



# Samsung Innovation Campus

| Coding and Programming

Together for Tomorrow!  
**Enabling People**

Education for Future Generations

Chapter 1.

# Programming Basic Concept and Starting Python

Coding and Programming

# Chapter Description

---

## Chapter objectives

- ✓ Learners will be able to configure a practice environment for Python and learn the basics of programming. Learners will become familiar with the foundations of Python such as input/output, data types, and variables. Furthermore, they will acquire problem solving skills in coding through various operators, conditionals, and loops.

## Chapter contents

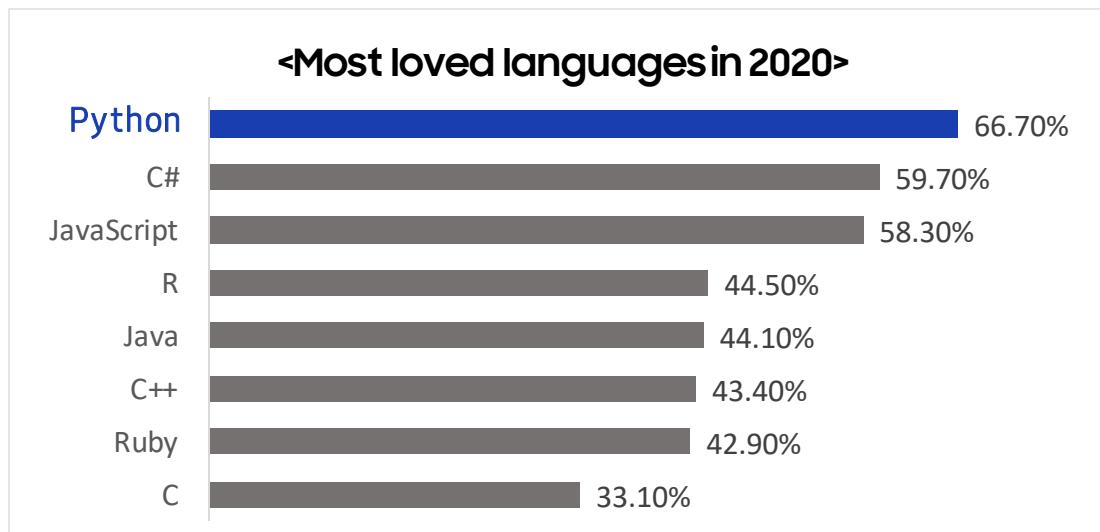
- ✓ Unit 1. Sequential Programming
- ✓ Unit 2. Planning for Programming
- ✓ Unit 3. Basic of Numeric Data Types and Arithmetic Operation
- ✓ Unit 4. Variables and Inputs
- ✓ Unit 5. Logic and Comparison Operations
- ✓ Unit 6. Conditional Statement-1: Conditions and Decision Making
- ✓ Unit 7. Conditional Statement-2: Making decisions in two directions and applying conditional statements
- ✓ Unit 8. Loop-1
- ✓ Unit 9. Loop-2

Orientation

# Introduction to Python

# What is Python?

- Python is a high-level programming language and is one of the most popular programming languages in the world.
- Python is a general purpose language that is broadly applicable across application domains. Therefore, Python is widely used to create a variety of different programs. It is used in everything from data science, web development, automation to generally getting stuff done.
- Because of its versatility, Python can be used by developers and non-developers alike.



<https://insights.stackoverflow.com/survey/2020>

## Why is Python so popular?

Python is also loved by major companies and applications because of its versatility.

- ▶ Python is wanted not only in educational fields but also by **many developers in professional workplaces**. Big companies, including Google, Intel, eBay, Netflix, Instagram, Dropbox, and Slack, utilize Python for developing and maintaining their applications.

### <Major companies that use Python>



- ▶ **Google** uses Python software development to enhance its search engine capabilities.
- ▶ **Netflix** uses Python to recommend movies, TV shows, and documentaries based on a user's history.
- ▶ **Instagram** uses Python to personalize the explore section for users.
- ▶ **Spotify** uses Python for personalized music recommendation to its users.
- ▶ **Reddit** uses Python for web development to simplify the Q&A sections.

<https://www.botreetechnologies.com/blog/pros-and-cons-of-python/>

## Why is Python so popular?

Many industries use Python as it offers open-source libraries that contain built-in modules.

