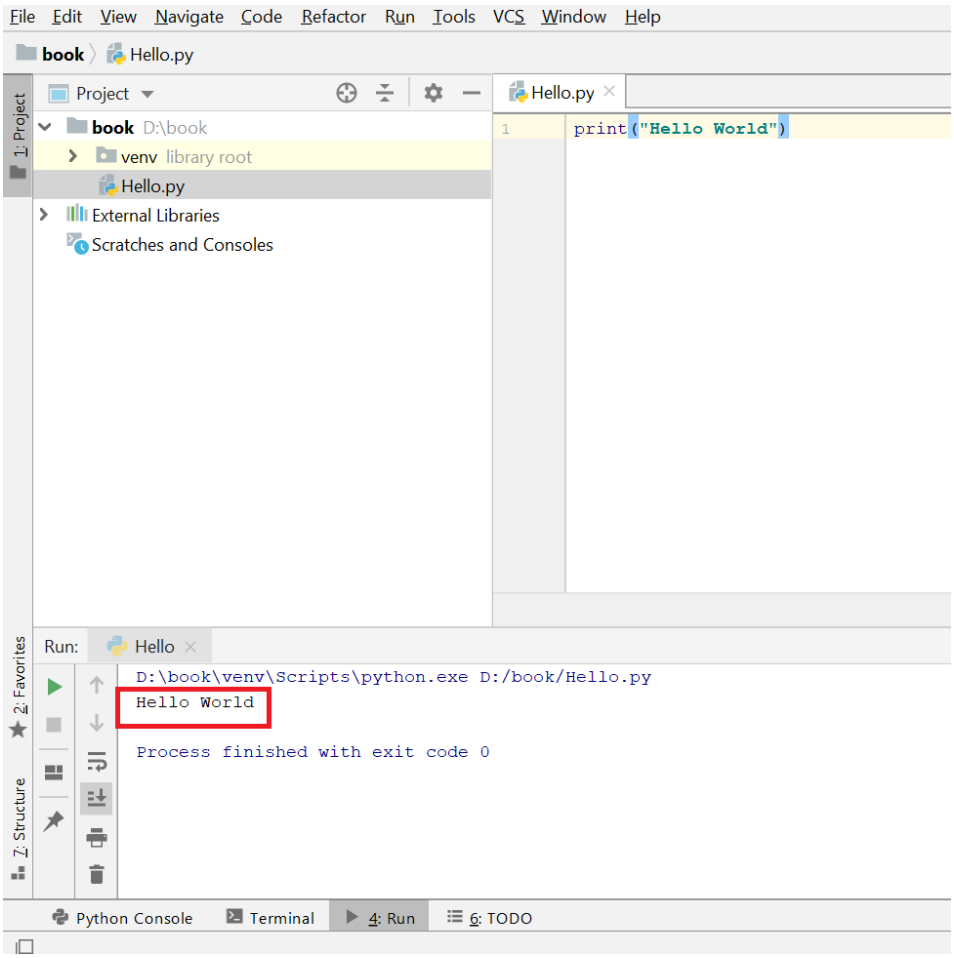


Figure 18 First Python program (3)

The result will show as follow with the word "Hello World," as in Figure 18.

1.3 Installing and Using GitHub

Now we finish the first program, and the next step is we have to deal with source code that has been write in each version or each class. The major problem of a programmer is about "version control" and "code explanation." GitHub is a platform that allows programmers to upload source code into the cloud. Besides, GitHub can detect or track the changing of source code that updates in each time that we program, resulting in we can roll back or update to be the last version in a natural way. Moreover, GitHub is enabling the programmer and their team to share the knowledge and progress of each other by adding everyone to be collaborators. That is simply a way to present their progress. Therefore, the limitation of the developer about version control and code explanation can eliminate. We

can access the website https://github.com/ as Figure 19 and also download GitHub Desktop as Figure 20 to synchronous working between On-Cloud and On-Premise.

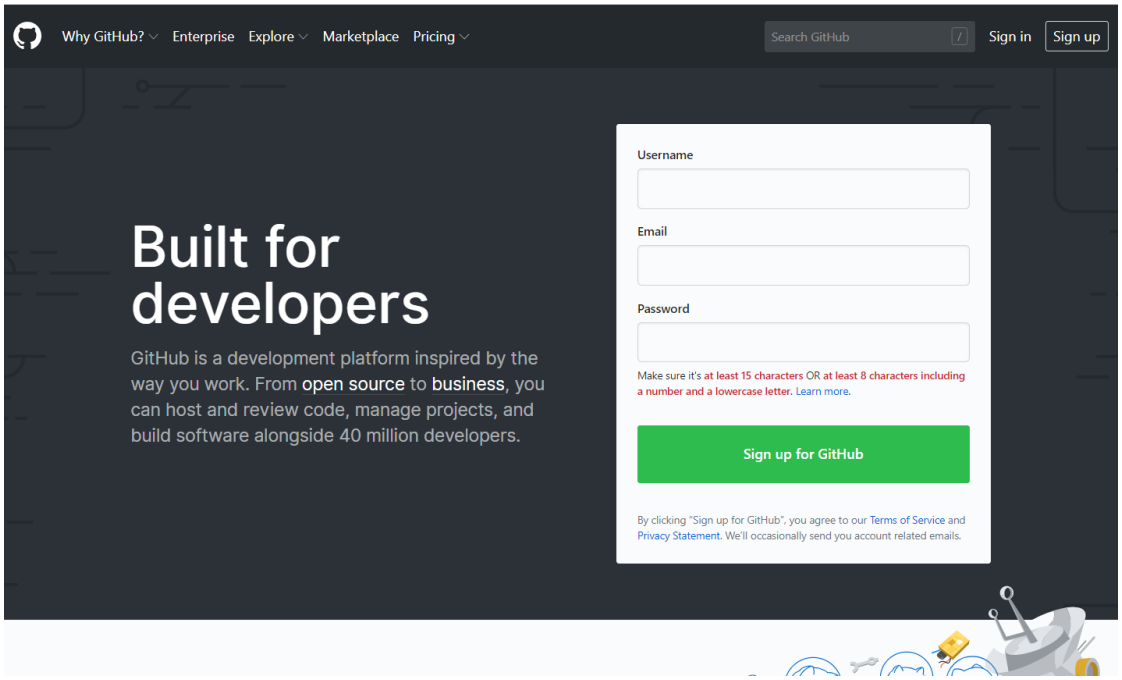


Figure 19 Sign up page for GitHub website

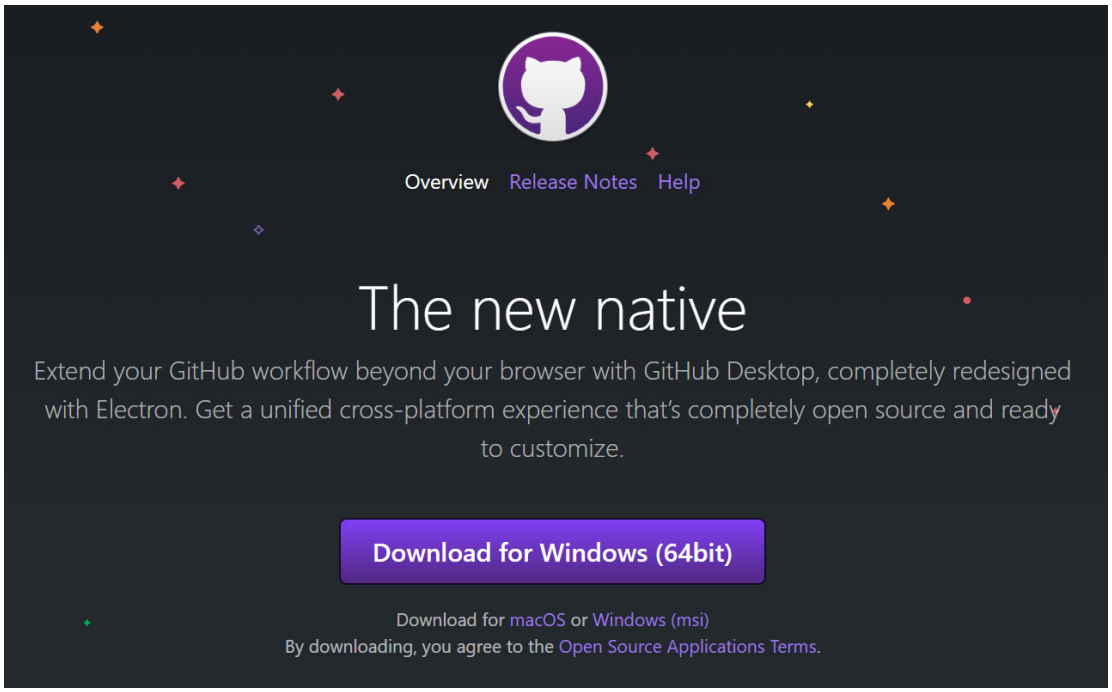


Figure 20 Download for GitHub Desktop

1.3.1 GitHub Introduction

GitHub is a provider hosting software development version control by using Git. GitHub also offers many features for software development, such as the distributed version control or source management (SCM). Also, it has a function for collaboration features such as bug tracking, feature requests, task management, and wikis for every project [3]. This platform overcomes the problem of working as a group of the developer. It has to update with a team member about the progress of every member.

Previously, GitHub allows developers for free accounts only in case of a public project. However, GitHub presents a private repository for all plans, including a free account in January 2019. As of May 2019, GitHub reports having over 37 million users and more than 100 million repositories (including at least 28 million public repositories), making it the largest host of source code in the world.

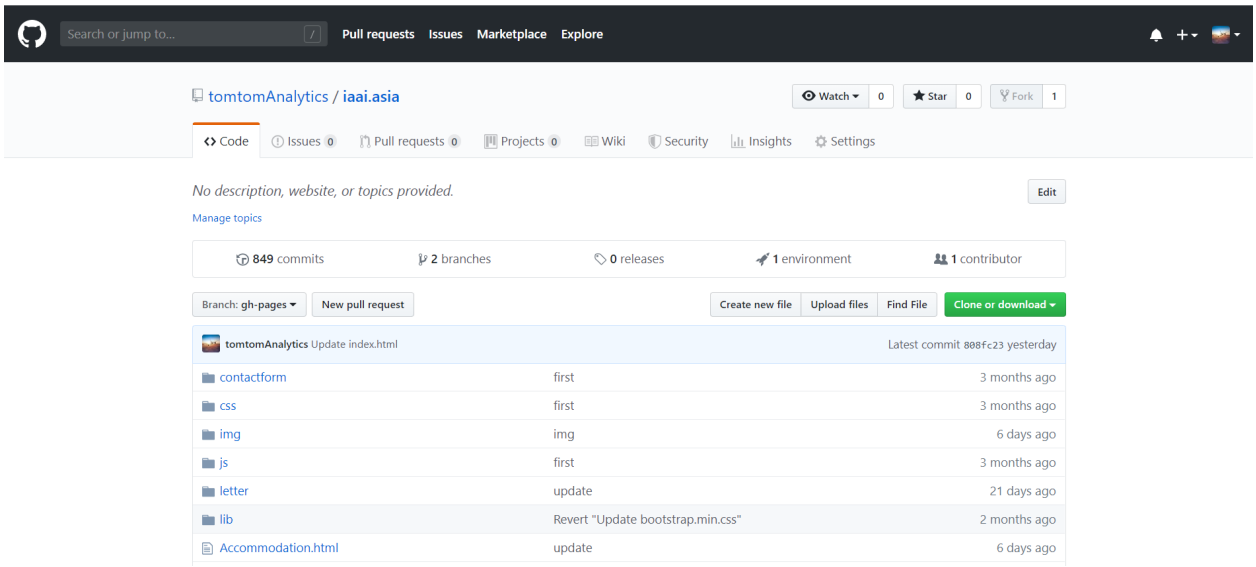


Figure 21 Example of the repository in GitHub

1.3.2 Version Control Using GitHub Desktop

What is Version Control and Why Use It?

Version control helps track the document or modified source code in each version, and the powerful version control method can help the programmer save time to find which file is corresponding to what it changes. It is the same as other documents that we have to write or modify it in many versions when we would like to step back to the previous version. It considers being a difficult task, especially in the source code of all programming languages.

The most solution that we use is named that file as related as possible, or some persons use the word or syntax that indicate version directly as Figure 22.

```
mydocument.txt
mydocumentversion2.txt
mydocumentwithrevision.txt
mydocumentfinal.txt
```

Figure 22 Typically named document version (https://programminghistorian.org/en/lessons/retired/getting-started-with-github-desktop)

Some persons define the document version by using some system such as a date-time system to indicate the last time of modification.

```
mydocument2016-01-06.txt
mydocument2016-01-08.txt
```

Figure 23 Named document version by using date-time system (https://programminghistorian.org/en/lessons/retired/getting-started-with-github-desktop)

The date-time system seems too easy and straightforward to use it. However, this method still lacks the record or describe the change that is occurring in each version or the difference between the two (previous and current) document version. Therefore, if we would like to roll back the source code to the previous one, it is a hard task for the developer or programmer because they do not know or forget about the last version.

Version control can address this problem by implementing a systematic approach to recording and managing changes in files. GitHub processes a version control by taking 'snapshots' of the file and detect a change that happens between snapshots. It enables the developer to detect the change compared with the old file.

What is Git and GitHub?

Git and GitHub are two things differently. Git is originally for implement of version control design by Linus Torvalds by deploying in the way of version control in Linux, while other systems are less frequently implement version control.

Figure 24 Linus Torvalds(https://en.wikipedia.org/wiki/Linus_Torvalds)

GitHub is a company that hosts Git repositories and provides software for using Git. GitHub also provides "GitHub Desktop" to download. Recently, GitHub is the most popular host of open source projects. Even though, GitHub primary concentrate on source code, other applications or projects trend to used GitHub for their smoothly work-flow because of version control and management feature such as journal publishing, open textbooks, and humanities projects

Why Not use Dropbox or Google Drive?

Dropbox, Google Drive, and other services provide a version control in their system, but it is not enough for sufficient that we need when compare with Git. Some advantage of Git over Dropbox, Google Drive, and other services can list as follow:

- Language support: since Git is the original design for source code; therefore, it has more format support than others.
- More control: a proper version control system gives a much higher deal of control over how to manage changes in a document.
- Useful history: using version control systems like Git will allow you to produce a history of your document in which different stages of the documents can be navigated easily both by yourself and by others.

1.3.3 Using GitHub and GitHub Desktop

GitHub offers GitHub Desktop with has a Graphical User Interface (GUI) to use Git. It allows the developer to use Git without the command line, which is easy to learn with the user interface. The step for using GitHub and GitHub Desktop is to register for an account on

the website and install GitHub Desktop. To allow both are working as synchronous as Figure 25, it needs some configuration like the following step:

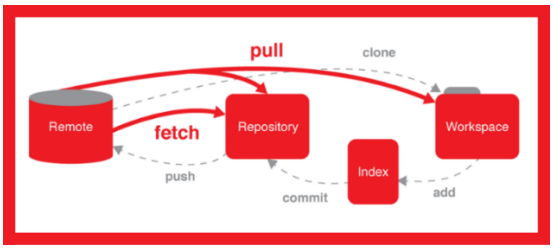


Figure 25 Workflow between GitHub website (Remote) and GitHub Desktop(workspace)(https://hackernoon.com/top-5-free-courses-to-learn-git-and-github-best-of-lot-2f394c6533b0)

1) Register for a GitHub Account as Figure 19
2) Download and Install GitHub Desktop as Figure 20
3) Creating a Repository in GitHub Account as Figure 26 and Figure 27.