- ▶ Python includes **various libraries**, such as pandas, NumPy, SciPy, Matplotlib, TensorFlow, and Keras, that can be extensively used in different fields like web development, data analytics, and artificial intelligence.



- ▶ **pandas** is used for data manipulation and analysis, particularly featuring numerical tables and time series operation.
- ▶ **NumPy** supports for large, multi-dimensional arrays and matrices, along with a collection of high-level mathematical functions.
- ▶ **TensorFlow** is based on dataflow and differential programming, which focuses on deep neural networks for machine learning.
- ▶ **Keras** provides an interface for artificial neural networks with a focus on enabling fast experimentation.

<https://light-it.net/blog/top-10-python-libraries-for-machine-learning/>

## What is Python used for?

Artificial Intelligence  
&  
Machine Learning

Data Science  
&  
Data Visualization

Web Development

Game Development

Automation

Automate the simple  
tasks on the computer

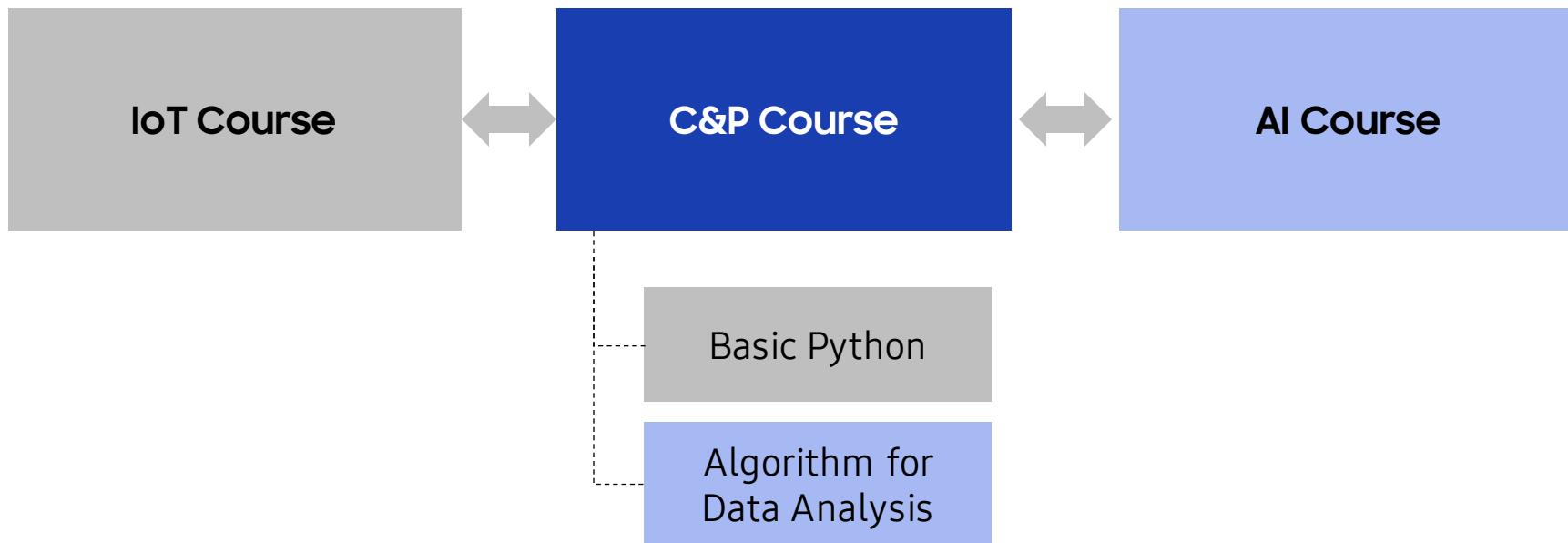