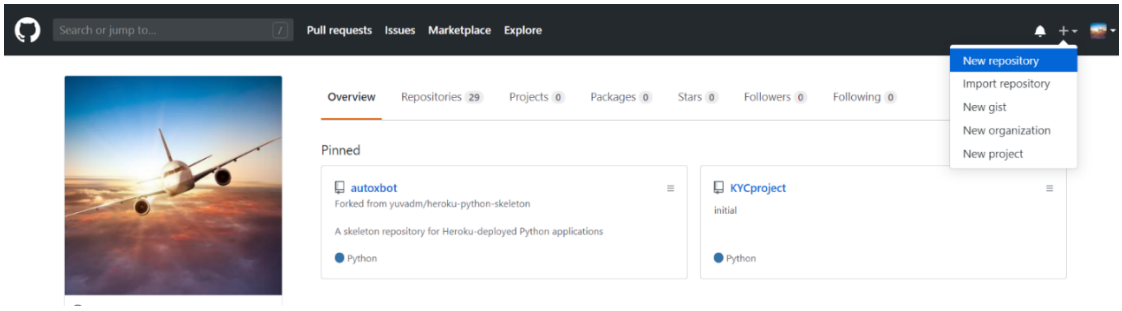


Figure 26 Creating Repository in GitHub Account (1)

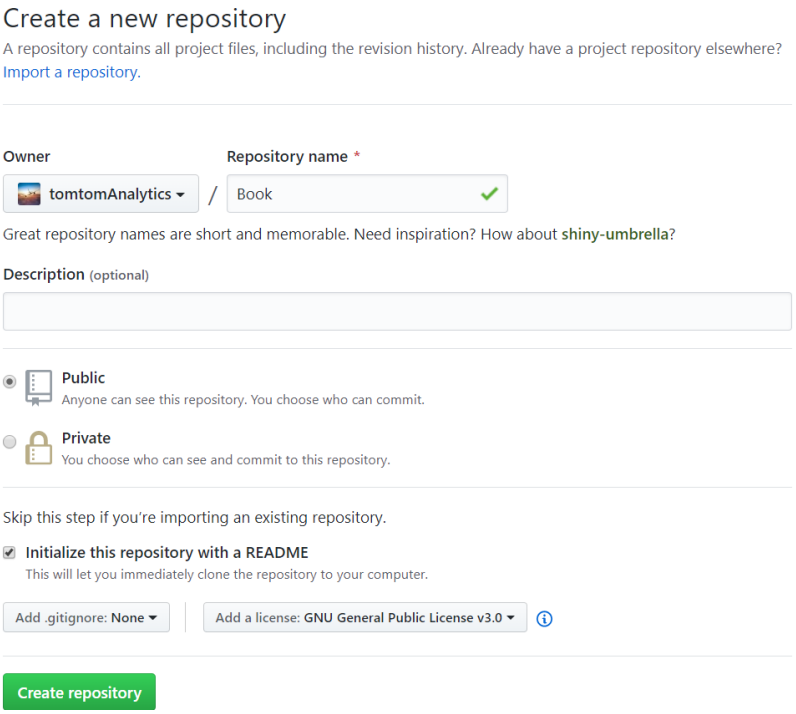


Figure 27 Creating Repository in GitHub Account (2)

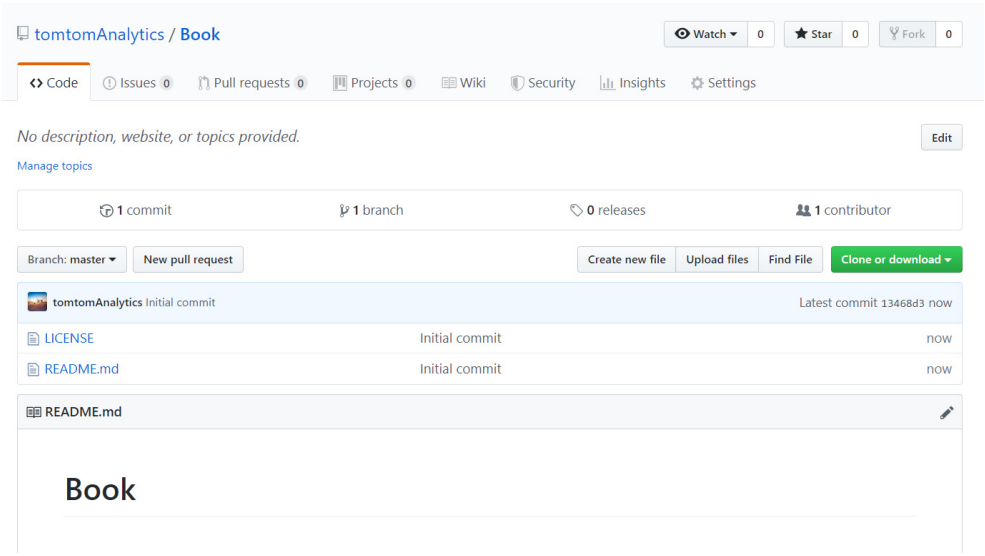


Figure 28 Creating Repository in GitHub Account (3)

4) Clone Repository from GitHub Account to Local by GitHub Desktop as Figure 29, Figure 30, Figure 31, Figure 32, and Figure 33, respectively.

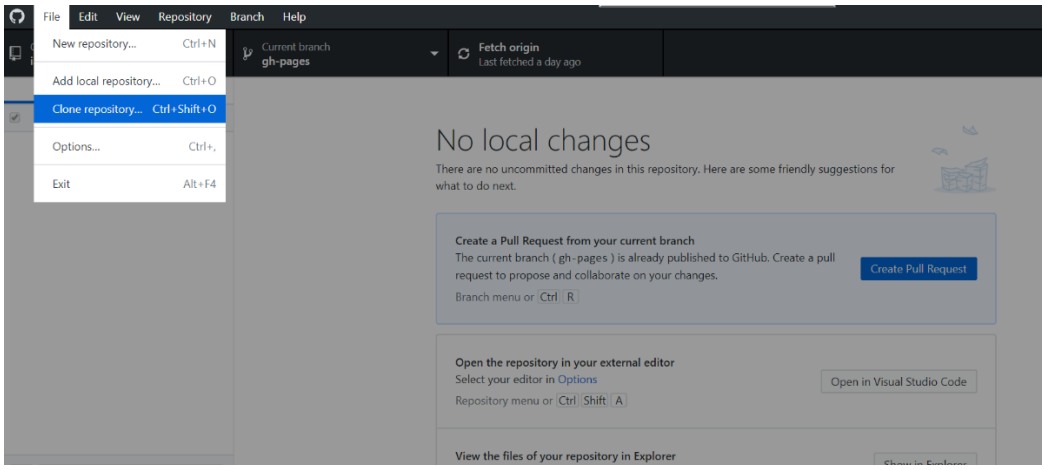


Figure 29 Clone Repository from GitHub Account to Local by GitHub Desktop (1)

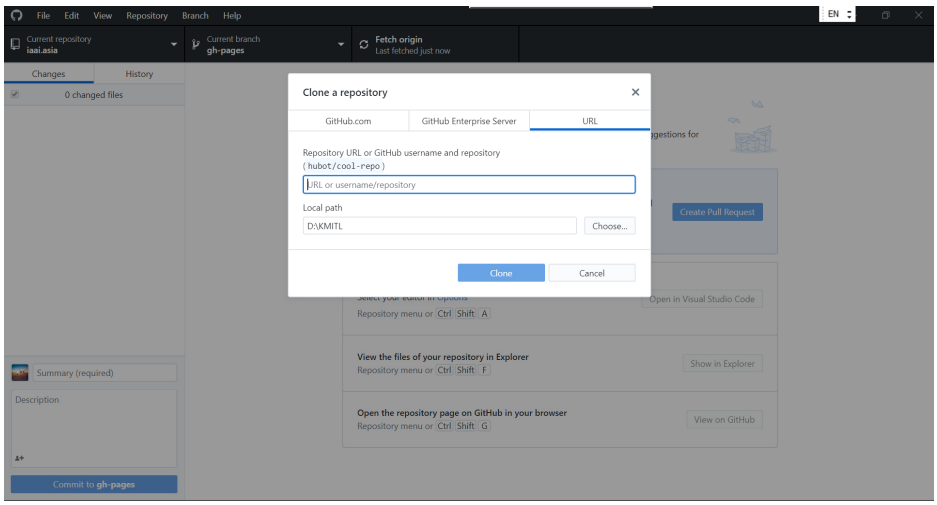


Figure 30 Clone Repository from GitHub Account to Local by GitHub Desktop (2)

Copy HTTPS path from GitHub account and paste to GitHub Desktop, then select the location in your computer and following with Clone in the last step.

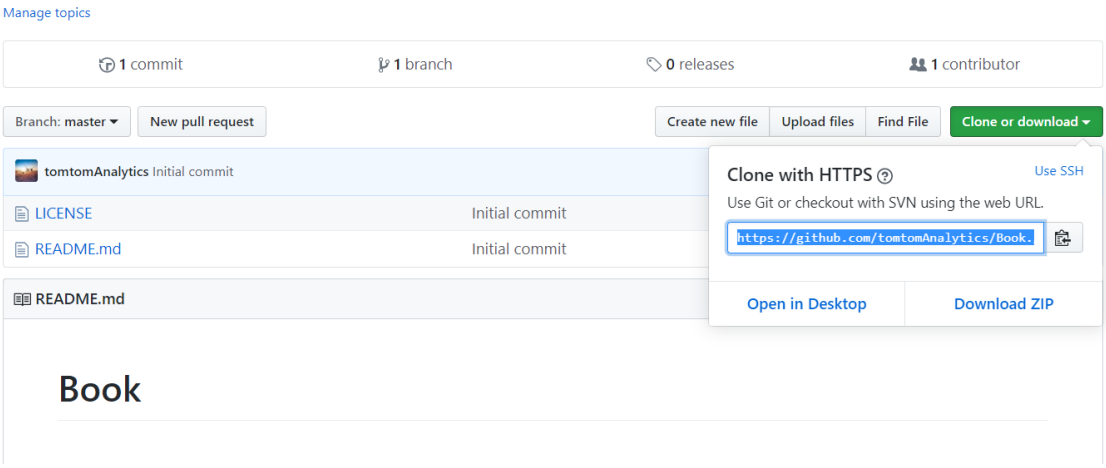


Figure 31 Clone Repository from GitHub Account to Local by GitHub Desktop (3)

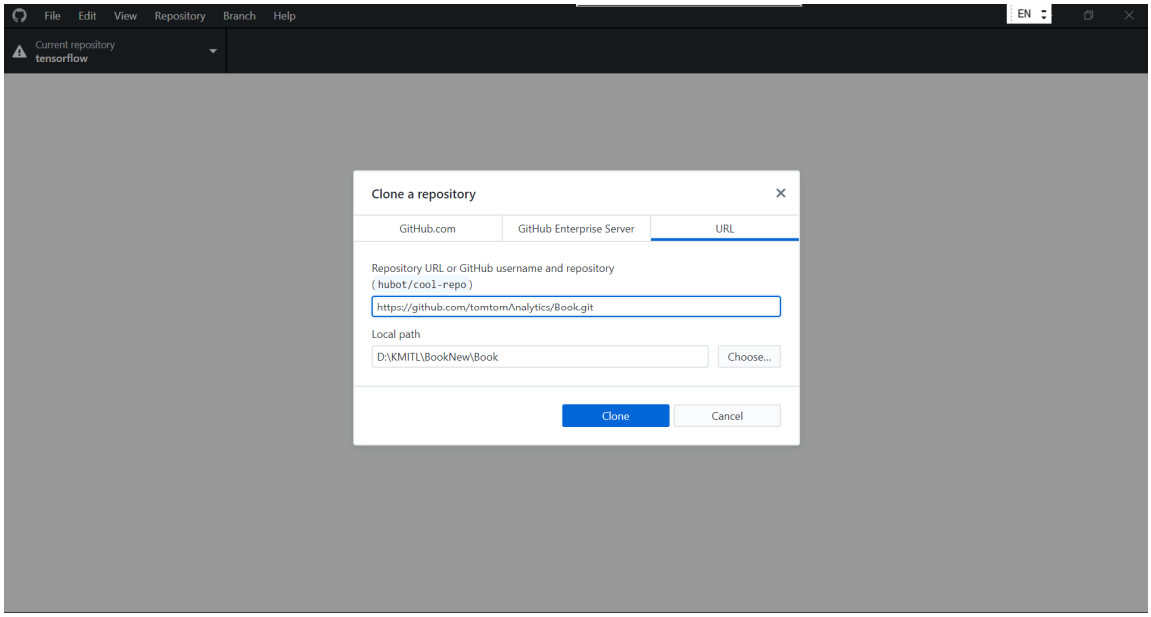


Figure 32 Clone Repository from GitHub Account to Local by GitHub Desktop (4)

After the clone Repository process finish, the result will show as the following figure.

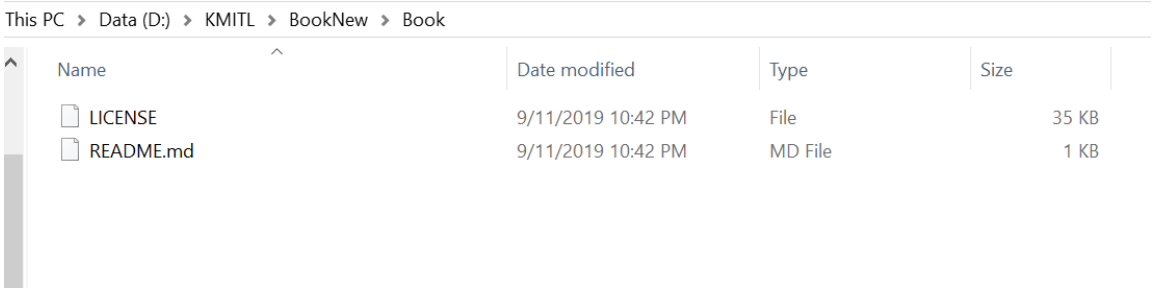


Figure 33 Clone Repository from GitHub Account to Local by GitHub Desktop (5)

Creating a Document

To make it simple, we can think that one repository is one folder, and this folder contains only the document or file that related together. For example, one folder is about homework in computer programming, and one folder is about homework in mathematics II. The repository is a similar thing with a folder, but the difference from the normal one is repository can track the change of each file in the repository. The track change can be a benefit for users or developers as previously describe, especially for version control. Figure 34 shows an example of creating a folder named "book" in the local repository.

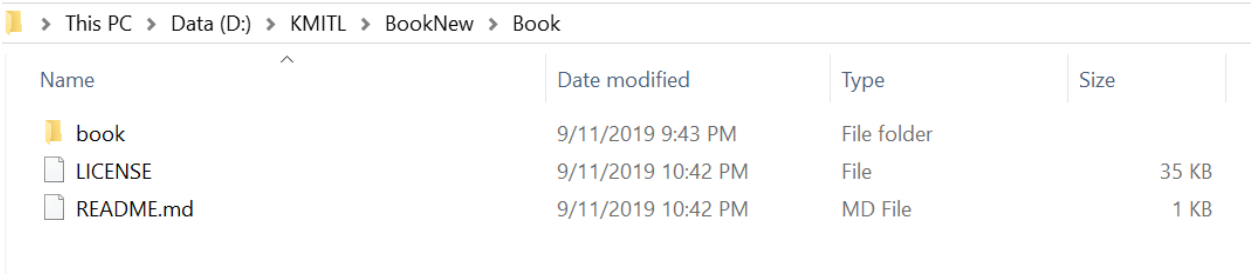


Figure 34 Create folder "book" in local repository

Adding a Document

After we create a folder in the local repository, we have added files in this folder. GitHub Desktop will track all change that we have made in this local repository and illustrate as Figure 35.

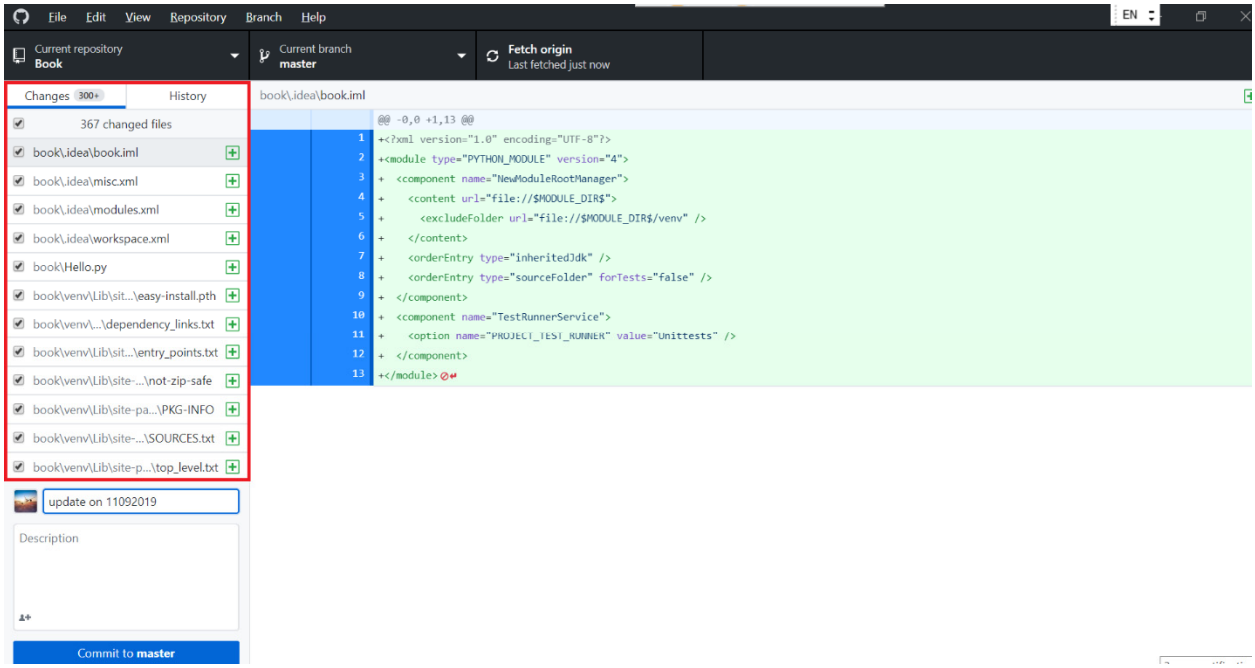


Figure 35 GitHub Desktop show the result of track change in local repository

Committing Changes

Typically, when we create some documents or files, we have to save them in some locations. A commit is a similar thing to save the filing process. However, it has something different is save the file; in general, we cannot back to see the previous version of the difference between current and previous one. In case of committing changes, it is like a snapshot that snaps the current version and records it. When it has changes and user committing a change again, Git will do a snapshot and record again. Therefore, all changes version will be kept as a record of the snapshot, resulting in we can see the change or difference between two versions or more than two versions.

In GitHub Desktop, the method to commit a change types some note in the box Summary(required) and press a Commit to master and following with Fetch origin to push all change to GitHub account as Figure 36.

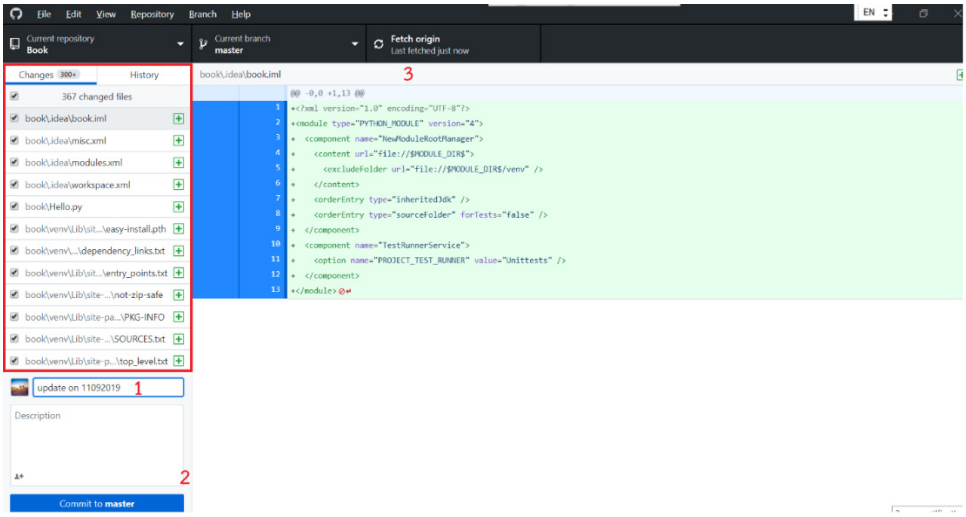


Figure 36 Committing change in GitHub Desktop

We can see the result of committing a change in the GitHub account by refresh the webpage of your GitHub account. The result will show in Figure 37.

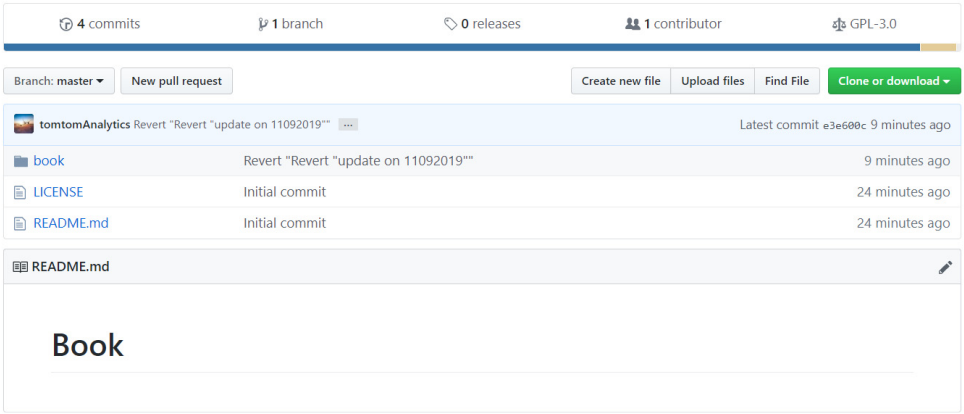


Figure 37 The result in GitHub account after committing changes

Making Changes Remotely

In GitHub account also allow us to make a change or modify the file or our source code on the website. It can be done by click on the name of the file that we would like to have a change in Figure 38 is that we would like to have a change in "Hello.py." On the right side, it will have an edit symbol in the pencil icon. After clicking the pencil icon, you will now be able to edit the file and add some new text as Figure 39.

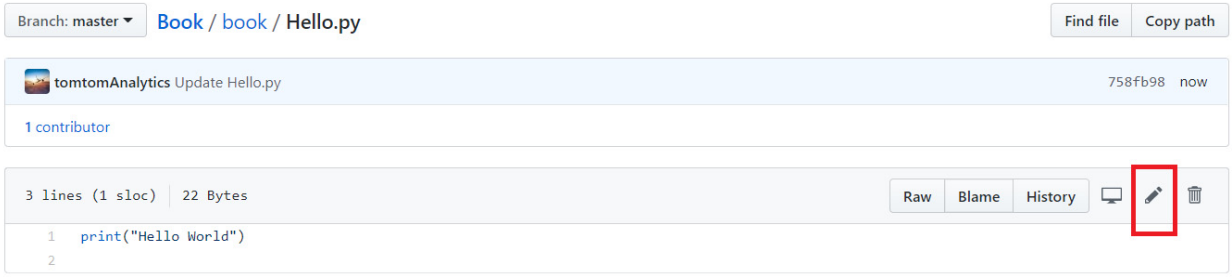


Figure 38 Example of making change remotely (1)

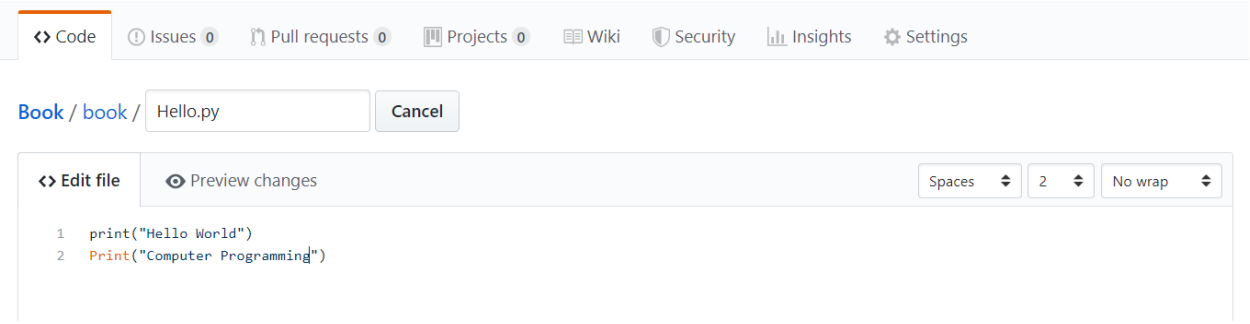


Figure 39 Example of making change remotely (2)

The same as the previous concept of GitHub Desktop, when we have made some changes in a file, we will again see the option to commit changes at the bottom of the text entry box.

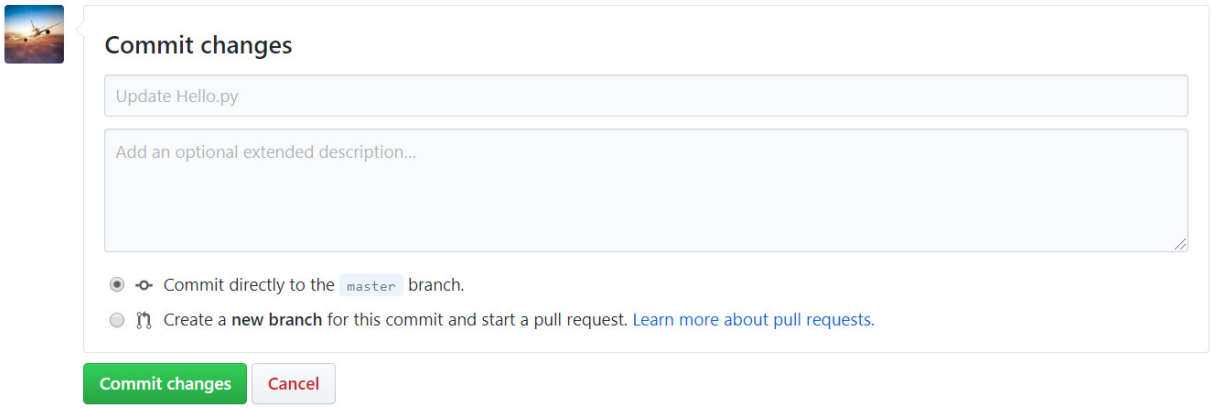


Figure 40 Commit changes in GitHub account

When we have to commit changes, all that we have made a change in the file will be record on the GitHub account. However, this change will not automatically update in GitHub Desktop until we click the 'Push Origin' button on GitHub Desktop to syn it. The next step is Push Origin to update the local repository with the changing code in the GitHub account. It just clicks Push Origin in GitHub Desktop to update it as Figure 41.

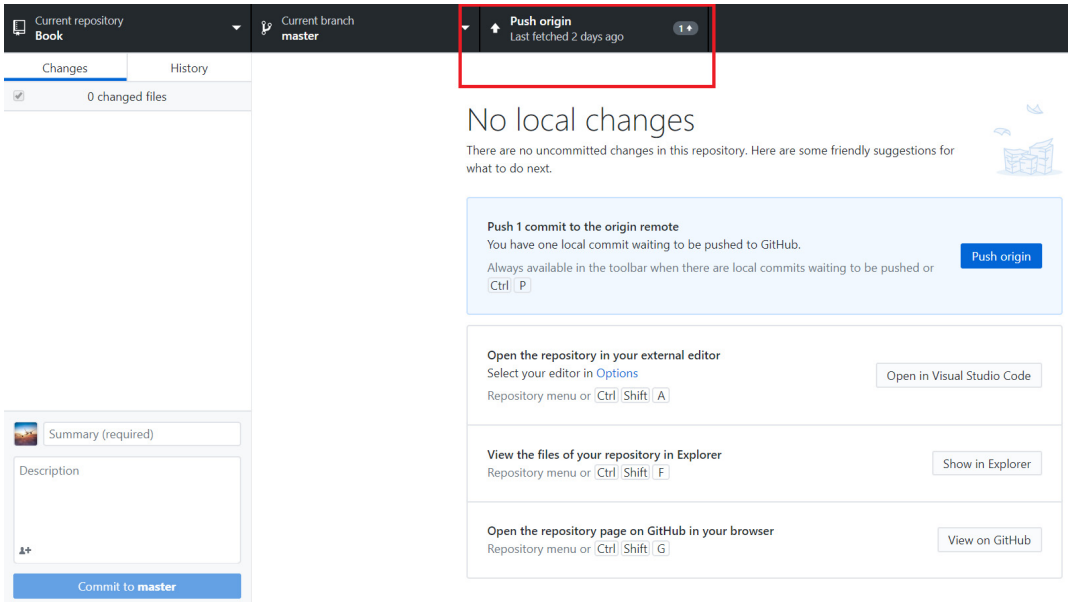


Figure 41 Push Origin in GitHub Desktop

Exercise

1. Install python compiler and PyCharm as python IDE.
2. Install GitHub Desktop, register for GitHub, and create a testing file for test version control by GitHub.
3. GitHub also can be used as a host for a website. Please try to make a website for presenting a mini-project.

Chapter 2 Variables, expressions, and statements

2.1 Values and types

The fundamental in every programming language is understanding the type of variable. Type of variable in Python in this teaching material has seven types, namely:

- Number (1)
- String (“1”)
- Boolean (True, False)
- List
- Tuple
- Set
- Dictionary

The last four types are belonging to an array that will explain in the next chapter. For the number in Python has supported four different numerical types, namely:

- int (signed integers)
- long (long integers, they can also be represented in octal and hexadecimal)
- float (floating point real values)
- complex (complex numbers)

int	long	float	complex
10	51924361L	0.0	3.14j
100	-0x19323L	15.20	45.j
-786	0122L	-21.9	9.322e-36j
080	0xDEFABCECBDAECBFBAEl	32.3+e18	.876j
-0490	535633629843L	-90.	-.6545+0J
-0x260	-052318172735L	-32.54e100	3e+26J
0x69	-4721885298529L	70.2-E12	4.53e-7j

Figure 42 Example of four data type in the number

(https://www.tutorialspoint.com/python/python_variable_types.htm)

String type will use identify for the character, for example, “Hello World,” “1”, “2”. If we would like to do conversion between data type, we can use simple syntax such as:

$int(3.1)\ obtain\ output\ \ 3$
$float(2)\ obtain\ output\ 2.0$
$int("123")\ obtain\ output\ 123$
$str(345)\ obtain\ output\ "345."$

Figure 43 demonstrates an example of a print value of 4, which is int number.

Figure 43 Example of print value in PyCharm

If you are not sure what type a value has, the interpreter (PyCharm) can tell you by setting the debug line (result in red line) before selecting Debug mode in PyCharm as Figure 44.

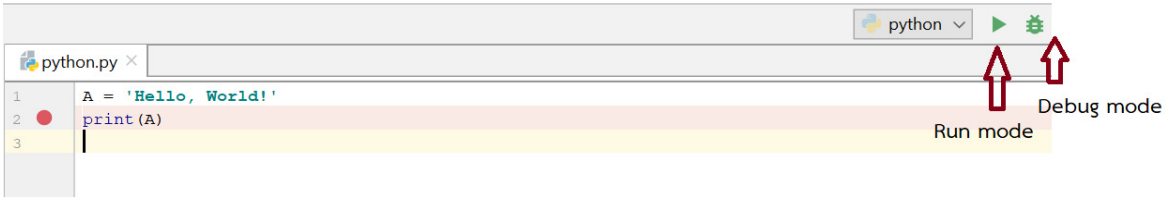


Figure 44 Debug mode to see data type in PyCharm (1)

Alternatively, right-click in the coding area and then select debug mode. The result will show in the result area as Figure 45

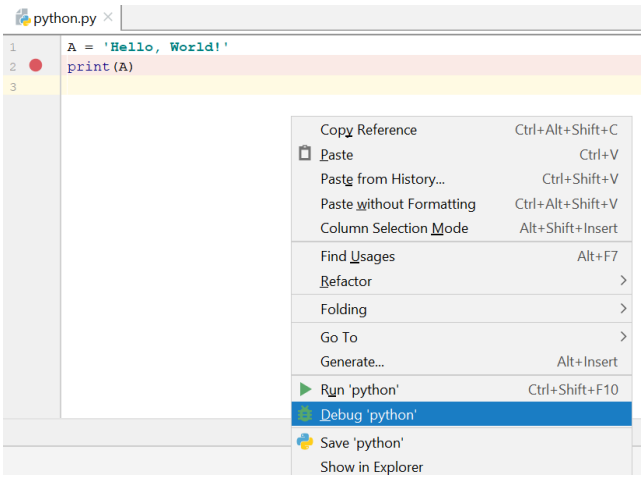


Figure 45 Debug mode to see data type in PyCharm (2)

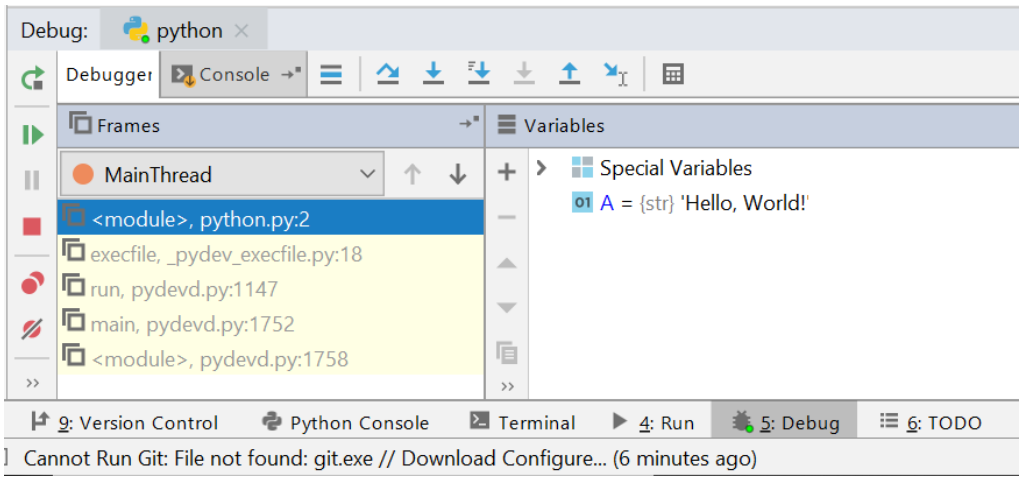
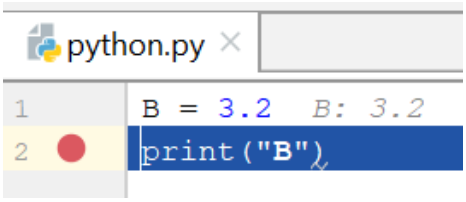


Figure 46 Result from debug mode to see data type in PyCharm

Figure 46 shows a result that variable "A" is str or string. Using the same way to see data type of $B = 3.2$ in Figure 47 present that variable "B" is float.

Code



Result

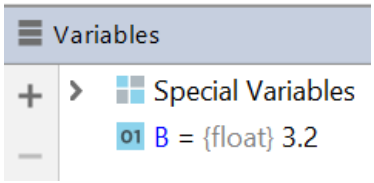
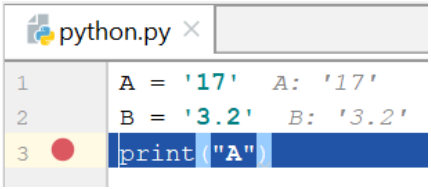


Figure 47 Result from debug mode to see data type float in PyCharm

If we try again with '17' and '3.2', In the first stage, we may think that they should be int and float, respectively. In contrast, these two variables are not as we expect because they are in quotation marks. Therefore, they both are string type as Figure 48.

Code



Result

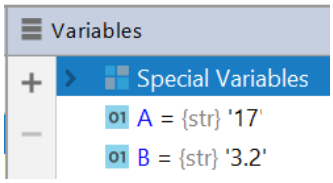


Figure 48 Result from debug mode to see data type string in PyCharm

As usual, in daily life, we will use comma to separate digit when writing a large integer such as 1,000,000. In Python language, it is illegal for integer, but it is legal to write it. However, the result

will be shown in Figure 49. In this case, a comma will have a function as space between the sequence of the integer.

Code

```
python.py
1   print(1,000,000)
```

Result

```
Run:  python
C:\Users'
1 0 0
```

Figure 49 Result from debug mode to see data type integer with a comma in PyCharm

2.2 Variables

Variable is a word that refers to a value. Variable can be assigned to be any data type. An assignment statement refers to all states, including the name of the variable and a value.

```
python.py
1   message = 'And now for something completely different'
2   n = 17
3   pi = 3.1415926535897931
```

Figure 50 Assignment statement

Figure 50 demonstrates three examples of an assignment statement. The first one is a variable named "message" that is assigned by a string. The second is variable "n" is assigned by integer 17. The last assignment statement is the approximate value of π assigns a variable named "pi." To make it easier understand, we can write the assignment statement in term of state diagram as Figure 51, which use the arrow to point between variable name and value.

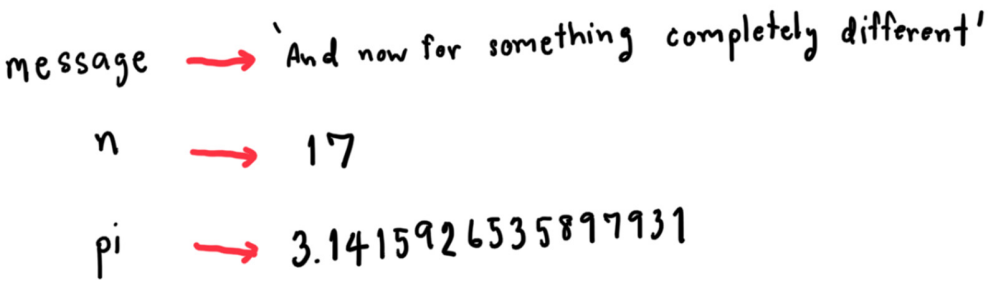


Figure 51 State diagram of the assignment statement

Once the variable was assigned, we can display the value of a variable by using a print statement as Figure 52:

Code

```
message = 'And now for something completely different'
n = 17
pi = 3.1415926535897931
print(n)
print(pi)
```

Result

```
C:\Users\LENOVO\Ap
17
3.141592653589793
```

Figure 52 Print value of the variable

Also, it can show the type of a variable, which is the type of value it refers to.

Code

```
message = 'And now for something completely different'
n = 17  n: 17
pi = 3.1415926535897931  pi: 3.141592653589793
print(n)
print(pi)
```

Result

Variables

Special Variables

message = {str} 'And now for something completely different'

n = {int} 17

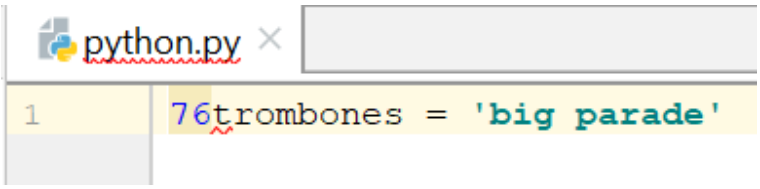
pi = {float} 3.141592653589793

Figure 53 Type of variable refer to it assign value

2.3 Variable names and keywords

A right variable name should meaningful to its function or its meaning. Programmers can choose the names as they want. It can contain both letter and number, and both uppercase letters and lowercase letters. Variable names can be arbitrarily long. Variable illegal to begin with a number, it allows the only letter which is uppercase or lowercase. The programmer frequently begins the variable name with a lowercase letter. Sometime, the variable will have a long length to indicate its role or value. In this case, we can use the underscore character (_) to connect between the word of a variable. For example, “horizontal_distance” or “vertical_distance.” In Python programming, if we choose an illegal name such as a variable begin with a number, it will get an error syntax once write and run a program as Figure 54.

Code



Result

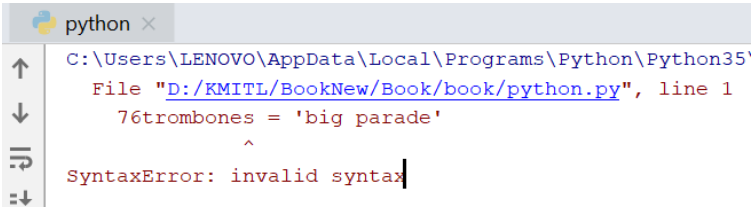


Figure 54 An illegal variable name will get a syntax error

Another rule to choose a variable name is keywords. It has 31 keywords in Python that illegal to use as a variable name because the interpreter uses these keywords to construct a program.

```
and       del       from      not       while
as        elif      global    or        with
assert    else      if        pass      yield
break     except    import    print
class     exec      in        raise
continue  finally   is        return
def       for       lambda    try
```

Figure 55 Keyword in Python

2.4 Statements

The previous section already mentions about assignment statement. A statement is a unit of code that Python interpreter can execute. From the previous, we can see that there are two kinds of statements, namely print and assignment. The interpreter will process as a sequence of the statement like the following example:

Code	Produces the output
$print(1)$	1
$x = 2$	
$print(x)$	2

It can see that the assignment statement produces no output.

2.5 Operators

A special symbol is used for computation on the programming language. It can represent a mathematic operation called “operator.” In different programming, language has a different symbol for each operator. Therefore, the developer should learn and understand before beginning a program. For Python, we can categorize an operator as following [4]:

- Arithmetic operators
- Comparison operators
- Logical operators
- Bitwise operators
- Assignment operators
- Membership operators

1) Arithmetic operators

Table 1 Arithmetic operators

Operator	Meaning	Example
+	Add two operands or unary plus	x + y
-	Subtract right operand from the left or unary minus	x - y
*	Multiply two operands	x * y
/	Divide left operand by the right one (always results into float)	x / y
%	Modulus - the remainder of the division of left operand by the right	x % y (remainder of x/y)
//	Floor division - division that results in the whole number adjusted to the left in the number line	x // y
**	Exponent - left operand raised to the power of right	x**y (x to the power y)

Try with $x = 12, y = 20$

2) Comparison operators

Table 2 Comparison operators

Operator	Meaning	Example
>	Greater than - True if the left operand is greater than the right	x > y
<	Less than - True if the left operand is less than the right	x < y
==	Equal to - True if both operands are equal	x == y
!=	Not equal to - True if operands are not equal	x != y
>=	Greater than or equal to - True if the left operand is greater than or equal to the right	x >= y
<=	Less than or equal to - True if the left operand is less than or equal to the right	x <= y

Try with $x = 20, y = 12$

3) Logical operators

Table 3 Logical operators

Operator	Meaning	Example
and	True if both the operands are true	x and y
or	True if either of the operands is true	x or y
not	True if the operand is false (complements the operand)	not x

Try with $x = Ture, y = False$

4) Bitwise operators

Table 4 Bitwise operators

Operator	Meaning	Example
&	Bitwise AND	x& y
|	Bitwise OR	x | y
~	Bitwise NOT	~x
^	Bitwise XOR	x ^ y
>>	Bitwise right shift n bit	x << n
<<	Bitwise left shift n bit	x >> n

Try AND, OR, XOR with $x = 0, y = 0: x = 0, y = 1: x = 1, y = 0: x = 1, y = 1$

5) Assignment operators

Table 5 Assignment operators

Operator	Example	Equivalents to
=	x = 5	x = 5
+=	x += 5	x = x + 5
-=	x -= 5	x = x - 5
*=	x *= 5	x = x * 5
/=	x /= 5	x = x / 5
%=	x %= 5	x = x % 5
//=	x //= 5	x = x // 5
**=	x **= 5	x = x ** 5
&=	x &= 5	x = x & 5
|=	x |= 5	x = x | 5
^=	x ^= 5	x = x ^ 5
>>=	x >>= 5	x = x >> 5
<<=	x <<= 5	x = x << 5

6) Membership operators

Table 6 Membership operators

Operator	Meaning	Example
in	True if value/variable is found in the sequence	5 in x
not in	True if value/variable is not found in the sequence	5 not in x

Once $x = 'Hello\ world$

Try “H” in x or not?

2.6 Expressions

An expression is a combination of values, variables, and operators. It is different from the statement that statement only do something, while, expression representation of value. The example of an expression is $x\ for\ x\ in\ range(10)$.

2.7 Order of operations

The expression can have more than one operator. The order to process the operator in Python similar to the mathematical convention.

1) Parentheses or () is not only helping us read the expression easier but also the highest order, and we can use parentheses to force the interpreter process orderly as we want because of it the same as the rule of mathematic that process in the parentheses in first priority. For example, $2*(3-1)$ is 4, and $(1+1)**(5-2) = 8$.
2) Exponentiation has the next highest precedence. For instance, $2**1+1 = 3$, not 4, and $3*1**3 = 3$, not 27.
3) Multiplication and division have the same level, which is higher than addition and subtraction, which also have the same level. For example, $2*3-1 = 5$, not 4 , and $6+4/2 = 8$, not 5.
4) Operators with the same order level are evaluated from left to right.

2.8 String operations

It is different from number operations; string operation cannot perform even though some string expresses like a number. For example, $'2'-'1'$ or $'eggs'/'easy'$. If we try to use some operation such as +,* with string, it may present unexpected result such as:

```
first = 'throat'
second = 'warbler'
print(first + second)
```

The output will show the concatenation of two variables “first” and “second,” which the result will show “throatwarbler”

The multiple operators (*) can also work on strings by performing a repetition of string. For example, $'Hello'*3$ resulting in $HelloHelloHello$. Therefore, the developer should pay more attention when needs to use string operations in their programming.

2.9 Comments

Programming is a thing to maintain and improve always to obtain the best efficiency. It makes a program come to complicated and extensive when we have some modifications. Especially, programming as teamwork which all team member needs to understand and know that which part belongs to which programmer. A comment can help us to leave some message for each program line or each section. Comment in Python will begin with a symbol #. For example,

$$\#\ compute\ the\ percentage\ of\ the\ hour$$
$$percentage\ =\ (minute\ *\ 100)\ /\ 60$$

Alternatively, we can put a comment in the same line with code as:

$$percentage\ =\ (minute\ *\ 100)\ /\ 60\ \#\ percentage\ of\ an\ hour$$

The interpreter will ignore all letters or number after symbol #, which mean it does not include in the process of a program and no affect the program. A comment is useful for the program because of can explain the information or logical thinking of the programmer, and it can help programmer to save a time when they want to fix some feature in the program. Another example is:

$$v = 5\ \#assign\ 5\ to\ v$$

This comment contains useful information that is not in the code:

$$v\ =\ 5\ \#velocity\ in\ meters/second$$

More tellingly, we can use both comments and choose a good variable name to detail indicating information or feature of the code line. However, this may make the variable name too long. Thus, the tradeoff between long variable names and comments should be considered.

Exercise

1. Prints my name, address, and phone number.
2. Computes and prints the area of a rectangle, given an input width and height.
3. Computes and prints the area of a triangle, given an input base and height.
4. Computes and prints the area of a circle, given an input radius.
5. Input the user's name and age and outputs a sentence containing them.
6. Compute a person's income tax.
    a. Significant constants
        i. tax rate
        ii. standard deduction
        iii. deduction per dependent
    b. The inputs are
        i. gross income
        ii. number of dependents
    c. Computations:
        i. net income = gross income - the standard deduction - a deduction for each dependent
        ii. income tax = is a fixed percentage of the net income
    d. The outputs are
        i. the income tax, rounded to two figures
7. Compute the surface area of a cube. The input is the cube's edge. The output is the surface area of the cube.
8. Calculate the total charge for a customer's video rentals, given the number of each type of video.
    a. New video rental = $3.00
    b. Oldie rental = $2.00
9. Given the radius, compute the diameter, circumference, and volume of a sphere.
    a. Useful facts:
        i. diameter = 2 * radius

   ii. circumference = diameter * 3.14
   iii. surface area = 4 * PI * radius * radius
   iv. volume = 4/3 * PI * radius * radius * radius

10. Given an object's mass and velocity, compute its momentum.
11. Given an object's mass and velocity, compute its momentum and kinetic energy.
12. Compute the number of minutes in a year.
    a. Useful facts:
       i. 1 year = 365 days (we ignore leap years)
       ii. 1 day = 24 hours
       iii. 1 hour = 60 minutes
13. Compute the distance that light travels in a year.
    a. Useful facts:
       i. rate = 3 * 10 ** 8 meters per second
       ii. seconds in a year = 365 * 24 * 60 ** 2
14. Convert kilometers to nautical miles.
    a. Useful facts:
       i. 1 kilometer = 1/10000 of the distance between the North Pole and the Equator
       ii. there are 90 degrees between the North Pole and the Equator
       iii. 1 degree = 60 minutes of arc
       iv. 1 nautical mile = 1 minute of arc
15. Compute an employee's total weekly pay.
    a. Useful facts:
       i. An employee's total weekly pay equals the hourly wage multiplied by the total number of regular hours plus any overtime pay. Overtime pay equals the total overtime hours multiplied by 1.5 times the hourly wage.

Write a program that takes as inputs the hourly wage, total regular hours, and total overtime hours and displays an employee's total weekly pay.

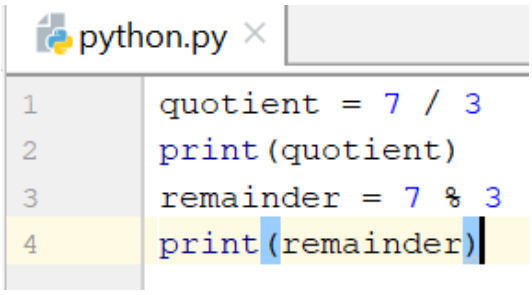
Chapter 3 Conditionals

This chapter explains the detail of conditionals in Python programming. The first section is a modulus operator who frequently implements to check a condition by the remainder of the divided operation. The next is detail about Boolean and logical operators, respectively. The remaining section is about the execution of several types of conditions.

3.1 Modulus operator

Modulus operator symbolic with a percent sign (%) working on integers and give output to be a reminder of the first variable is divided by the second variable. Figure 56 demonstrates the use and results of the divided operator compared with the modulus operator.

Code



Result

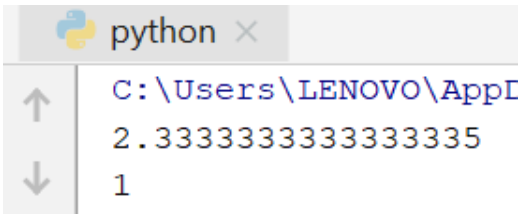


Figure 56 Example of the modulus operator

The first variable is 7 divided by 3, providing a remainder is 1. The modulus can use to check whether $x\%y = 0$ to indicate an extraction. For example, $x\%10 = 0$ indicates that x can extract by 10. Similarly, $x\%100 = 0$ indicates that x can extract by 100.

3.2 Boolean expressions

Another expression except for modulus, which mostly uses in conditional, is a Boolean expression. Boolean is either true or false. Figure 57 gives an example of Boolean expression by using an equal (==) operator. The result of the operator will come from comparing the two variables.

Code

```
python.py
1  print(5==5)
2  print(5==6)
```

Result

```
python
C:\Users\
True
False
```

Figure 57 Example of a Boolean expression

The summation of the Boolean operator already mentions in the previous section and also will give more detail as the following:

$x! = y$ # x is not equal to y

$x > y$ # x is greater than y

$x < y$ # x is less than y

$x >= y$ # x is greater than or equal to y

$x <= y$ # x is less than or equal to y

One thing that makes the beginner confuse is the difference between an assignment operator and a comparison operator. For example, the symbol = is using for assignment value, while the double equal (==) is used for comparison operators. Therefore, it should be careful to use it. The following link is an example case of a developer using a wrong symbol but is just a human error, not about logical thinking(https://zcoin.io/zcoins-zerocoin-bug-explained-in-detail/).

3.3 Logical operators

As already mentioned in the previous section, a logical operator has three operators, namely and, or, and not. The meaning of these three operators is the same as the meaning in English. For example, $x > 0 \; and \; x < 10$ is true only if x is greater than 0 and less than 10. Another example is $n\%2 == 0 \; or \; n\%3 == 0$ is true if either of them has a condition true. Not operation is negating Boolean expression such as $not(x > y)$ is true if $x > y$ is false. The logical operators should give a result only true or false, but in Python, it is not very strict such as the following example that non-zero number do and operator with Boolean expression(true) resulting in True in the left side.

Code Result

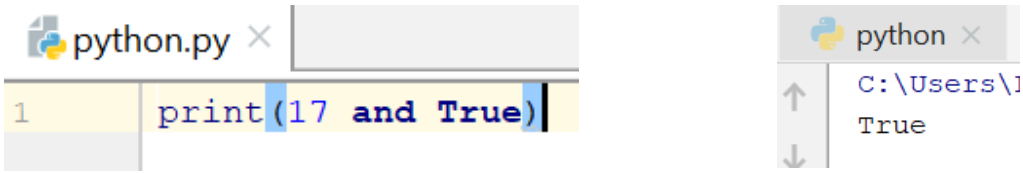


Figure 58 Non-zero number do and operator with Boolean expression(true)

3.4 Conditional execution

Conditional execution can be used for changing the workflow of the program because it can check conditions to allow or not allow the program doing which process by using the condition. For instance,

$if\ x > 0$:
    $print('x\ is\ positive')$

If the result of Boolean expression after if which in here is $x > 0$, which we called the condition, is true, the next indented code line will get executed. In contrast, if the result is false, there is nothing happen. In this example, if a statement can compare to be a function that contains a header and followed by an indented block. It is no limit of using condition in this expression, but at least one condition should appear. The conditional execution can also use for passing a statement, which means do nothing if the statement is true, such as:

$if\ x < 0$:
    $\#pass, no\ need\ to\ handle\ negative\ values$!

3.5 Alternative execution

Alternative execution can express by using if statement. It will have two choices or conditions, and the program will process as a way that the condition is true. The example is in Figure 59.

Code

```
python.py
1  x=10
2  if (x%2 == 0):
3      print ('x is even')
4  else:
5      print ('x is odd')
```

Result

```
python
C:\Users\LEI
x is even
```

Figure 59 if-else statement in alternative execution

As present in Figure 59, once x modulus by 2 and get result 0, the program will ddisplay$'x\ is\ even'$. If this condition is false, the program will display $'x\ is\ odd'$. The alternative execution will process either process depending on a result from the condition after if statement, but it has only one condition that will be true or false. Therefore, it will process only one branch.

3.6 Chained conditionals

Chained conditionals are used in case they have more than two branches. It can write a program as a chained of a condition such as Figure 60, which uses $if - elif - else$ for doing a chained condition.

Code

```
python.py
1  x = 1
2  y = 5
3  if (x < y):
4      print ('x is less than y')
5  elif (x > y):
6      print ('x is greater than y')
7  else:
8      print ('x and y are equal')
```

Result

```
python
C:\Users\LENOVO\A
x is less than y
```

Figure 60 $if - elif - else$ statement in chained condition

$elif$ is an abbreviation of "else if." Same as alternative execution, it will have only one branch executed, and there is no limit of $elif$ statement.

Each condition is checked in order. If the first is false, the next is checked, and so on. If one of the conditions is true, the corresponding branch executes, and the statement ends. Even if more than one condition is true, only the first true branch executes.

3.7 Nested conditionals

A condition can have sub-branches which has an example like Figure 61 or generally speaking that condition in the condition in this example.

```python
x = 1
y = 5
if (x == y):
    print ('x and y are equal')
else:
    if (x < y):
        print ('x is less than y')
    else:
        print ('x is greater than y')
```

Figure 61 Nested conditionals

The outer expression has two branches, which the first branch is easy to understand, while the second one has another condition inside. The inner condition also has two branches, the same as the outer one. There is no limit to having several nested conditionals, and it depends on the design of the program. However, many nested conditions make it difficult to understand the program, and we should avoid it by using a simple way, as presenting in Figure 62 and Figure 63, respectively.

```python
x = 1
y = 5
if 0 < x:
    if x < 10:
        print ('x is a positive single-digit number.')
```

Figure 62 Collapse nested conditionals (1)

```python
x = 1
y = 5
if 0 < x and x < 10:
    print ('x is a positive single-digit number.')
```

Figure 63 Collapse nested conditionals (2)

3.8 Keyboard input

Python has a built-in-function that called input. This function allows the program has interacted with the user. The program will wait for a user to type something for the next processing. Once the user return or hit enter to come back to a program, the next task will continue.

Code

```
python.py
1  input = input()
2  print(input)
```

Result

```
python
C:\Users\LENOVO\AppData\Lo
What are you waiting for?
What are you waiting for?
```

Figure 64 Example of using keyboard input in Python

Exercise

1. Determine whether or not three input sides compose an equilateral triangle.
2. Determine whether or not three input sides compose a right triangle.
3. Shifts the bits in an input string one place to the left. The leftmost bit wraps around to the rightmost position.
4. Shifts the bits in an input string one place to the right. The rightmost bit wraps around to the leftmost position.

Chapter 4 Loops and Iteration

There are several types of loops in a programming language. However, there are only two types of loops that widely use in program development. There are two kinds of the loop in this chapter, namely for loop and $while$ loop will focus. The primary for loop use once one knows the number of iterations and uses $while$ loop when one does not know the number of iterations a priori [5].

4.1 For loop

for loop can use for executing each element or item in a list, tuple, set, which is a kind of array. Python does not need to set an index number before processing for loop.

Code

```
python.py
1  fruits = ["apple", "banana", "cherry"]
2  for x in fruits:
3    print(x)
```

Result

```
python
C:\Users\
apple
banana
cherry
```

Figure 65 Example of for loop

From Figure 65, we can set a read point and line in PyCharm to get into debug mode and see how it is processing in each loop. It will begin with the first loop to print the first item in the variable "fruits," and so on.

4.1.1 Looping Through a String

for loop also can process in a string. The string contains a sequence of many characters. The program can see that this sequence is a set of letters that compose to be a string. For example, Figure 66 presents for loop working on string "banana." The loop will print one by one character of "banana."

Code Result

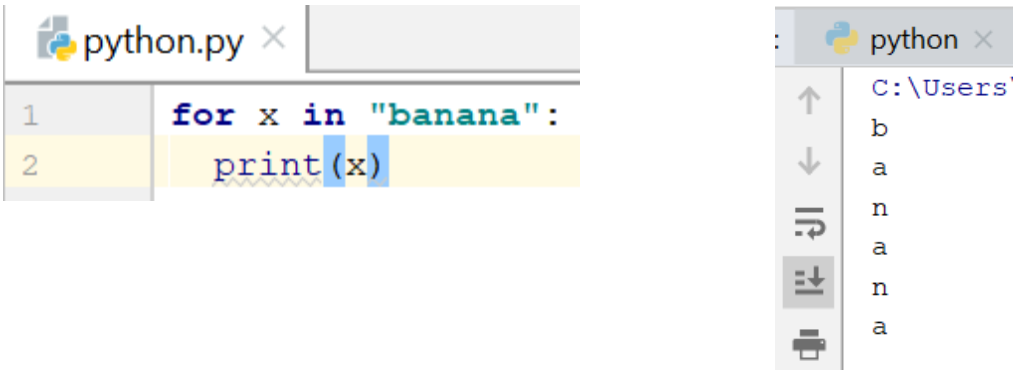
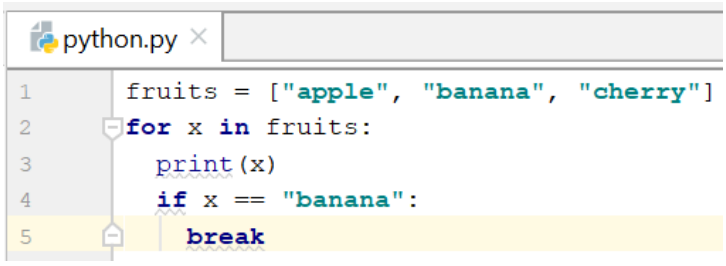


Figure 66 Example of for loop in a string

4.1.2 The break Statement

$break$ statement has an advantage in case if we would like to stop the loop if it reaches the expected result. For instance, Figure 67 shows using $break$ statement after printing the first two items already. While Figure 68 put the different location of the print statement, resulting in the program can print only the first item because when the item is "banana," the loop already exit before printing it.

Code



Result

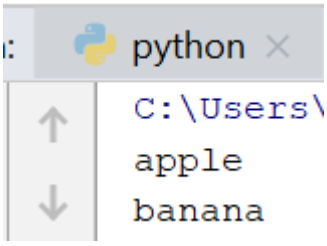
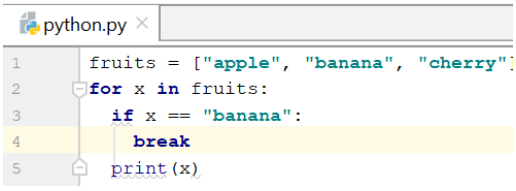


Figure 67 $break$ statement for stop a loop (1)

Code



Result

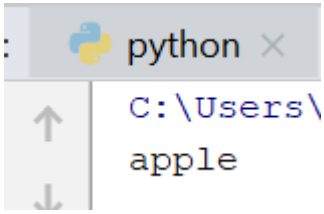


Figure 68 $break$ statement for stop a loop (2)

4.1.3 The continue Statement

continue statement use for stopping the current loop and go on for the next loop. Figure 69 demonstrates that when the item is "banana," the loop will stop and go to process that next loop, which is print "cherry."

Code

```
python.py
1  fruits = ["apple", "banana", "cherry"]
2  for x in fruits:
3    if x == "banana":
4      continue
5    print(x)
```

Result

```
python
C:\Users\LE
apple
cherry
```

Figure 69 Example of *continue* statement

4.1.4 The range() Function

range is used for specific number of times. The number after *range* will be the end. For example, if *range*(3) as Figure 70, the program will print from 0 and increments by 1 by default and will end at number 3 (exclude 3).

Code

```
python.py
1  for x in range(3):
2    print(x)
```

Result

```
python
C:\Users
0
1
2
```

Figure 70 Example of using *range*

As a similar concept, *range*(6) is not the values of 0 to 6, but the values 0 to 5. Therefore, keep in mind that the program will begin from 0. However, it also can specify the starting value by setting additional parameter such as *range*(2,6), it means the value will start from 2 to 6 (exclude 6) as Figure 71.

Code

```
python.py
1  for x in range(2, 6):
2    print(x)
3
```

Result

```
python
C:\Users\I
2
3
4
5
```

Figure 71 Example of $range$ by setting a start value

Normally, $range$ will increase equal to 1 in every iteration by default. However, it can also be set as a specific number by a parameter in $range$. For example, $range(2,11,2)$ in Figure 72. It means that the value will begin from 2 until 11 (exclude 11) by an increment of 2.

Code

```
python.py
1  for x in range(2, 11, 2):
2      print(x)
```

Result

```
python
C:\Users\
2
4
6
8
10
```

Figure 72 Example of $range$ by setting a start and increment value

4.1.5 Nested Loops

A nested loop is a loop inside a loop. There contain the outer and inner loop. Figure 73 illustrates a nested loop that the outer loop is about color, and the inner loop is about fruit. The program will process the first outer loop and make the inner loop until complete and then process the second outer loop and so on.

Code

```
python.py
1  adj = ["red", "big", "tasty"]
2  fruits = ["apple", "banana", "cherry"]
3
4  for x in adj:
5      for y in fruits:
6          print(x, y)
```

Result

```
python
C:\Users\LENOVO\
red apple
red banana
red cherry
big apple
big banana
big cherry
tasty apple
tasty banana
tasty cherry
```

Figure 73 Example of nested loops

4.2 While loop

$while$ loop mainly use a advantage of Boolean, by the $while$ loop will execute as long as a condition after $while$ will be true such as Figure 74 use $while$ loop to print a number since 1 to 5 or less than 6 (<6).

Code Result

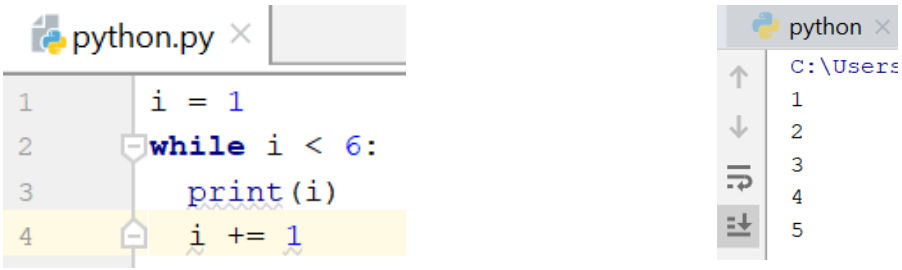


Figure 74 Example of ***while*** loop

The essential difference between for and ***while*** loop is indexing number or indexing variable, which in Figure 74 is i. The indexing number in this example will ready or begin with 1 and every ***while*** loop, it will increase 1 until the end of the loop.

4.2.1 The break Statement

while loop also have a ***break*** statement as same as for loop. The function is also the same as in for loop, which uses the user stopping the loop. Figure 75 shows using ***break*** statement in ***while*** loop. Everything will stop one $i == 3$, but the program will display 3 because of putting $print(i)$ before checking the condition.

Code

Result

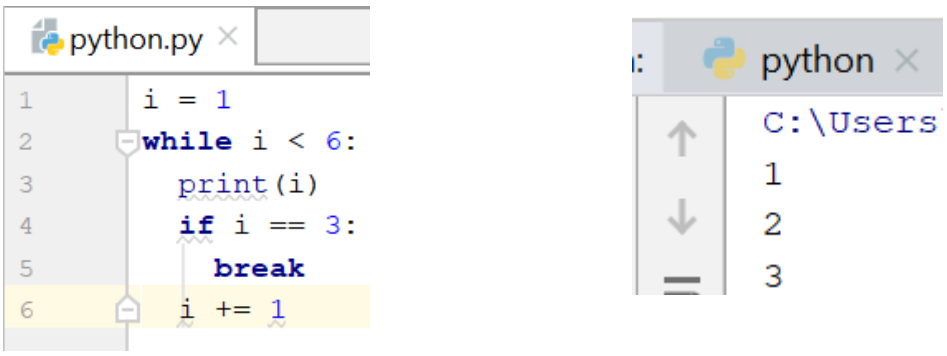


Figure 75 Example of ***break*** statement in ***while*** loop

4.2.2 The continue Statement

continue statement in ***while*** loop works the same as for loop, that is stop the current iteration and go on in the next iteration. Figure 76 shows the example of ***continue*** statement in ***while*** loop. It ignores the iteration that $i == 3$ and go to do a processing in the next iteration.

Code

Result

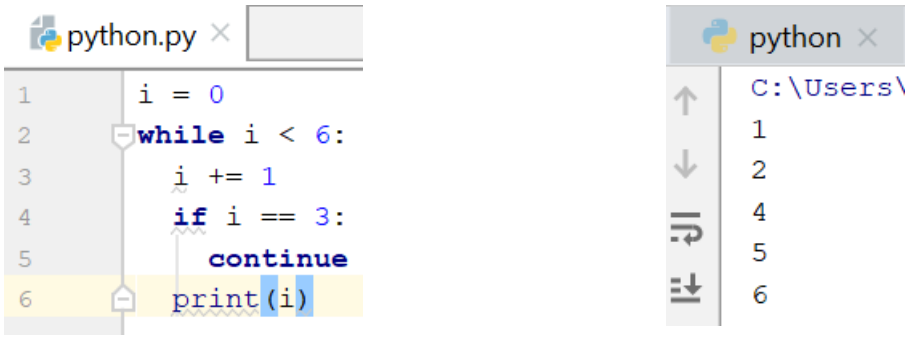


Figure 76 Example of ***continue*** statement in ***while*** loop

Exercise

1. The computer guesses the user's number using the minimum number of attempts and prevents cheating by the user.
2. Compute and print the greatest common divisor of two input integers.
3. Computes the sum and average of a series of input numbers.
4. Print a payment schedule for a loan to purchase a computer.
    a. Input
        i. purchase price
    b. Constants
        i. annual interest rate = 12%
        ii. downpayment = 10% of purchase price
        iii. monthly payment = 5% of purchase price
5. Simulate the game of lucky sevens until all funds are depleted.
    a. Rules:
        i. roll two dice
        ii. if the sum equals 7, win $4, else lose $1
    b. The input is:
        i. the amount of money the user is prepared to lose
    c. Computations:
        i. use a random number generator to simulate rolling the dice
        ii. loop until the funds are depleted
        iii. count the number of rolls
        iv. keep track of the maximum amount
    d. The outputs are:
        i. the number of rolls it takes to deplete the funds the maximum amount
6. Converts a decimal integer to a string of octal digits.
7. Encrypts an input string of characters and prints the result.
8. This program calculates the total distance a ball travels as it bounces given:
    a. the initial height of the ball
    b. its bounciness index

c. the number of times the ball is allowed to continue bouncing

9. Predict population growth, assuming that no organisms die.
    a. Inputs:
        i. the initial number of organisms
        ii. rate of growth (a float > 1)
        iii. the number of hours to achieve the rate
        iv. number of hours of growth
10. This program approximates the value of pi using an algorithm designed by the German mathematician Gottfried Leibniz. The algorithm is as follows:

    pi = 4 - 4 / 3 + 4 / 5 - 4 / 7 + . . .

This program allows the user to specify the number of iterations to use in the approximation.

11. Compute a school district's salary schedule.
    a. Inputs
        i. starting salary
        ii. annual percentage increase
        iii. number of years for which to print the schedule
    b. Outputs
        i. Two columns are containing the year and the salary after the increase.
12. Encrypts an input string of the ASCII characters and prints the result. The other input is the distance value.
13. Decrypts an input string character and prints the result. The other input is the distance value. (The ASCII values range from 0 through 127)
14. Converts a string of octal digits to a decimal integer.
15. Decrypts an input string of characters and prints the result.

Chapter 5 Arrays

Chapter 2 explains about variable: type, a value such as $a = 1$, that means variable a is assigned with value 1. This chapter is about arrays, which explain the collection of value in one variable. There are four types of arrays in the Python programming language that will describe in this chapter.

Table 7 Feature of List, Tuple, Set, and Dictionary

	Ordered	Changeable	Item duplicate
List	✓	✓	✓
Tuple	✓	✗	✓
Set	✗	✗	✗
Dictionary	✗	✓	✗

The important thing for a programmer is to select a suitable type of array for any application. It will have an impact on not only the efficiency of the program, but also it will influence the security of the program too.

5.1 Lists

A list in Python can be created as the example in Figure 77, which is in **square brackets.**

Code

```
thislist = ["apple", "banana", "cherry"]
print(thislist)
```

Result

```
C:\Users\LENOVO\AppData\Local\
['apple', 'banana', 'cherry']
```

Figure 77 Example of creating a list

5.1.1 Access Items

The list can access each item inside by indexing a number. For example, Figure 78 gives an example to access the second item of variable “thislist” which index by “1” because the program starts at 0.

Code Result

```
thislist = ["apple", "banana", "cherry"]
print(thislist[1])
```

```
C:\Users\
banana
```

Figure 78 Example of access item in the list

5.1.2 Negative Indexing

We also can access the item by using a negative index. However, the negative index will not begin from the state, but it counts from the end of the array variable. Figure 79 using "-1" to index the item, the result will display the first value from the last (cherry). If using "-2" as the index, the output will show "banana."

Code

```
thislist = ["apple", "banana", "cherry"]
print(thislist[-1])
```

Result

```
C:\Users
cherry
```

Figure 79 Example of using negative index

5.1.3 Range of Indexes

Besides, we can access the list items only specific range that we want. For instance, display only the list from index number 2 to index number 5 (exclude 5) as Figure 80.

Code

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[2:5])
```

Result

```
C:\Users\LENOVO\AppData\Local
['cherry', 'orange', 'kiwi']
```

Figure 80 Example of using a range of index

5.1.4 Range of Negative Indexes

Same as the previous one, the range of index also can use with the negative index. For example, display the items from index "-4" to index "-1" (exclude) as Figure 81.

Code

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[-4:-1])
```

Result

```
C:\Users\LENOVO\AppData\Loc
['orange', 'kiwi', 'melon']
```

Figure 81 Example of using a range of negative index

5.1.5 Change Item Value

As mention above, the list is changeable. The method to change is referring to a specific item that would like to change and re-assigned a value to the item again, as Figure 82, which changes the item form “banana” to “blackcurrant.”

Code

```
thislist = ["apple", "banana", "cherry"]
thislist[1] = "blackcurrant"
print(thislist)
```

Result

```
C:\Users\LENOVO\AppData\Local\Progr
['apple', 'blackcurrant', 'cherry']
```

Figure 82 Example of Change Item Value

5.1.6 Loop Through a List

We can do the iteration through a list. Figure 83 shows the example of for loop with “thislist” to print each item one by one.

Code

```
thislist = ["apple", "banana", "cherry"]
for x in thislist:
  print(x)
```

Result

```
C:\User
apple
banana
cherry
```

Figure 83 Example of Loop Through a List

5.1.7 Check if Item Exists

Membership operator (in) also can apply to determine if a specified item is present in a list as Figure 84 which check whether or not “apple” is in “thislist”

Code

```
thislist = ["apple", "banana", "cherry"]
if "apple" in thislist:
  print("Yes, 'apple' is in the fruits list")
```

Result

```
C:\Users\LENOVO\AppData\Local\Prog
Yes, 'apple' is in the fruits list
```

Figure 84 Example of Check if Item Exists

5.1.8 List Length

List length can determine by syntax len() as Figure 85.

Code

```
thislist = ["apple", "banana", "cherry"]
print(len(thislist))
```

Result

```
C:\U
3
```

Figure 85 Example to get List Length

5.1.9 Add Items

append() is an extensive method that use to add the item in to list by adding as the last item of list as Figure 86

Code

```
thislist = ["apple", "banana", "cherry"]
thislist.append("orange")
print(thislist)
```

Result

```
C:\Users\LENOVO\AppData\Local\Programs\
['apple', 'banana', 'cherry', 'orange']
```

Figure 86 Example of Add Items in the list by append()

Besides, we also can use insert() to add an item at the specified index as Figure 87.

Code

```
thislist = ["apple", "banana", "cherry"]
thislist.insert(1, "orange")
print(thislist)
```

Result

```
C:\Users\LENOVO\AppData\Local\Programs\I
['apple', 'orange', 'banana', 'cherry']
```

Figure 87 Example of Add Items in the list by insert()

5.1.10 Remove Item

The remove() method can remove the specified item as Figure 88.

Code

```
thislist = ["apple", "banana", "cherry"]
thislist.remove("banana")
print(thislist)
```

Result

```
C:\Users\LENOVO\Appl
['apple', 'cherry']
```

Figure 88 Example of Remove Item from the list by remove()

The pop() method also can remove the specified index or remove the last item if it is not a specific item number as Figure 89.

Code

Result

```
thislist = ["apple", "banana", "cherry"]
thislist.pop()
print(thislist)
```

```
C:\Users\LENOVO\App
['apple', 'banana']
```

Figure 89 Example of Remove Item from the list by pop()

Another keyword that can delete a specified item is del as Figure 90.

Code

```
thislist = ["apple", "banana", "cherry"]
del thislist[0]
print(thislist)
```

Result

```
C:\Users\LENOVO\AppD
['banana', 'cherry']
```

Figure 90 Example of Remove Item from the list by del (1)

The del keyword also can completely delete the list array as Figure 91.

Code

```
thislist = ["apple", "banana", "cherry"]
del thislist
print(thislist)
```

Result

```
print(thislist)
NameError: name 'thislist' is not defined
```

Figure 91 Example of Remove Item from the list by del (2)

Except for the delete operator, Python has a clear() operator to delete all items but remain the list array as empty as Figure 92.

Code

```
thislist = ["apple", "banana", "cherry"]
thislist.clear()
print(thislist)
```

Result

```
C:\Users
[]
```

Figure 92 Example of the clear item in the list

5.1.11 Copy a List

To copy a list in python, we cannot use assignment operator like equal (=) such as list2 = list1 because it means list2 will only be a reference to list1, and changes made in list1 will automatically also be made in list2. Python has a built-in function that is copy() to do copy tasks for the list as Figure 93.

Code

```
thislist = ["apple", "banana", "cherry"]
mylist = thislist.copy()
print(mylist)
```

Result

```
C:\Users\LENOVO\AppData\Local\
['apple', 'banana', 'cherry']
```

Figure 93 Example using copy() method in the list

Another method to copy a list is the syntax list(), which is a built-in function in Python as the example in Figure 94.

Code

```
thislist = ["apple", "banana", "cherry"]
mylist = list(thislist)
print(mylist)
```

Result

```
C:\Users\LENOVO\AppData\Local'
['apple', 'banana', 'cherry']
```

Figure 94 Example of using list() method to copy the list

5.1.12 Join Two Lists

Three methods that will present in this section to join or concatenate two or more lists in Python. The first one is + operator, which has the example of using it as Figure 95 by joining "list1" and "list2," resulting in "list3".

Code

```
list1 = ["a", "b" , "c"]
list2 = [1, 2, 3]

list3 = list1 + list2
print(list3)
```

Result

```
C:\Users\LENOVO\AppData\
['a', 'b', 'c', 1, 2, 3]
```

Figure 95 Example of + operator to join lists

The appending method is another way to use for joining lists by using syntax "append" as Figure 96.

Code

```
list1 = ["a", "b" , "c"]
list2 = [1, 2, 3]

for x in list2:
  list1.append(x)

print(list1)
```

Result

```
C:\Users\LENOVO\AppData\
['a', 'b', 'c', 1, 2, 3]
```

Figure 96 Example of append() to join lists

The same concept with "append," the syntax "extend" also can use for joining lists as Figure 97.

Code

Result

```
list1 = ["a", "b" , "c"]
list2 = [1, 2, 3]

list1.extend(list2)
print(list1)
```

```
C:\Users\LENOVO\AppData\
['a', 'b', 'c', 1, 2, 3]
```

Figure 97 Example of extend() to join lists

5.1.13 The list() Constructor

We also can use list() to be constructor the new list like Figure 98.

Code

```
thislist = list(("apple", "banana", "cherry"))
print(thislist)
```

Result

```
C:\Users\LENOVO\AppData\Local'
['apple', 'banana', 'cherry']
```

Figure 98 Example of using list() constructor

5.2 Tuple

A tuple has a different from list that tuple an unchangeable. However, we can convert a tuple to be list before doing a change process, which will mention later. In Python, Tuples are written with **round brackets,** as shown in Figure 99.

Code

```
thistuple = ("apple", "banana", "cherry")
print(thistuple)
```

Result

```
C:\Users\LENOVO\AppData\Local
('apple', 'banana', 'cherry')
```

Figure 99 Example of Tuple

5.2.1 Access Tuple Items

The way to access tuple items is the same as in a list that can specify the number of items that would like to access by the first item is 0 by default.

Code

```
thistuple = ("apple", "banana", "cherry")
print(thistuple[1])
```

Result

```
C:\Users\
banana
```

Figure 100 Access item of the tuple

5.2.2 Negative Indexing

The last item in tuple refers to index “-1”, the second item from the last refers to index “-2”, and so on.

Code

```
thistuple = ("apple", "banana", "cherry")
print(thistuple[-1])
```

Result

```
C:\Users\
cherry
```

Figure 101 Using negative indexing in the tuple

5.2.3 Range of Indexes

To determine the range of index in the tuple is the same within the list that Figure 102 shows the result of a program that displays item 2 to item 5 (exclude).

Code

```
thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")
print(thistuple[2:5])
```

Result

```
C:\Users\LENOVO\AppData\Loca
('cherry', 'orange', 'kiwi')
```

Figure 102 Range of Indexes in the tuple

5.2.4 Range of Negative Indexes

The range of negative index in tuple also has the same concept as that of the list. Figure 103 shows the program that displays by using a negative index from “-4” to “-1” (exclude).

Code

```
thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")
print(thistuple[-4:-1])
```

Result

```
C:\Users\LENOVO\AppData\Loc
('orange', 'kiwi', 'melon')
```

Figure 103 Range of Negative Indexes in the tuple

5.2.5 Change Tuple Values

Tuple cannot change the item. However, we can convert the tuple to list first and do a change with a list and convert back into a tuple again as Figure 104.

Code

```
x = ("apple", "banana", "cherry")
y = list(x)
y[1] = "kiwi"
x = tuple(y)
print(x)
```

Result

```
C:\Users\LENOVO\AppData\Loc
('apple', 'kiwi', 'cherry')
```

Figure 104 Change Tuple Values

5.2.6 Loop Through a Tuple

Figure 105 presents using of for loop over tuple to print each item one by one.

Code

```
thistuple = ("apple", "banana", "cherry")
for x in thistuple:
  print(x)
```

Result

```
C:\Users\
apple
banana
cherry
```

Figure 105 Loop Through a Tuple

5.2.7 Check if Item Exists

Same as in a list, tuple also can check an item is exists or not by using membership operator in.

Code

```
thistuple = ("apple", "banana", "cherry")
if "apple" in thistuple:
  print("Yes, 'apple' is in the fruits tuple")
```

Result

```
C:\Users\LENOVO\AppData\Local\Prograı
Yes, 'apple' is in the fruits tuple
```

Figure 106 Check if Item Exists in the tuple

5.2.8 Tuple Length

To obtain the length of the tuple, we can use len(), as demonstrated in Figure 107.

Code

```
thistuple = ("apple", "banana", "cherry")
print(len(thistuple))
```

Result

```
C:\U
3
```

Figure 107 Obtaining Tuple Length

5.2.9 Add Items

Since tuple cannot change, therefore, it is not possible to add an item into a tuple.

Code

```
thistuple = ("apple", "banana", "cherry")
thistuple[3] = "orange" # This will raise an error
print(thistuple)
```

Result

```
  thistuple[3] = "orange" # This will raise an error
TypeError: 'tuple' object does not support item assignment
```

Figure 108 Add items to a tuple

5.2.10 Create Tuple With One Item

If we would like to create tuple by that tuple has only one item, it has to add a comma at the end to indicate that this variable is a tuple. Otherwise, the program will interpret that variable to be a string, as shown in Figure 109.

Code

```
thistuple = ("apple",)
print(type(thistuple))

#NOT a tuple
thistuple = ("apple")
print(type(thistuple))
```

Result

```
C:\Users\LENOVO\
<class 'tuple'>
<class 'str'>
```

Figure 109 Create tuple with one Item

5.2.11 Remove Items

Remember that tuple cannot change. Therefore, we cannot remove an item from the tuple. However, we can completely delete tuple as Figure 110.

Code

```
thistuple = ("apple", "banana", "cherry")
del thistuple
print(thistuple) #this will raise an error because the tuple no longer exists
```

Result

```
    print(thistuple) #this will raise an error because the tuple no longer exists
NameError: name 'thistuple' is not defined
```

Figure 110 Remove items in the tuple

5.2.12 Join Two Tuples

The simple way to join two or more tuples is + operator like Figure 111.

Code

```
tuple1 = ("a", "b" , "c")
tuple2 = (1, 2, 3)

tuple3 = tuple1 + tuple2
print(tuple3)
```

Result

```
C:\Users\LENOVO\AppData\
('a', 'b', 'c', 1, 2, 3)
```

Figure 111 Join Two Tuples

5.2.13 The tuple() Constructor

Same with a list, we can use tuple() syntax to create the new tuple as Figure 112.

Code

```
thistuple = tuple(("apple", "banana", "cherry"))
print(thistuple)
```

Result

```
C:\Users\LENOVO\AppData\Local
('apple', 'banana', 'cherry')
```

Figure 112 The tuple() Constructor

5.3 Set

In Python, sets are written with **curly brackets**. Figure 113 present creating set and print it out. Please note that sets are unordered. Thus it is not determining the order of items that will appear.

Code

```
thisset = {"apple", "banana", "cherry"}
print(thisset)
```

Result

```
C:\Users\LENOVO\AppData\Local\
{'apple', 'banana', 'cherry'}
```

Figure 113 Creating set

5.3.1 Access Items

Since sets cannot orders, thus it is also cannot be indexing because the item has no index. Therefore, the loop displays not in order in Figure 114. However, we can specifically value or check membership is set the same as list and tuple as Figure 115.

Code

```
thisset = {"apple", "banana", "cherry"}

for x in thisset:
  print(x)
```

Result

```
C:\Users
banana
apple
cherry
```

Figure 114 Access Items of Set (1)

Example, check if "banana" is present in the set:

Code

```
thisset = {"apple", "banana", "cherry"}

print("banana" in thisset)
```

Result

```
C:\Users\
True
```

Figure 115 Access Items of Set (2)

5.3.2 Change Items

Sets cannot change, but they can add new items.

5.3.3 Add Items

Two methods to add items to set is by syntax add() and update(). The first one uses when to add item only one item into a set, while update() use for adding more than two items into a set as Figure 116 and Figure 117, respectively.

Code

```
thisset = {"apple", "banana", "cherry"}
thisset.add("orange")
print(thisset)
```

Result

```
C:\Users\LENOVO\AppData\Local\Programs\
{'apple', 'orange', 'banana', 'cherry'}
```

Figure 116 Add Items into the set by add()

Code

```
thisset = {"apple", "banana", "cherry"}
thisset.update(["orange", "mango", "grapes"])
print(thisset)
```

Result

```
{'orange', 'banana', 'apple', 'grapes', 'mango', 'cherry'}
```

Figure 117 Add Items into the set by update()

5.3.4 Get the Length of a Set

Same as in list and tuple, to get the length of the set, we can use syntax len() as Figure 118.

Code

```
thisset = {"apple", "banana", "cherry"}
print(len(thisset))
```

Result

```
3
```

Figure 118 Get the Length of a Set

5.3.5 Remove Item

Two methods to remove an item in the set is remove() and discard(). Both methods need to a specific item, if the item does not exist, the program will raise an error in case of remove() but in case of discard() will not raise an error.

Code

```
thisset = {"apple", "banana", "cherry"}
thisset.remove("banana")
print(thisset)
```

Result

```
{'apple', 'cherry'}
```

Figure 119 Remove item in a set by remove()

Code

Result

```
thisset = {"apple", "banana", "cherry"}
thisset.discard("banana")
print(thisset)
```

```
{'cherry', 'apple'}
```

Figure 120 Remove item in a set by discard()

The pop() method is allowed to use for remove items in the set, but pop() method will remove the last item which sets are unordered. Therefore, we cannot know which item will be removed. Figure 121 shows the result of using pop() to remove an item in sets.

Code

```
thisset = {"apple", "banana", "cherry"}
x = thisset.pop()
print(x)
print(thisset)
```

Result

```
banana
{'cherry', 'apple'}
```

Figure 121 Remove item in a set by pop()

Another command that can be used with the set is clear(), which is used for clear all items in the sets as Figure 122.

Code

```
thisset = {"apple", "banana", "cherry"}
thisset.clear()
print(thisset)
```

Result

```
set()
```

Figure 122 Clear item in the set

Same as a previous array type, del syntax can apply for completely delete the set as Figure 123.

Code

```
thisset = {"apple", "banana", "cherry"}
del thisset
print(thisset)
```

Result

```
  print(thisset)
NameError: name 'thisset' is not defined
```

Figure 123 Completely delete set

5.3.6 Join Two Sets

Two methods for joining two sets, the first one is union(), and the second one is update() method. Please note that both two methods will not have duplicate items.

Code

```
set1 = {"a", "b", "c"}
set2 = {1, 2, 3}
set3 = set1.union(set2)
print(set3)
```

Result

```
{'b', 'a', 1, 2, 'c', 3}
```

Figure 124 Join Two Sets by union()

Code

```
set1 = {"a", "b", "c"}
set2 = {1, 2, 3}
set1.update(set2)
print(set1)
```

Result

```
{1, 2, 3, 'a', 'b', 'c'}
```

Figure 125 Join Two Sets by update()

5.3.7 The set() Constructor

It is also possible to use the set() constructor to create a new set as Figure 126.

Code

```
thisset = set(("apple", "banana", "cherry"))
print(thisset)
```

Result

```
{'banana', 'cherry', 'apple'}
```

Figure 126 The set() Constructor

5.4 Dictionary

In Python, dictionaries are written with **curly brackets**, and they have **keys and values**. Figure 127 is the example of creating a dictionary that has keys (“brand,” “model,” “year”), and values (“Ford,” “Mustang,” 1964).

Code

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(thisdict)
```

Result

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
```

Figure 127 Creating dictionary

5.4.1 Accessing Items

Dictionary has a different way to access the item from the previous three kinds of the array. Dictionary needs to a specific key in square brackets to access the item, which other use numbers.

Code

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(thisdict)

x = thisdict["model"]
print(x)
```

Result

```
{'model': 'Mustang', 'brand': 'Ford', 'year': 1964}
Mustang
```

Figure 128 Accessing Items of Dictionary

Figure 128 result is the first line that comes from print all key and value in “thisdict,” which it is unordered. The second line of the result is from a specific key equal “model,” thus, the return value is “Mustang.” Another syntax to access the dictionary is get() method presenting the example in Figure 129.

Code

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(thisdict)

x = thisdict["model"]
print(x)

x = thisdict.get("model")
print(x)
```

Result

```
{'year': 1964, 'model': 'Mustang', 'brand': 'Ford'}
Mustang
Mustang
```

Figure 129 Accessing Items of Dictionary by using get()

5.4.2 Change Values

We can do a change in the dictionary by referring to its key. For example, change the value of key “year” from 1964 to be 2018 as Figure 130.

Code

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict["year"] = 2018
print(thisdict["year"])
```

Result

```
2018
```

Figure 130 Change values in the dictionary

5.4.3 Loop Through a Dictionary

As mention before, the dictionary consists of key and value which each key and value we called item. Once we de loop through a dictionary should be careful about which one we want to get from the loop; key, value, or item. Figure 131 shows a loop to obtain a key to the dictionary.

Code

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
for x in thisdict:
  print(x)
```

Result

```
brand
year
model
```

Figure 131 Loop to get key of the dictionary

Figure 132 presents the example of the loop to get the value of the dictionary.

Code

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
for x in thisdict:
  print(thisdict[x])
```

Result

```
Ford
1964
Mustang
```

Figure 132 Loop to get the value of dictionary (1)

Figure 133 presents another method to get the value of the dictionary by using values().

Code

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
for x in thisdict.values():
  print(x)
```

Result

```
Ford
1964
Mustang
```

Figure 133 Loop to get the value of dictionary (2)

Figure 134 shows the loop that gets both key and value or item of the dictionary.

Code

Result

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
for x, y in thisdict.items():
  print(x, y)
```

```
year 1964
brand Ford
model Mustang
```

Figure 134 Loop to get an item of dictionary

5.4.4 Check if Key Exists

We can use *in* operator to check whether or not that a specific key exists in a dictionary.

Code

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
if "model" in thisdict:
  print("Yes, 'model' is one of the keys in the thisdict dictionary")
```

Result

```
Yes, 'model' is one of the keys in the thisdict dictionary
```

Figure 135 Check if Key Exists in the dictionary

5.4.5 Dictionary Length

The length of the dictionary or how many items (key and value) in a dictionary can get by using len() method the same as other arrays.

Code

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(len(thisdict))
```

Result

```
3
```

Figure 136 Dictionary Length

5.4.6 Adding Items

Adding an item or adding key-value pair can be done by using a new index key and assign with value to that new key as Figure 137.

Code

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict["color"] = "red"
print(thisdict)
```

Result

```
{'color': 'red', 'model': 'Mustang', 'brand': 'Ford', 'year': 1964}
```

Figure 137 Adding items in a dictionary

5.4.7 Removing Items

Figure 138 presents using pop() method with a specific key to remove the item.

Code

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.pop("model")
print(thisdict)
```

Result

```
{'brand': 'Ford', 'year': 1964}
```

Figure 138 Removing items in a dictionary by using pop()

Figure 139 show another method that is the popitem() method to remove the last inserted item (in versions before 3.7, a random item is removed instead):

Code Result

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.popitem()
print(thisdict)
```

```
{'model': 'Mustang', 'year': 1964}
```

Figure 139 Removing items in the dictionary by using popitem()

The del method also can remove a specific item by using a key to indicate the item as Figure 140.

Code

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
del thisdict["model"]
print(thisdict)
```

Result

```
{'year': 1964, 'brand': 'Ford'}
```

Figure 140 Removing items in a dictionary by using del

Figure 141 shows that the del method can completely delete the dictionary the same with other previous three arrays.

Code

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
del thisdict
print(thisdict) #this will cause an error because "thisdict" no longer exists.
```

Result

```
    print(thisdict) #this will cause an error because "thisdict" no longer exists.
NameError: name 'thisdict' is not defined
```

Figure 141 Completely delete dictionary by del

Figure 142 shows clearing all items in the dictionary by using the clear() method.

Code

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.clear()
print(thisdict)
```

Result

```
{}
```

Figure 142 Clear items in a dictionary

5.4.8 Copy a Dictionary

Same as a list, we cannot use a simple method like dict2 = dict1 to make a copy of dictionary because dict2 will only be a reference to dict1, and all changes made in dict1 will automatically also be made in dict2. Two methods to copy the dictionary is copy() method and dict() method, as doing in Figure 143 and Figure 144, respectively.

Code

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
mydict = thisdict.copy()
print(mydict)
```

Result

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
```

Figure 143 Copy a dictionary by copy() method

Code

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
mydict = dict(thisdict)
print(mydict)
```

Result

```
{'model': 'Mustang', 'brand': 'Ford', 'year': 1964}
```

Figure 144 Copy a dictionary by dict() method

5.4.9 Nested Dictionaries

A nested dictionary is a dictionary that contains many dictionaries inside. Figure 145 shows the nested dictionary by “myfamily” is the main or outer dictionary, has “child1”, “child2”, “child3” are key. While “child1”, “child2”, “child3” are also dictionaries by having “name,” “year” are the key.

Code

```
myfamily = {
  "child1" : {
    "name" : "Emil",
    "year" : 2004
  },
  "child2" : {
    "name" : "Tobias",
    "year" : 2007
  },
  "child3" : {
    "name" : "Linus",
    "year" : 2011
  }
}
print(myfamily)
```

Result

```
{'child1': {'year': 2004, 'name': 'Emil'}, 'child3': {'year': 2011, 'name': 'Linus'}, 'child2': {'year': 2007, 'name': 'Tobias'}}
```

Figure 145 Example of the nested dictionary (1)

Figure 146 shows the easy way the create a nested dictionary which can easy to read and make understanding.

Code

```
child1 = {
  "name" : "Emil",
  "year" : 2004
}
child2 = {
  "name" : "Tobias",
  "year" : 2007
}
child3 = {
  "name" : "Linus",
  "year" : 2011
}
myfamily = {
  "child1" : child1,
  "child2" : child2,
  "child3" : child3
}
print(myfamily)
```

Result

```
{'child1': {'year': 2004, 'name': 'Emil'}, 'child2': {'year': 2007, 'name': 'Tobias'}, 'child3': {'year': 2011, 'name': 'Linus'}}
```

Figure 146 Example of the nested dictionary (2)

5.4.10 The dict() Constructor

The dict() syntax can be used as a constructor to make a new dictionary as Figure 147.

Code

```
thisdict = dict(brand="Ford", model="Mustang", year=1964)
# note that keywords are not string literals
# note the use of equals rather than colon for the assignment
print(thisdict)
```

Result

```
{'model': 'Mustang', 'brand': 'Ford', 'year': 1964}
```

Figure 147 The dict() Constructor

Chapter 6 Functions

The function can reduce the time of a programmer to write a program that needs to do the routine task. They can write a function and call it for processing when need and the result of the function will return to the main program for further processing. In a computer programming language, the function has five components, which inhere give an example in Python language as in Figure 148.

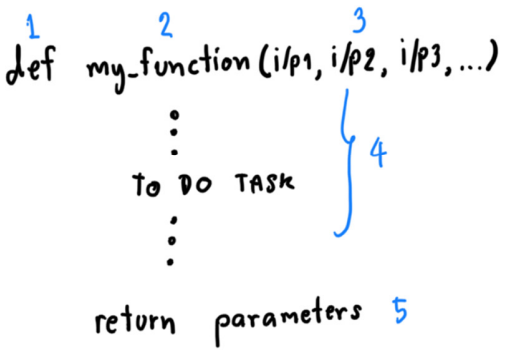


Figure 148 Component of function

1.) def (for define, this is a function)

2.) function name

3.) input parameter

4.) to do tasks

5.) return parameters

6.1 Creating a Function

The def keyword is used for defining function as Figure 149.

```
def my_function():
  print("Hello from a function")
```

Figure 149 Use def to define a function

6.2 Calling a Function

To call a function, type the function name followed by a parenthesis and may have an input parameter for processing in the function (optional). Figure 150, the last code line calls a function named “my_function(),” resulting in having a process function “my_function” and print the sentence.

Code

```
def my_function():
  print("Hello from a function")

my_function()
```

Result

```
Hello from a function
```

Figure 150 Calling a function

6.3 Parameters

As mention above, we can put the input for processing in a function (optional). It can put many parameters in the parentheses by separate them with a comma. Figure 151 shows that “my_function” uses the input from the main program that the first input is “Emil,” the second is “Tobias,” and the third is “Linus.”

Code

```
def my_function(fname):
  print(fname + " Refsnes")

my_function("Emil")
my_function("Tobias")
my_function("Linus")
```

Result

```
Emil Refsnes
Tobias Refsnes
Linus Refsnes
```

Figure 151 Parameters in function

6.4 Default Parameter Value

It can set default parameter value for a function. If the function is called without an input parameter, it will use the default one. Figure 152 shows a setting the default value is “Norway,” and it will display when no input from the main program.

Code

```
def my_function(country = "Norway"):
  print("I am from " + country)

my_function("Sweden")
my_function("India")
my_function()
my_function("Brazil")
```

Result

```
I am from Sweden
I am from India
I am from Norway
I am from Brazil
```

Figure 152 Default Parameter Value

6.5 Passing a List as a Parameter

The parameter can be any type such as string, list, dictionary, number, and so on). Figure 153 gives an example to use a list as an input parameter for a function.

Code

```
def my_function(food):
  for x in food:
    print(x)
fruits = ["apple", "banana", "cherry"]

my_function(fruits)
```

Result

```
apple
banana
cherry
```

Figure 153 Passing a List as a Parameter

6.6 Return Values

A function can return value by using return syntax following the value that would like to return as Figure 154.

Code

Result

```
def my_function(x):
  return 5 * x

print(my_function(3))
print(my_function(5))
print(my_function(9))
```

```
15
25
45
```

Figure 154 Return values from a function

6.7 Keyword Arguments

Keyword arguments can overcome the ordering of input parameter because it is specified by key and value as Figure 155.

Code

```
def my_function(child3, child2, child1):
  print("The youngest child is " + child3)

my_function(child1 = "Emil", child2 = "Tobias", child3 = "Linus")
```

Result

```
The youngest child is Linus
```

Figure 155 Keyword arguments for a function

6.8 Arbitrary Arguments

Sometimes we do not know the exact number of parameters. In this case, it can add a * before the parameter name inside parenthesis after the function name, as shown in Figure 156. Then, it can receive a tuple of arguments and can access the items accordingly.

Code

```
def my_function(*kids):
  print("The youngest child is " + kids[2])

my_function("Emil", "Tobias", "Linus")
```

Result

```
The youngest child is Linus
```

Figure 156 Arbitrary arguments in function

Exercise

1. Defines functions to compute the mean, median, and mode of a list of numbers.
2. Generates and displays sentences using simple grammar and vocabulary. Words are chosen at random. Adds optional adjectives to noun phrases, optional prepositional phrases, and optional independent clauses connected by conjunctions.
3. Defines a function repToDecimal that converts a number in any base to decimal.
4. Defines a function decimalToRep that converts a number in decimal to a number in a given base.
5. Conducts an interactive session of nondirective psychotherapy. Fixes problem of responding to sentences that address the doctor using second-person pronouns.
6. Conducts an interactive session of nondirective psychotherapy. Adds a history list of earlier patient sentences, which can be chosen for replies to shift the conversation to an earlier topic.
7. Compute the square root of a number (uses function with loop).
    a. The input is a number, or enter/return to halt the input process.
    b. The outputs are the program's estimate of the square root using Newton's method of successive approximations, and Python's own estimate using math.sqrt.
8. Compute the square root of a number (uses recursive function).
    a. The input is a number, or enter/return to halt the input process.
    b. The outputs are the program's estimate of the square root using Newton's method of successive approximations, and Python's own estimate using math.sqrt.
9. Compute the square root of a number (uses a recursive function with default second parameter).
    a. The input is a number, or enter/return to halt the input process.
    b. The outputs are the program's estimate of the square root using Newton's method of successive approximations, and Python's own estimate using math.sqrt.

10. Compute the square root of a number (uses a recursive function with default second argument).
    a. The input is a number, or enter/return to halt the input process.
    b. The outputs are the program's estimate of the square root using Newton's method of successive approximations, and Python's own estimate using math.sqrt.
11. Defines a predicate to test lists for being sorted.
12. Allows the user to visit all of the files in the current path and view them.
13. Defines the printAll function with a trace. Before each recursive call, the function creates a slice of its nonempty list argument. The hidden cost is that each slice produces a copy of the list, less its first item. This process requires time and memory.
14. Defines a function that behaves like Python's range function.

Chapter 7 Files Handling

In Python, there are several libraries to apply for file handling. For example, pandas which keen to handle .csv or .xls files and widely used in data science. In this chapter, the basic command for file handling will be explained.

7.1 File handling and open file syntax

First, we should know and understand the mode of handling files. The mode can be categorized as the following:

1) "r" - Read - Default value. Opens a file for reading, error if the file does not exist
2) "a" - Append - Opens a file for appending, creates the file if it does not exist
3) "w" - Write - Opens a file for writing, creates the file if it does not exist
4) "x" - Create - Creates the specified file, returns an error if the file exists

Also, it can specify if the file should be handled as binary or text mode

1) "t" - Text - Default value. Text mode
2) "b" - Binary - Binary mode (e.g. images)

The syntax to open the file is “open(file_name)” as Figure 157

```
f = open("demofile.txt")
```

Figure 157 Syntax to open file

The code in Figure 157 has the same result as Figure 158.

```
f = open("demofile.txt", "rt")
```

Figure 158 Syntax to open file with the specified mode

Here, "r" is for reading, and "t" is for text are the default values, which means it does not need to specify in the program. Besides, we should make sure the file exists; otherwise, it will get an error.

7.2 Read file and close file

7.2.1 Read all part of a file

Assume we have the following file (demofile.txt), located in the same folder wit Python file (code.py):

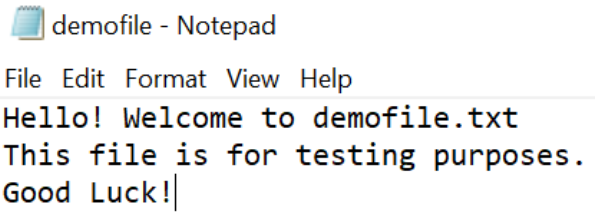


Figure 159 Example of demofile.txt located in the same location as .py

Figure 160 shows that example to read all file in demofile.txt by using open() syntax and "r" mode, and print it out in the final.

Code

```
f = open("demofile.txt", "r")
print(f.read())
```

Result

```
Hello! Welcome to demofile.txt
This file is for testing purposes.
Good Luck!
```

Figure 160 Read all part of the file

7.2.2 Read Only Parts of the File

However, if we would like to read only some part of the file, we can set how many characters that we need to read as Figure 161.

Code

```
f = open("demofile.txt", "r")
print(f.read(5))
```

Result

```
Hello
```

Figure 161 Read Only Parts of the File

7.2.3 Read Lines

The readline() method is used for reading line by line from "demofile.txt" as Figure 162.

Code

```
f = open("demofile.txt", "r")
print(f.readline())
```

Result

```
Hello! Welcome to demofile.txt
```

Figure 162 Read Lines (1)

The next, we calling readline() two times. It can print the first two-row in “demofile.txt” as Figure 163.

Code

```
f = open("demofile.txt", "r")
print(f.readline())
print(f.readline())
```

Result

```
Hello! Welcome to demofile.txt

This file is for testing purposes.
```

Figure 163 Read Lines (2)

If we loop through the lines of the file, we can read the whole file and print line by line, as shown in Figure 164.

Code

```
f = open("demofile.txt", "r")
for x in f:
  print(x)
```

Result

```
Hello! Welcome to demofile.txt

This file is for testing purposes.

Good Luck!
```

Figure 164 Read Lines (3)

7.2.4 Close Files

After an open file to read it and to do some process, it is good practice to close the file, as shown in Figure 165, to protect buffering changes made to a file.

```
f = open("demofile.txt", "r")
print(f.readline())
f.close()
```

Figure 165 Close files

7.3 Write/Create File

7.3.1 Write to an Existing File

This section base on the existing file, which is “demofile2.txt” and “demofile3.txt”. Both of file is empty. Figure 166 demonstrates how to append data to a file. In the first state, we have to open file with append mode ("a" - Append - will append to the end of the file) and write

something in the next line, then close it. After finish, check the result in the file by open and read it all part of it.

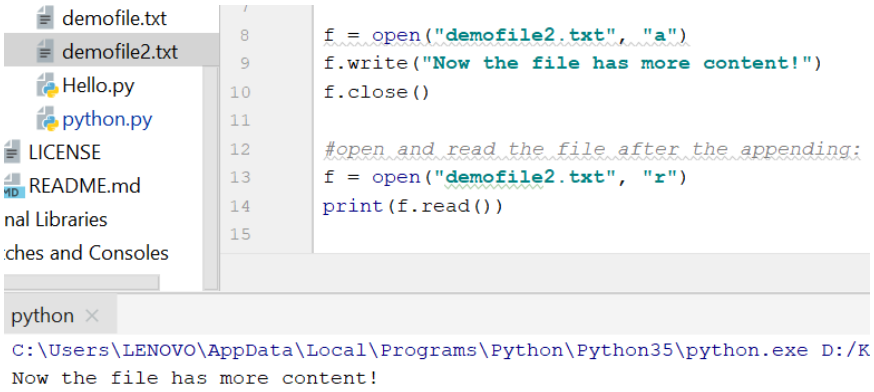


Figure 166 Append data to file

Figure 167 illustrates the use of write mode. Its processes are similar to Figure 166 but have different in the setting mode in the second line of the program, which in Figure 167 is written mode ("w" - Write - will overwrite any existing content). After the finish writes process, we can check its result by open and read all parts of the file.

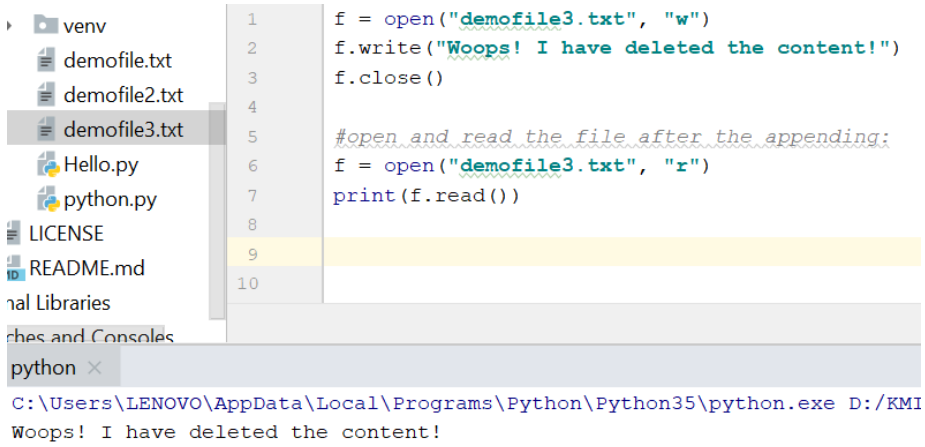


Figure 167 Write data to file

Now, let try both modes with “demofile2.txt” and “demofile3.txt” which do not empty.

7.3.2 Create a New File

To create a new file which does not exist in Python, use the open() method, with one of the following mode parameters:

"x" - Create - will create a file, returns an error if the file exists

"a" - Append - will create a file if the specified file does not exist

"w" - Write - will create a file if the specified file does not exist

Example, create a file called "myfile.txt":

```
f = open("myfile.txt", "x")
```

Figure 168 Create a new file with “x” mode

```
f = open("myfile.txt", "w")
```

Figure 169 Figure 170 Create a new file with “w” mode

7.4 Delete a File

To delete a file, it must import the OS module and run a program “os.remove(file_name)” as Figure 171, which deleting “demofile.txt.”

```
import os
os.remove("demofile.txt")
```

Figure 171 Delete a file

7.4.1 Check if file exists

Delete a file that can raise an error if the file does not exist. Therefore, we should check it existing first before deleting it as Figure 172.

Code

```
import os
if os.path.exists("demofile.txt"):
  os.remove("demofile.txt")
else:
  print("The file does not exist")
```

Result

```
The file does not exist
```

Figure 172 Check if file exists

7.4.2 Delete folder

Same as delete the file, it needs to import the OS module and run an os.rmdir() method to remove the folder "myfolder" as Figure 173. However, only empty folders can be deleted.

```
import os
os.rmdir("myfolder")
```

Figure 173 Delete folder

Exercise

1. Decrypts a file of encrypted text. And prints the result. The other input is the distance value.
2. Encrypts a text file. The inputs are the names of the input file and the output file and the distance value. The encrypted code is written to a new file.
3. Copies the text from a given input file to a given output file.
4. Copies the text from a given input file to a given output file, numbering them as it goes.
5. Determines whether or not the contents of two text files are the same. Outputs "Yes" if that is the case or "No" and the first two lines that differ if that is not the case.
6. Print a payroll report.
    a. Input
        i. A file in which each line has the form
        ii. <last name> <hourly wage> <hours worked>
    b. Output
        i. A report in tabular format. Each line has the form
        ii. <last name> <hours worked> <total wages>
7. Allows the user to navigate to any line in a text file.
8. Prints the unique words in a text file in alphabetical order.
9. Prints the unique words in a text file and their frequency.
10. Prints the average of the numbers in a text file

Note! Some text files need to create before the program.

References

[1] [Online]. Available: https://www.jetbrains.com/pycharm/download/#section=windows.

[2] [Online]. Available: https://www.python.org/downloads/.

[3] [Online]. Available: https://programminghistorian.org/en/lessons/getting-started-with-github-desktop#what-are-git-and-github.

[4] [Online]. Available: https://www.programiz.com/python-programming/operators.

[5] [Online]. Available: https://www.w3schools.com/python/python_for_loops.asp.

[6] [Online]. Available: http://www.pythonlearn.com/html-007/cfbook002.html.

[7] [Online]. Available: https://www.softwaretestinghelp.com/python-ide-code-editors/.

[8] [Online]. Available: https://desktop.github.com/.

[9] [Online]. Available: https://www.w3schools.com/python/.

[10] K. A. Lambert., Fundamentals of Python: First Programs (2nd. ed.)., Boston, MA, USA.: Course Technology Press, 2018.