Software Testing and  
Prototyping

Simplifying everyday tasks  
through automation

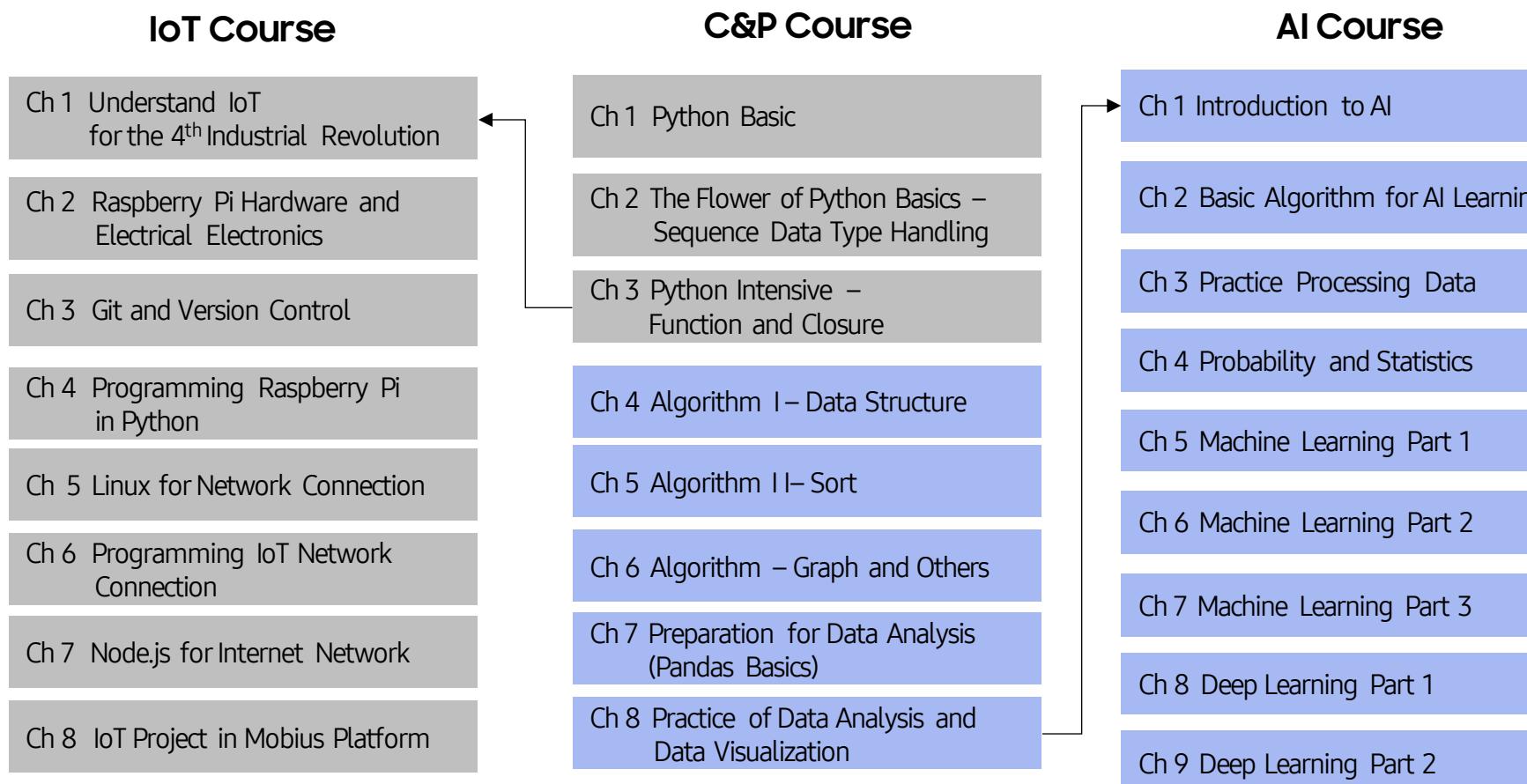
- ▶ Updating your grocery shopping list
- ▶ Renaming large batches of files
- ▶ Converting text files to spreadsheets
- ▶ Filling out online forms automatically and etc.

## How to use the C&P course?

- This course is designed that students can gain a strong foundation of Python. As the core of the SIC program, it is modularized to be effectively linked to the SIC's other courses: AI and IoT.
- By completing this course, you will be prepared to join the entry level job position that requires basic Python skill.



## How to use the C&P course?



Unit 1.

# **Sequential Programming**

## Learning objectives

- ✓ Learners will be able to explain the definition and necessity of computational thinking.
- ✓ Learners will be able to install necessary software for Python from scratch.
- ✓ Learners will be able to configure a new virtual environment where Python 3.x is installed, and will be able to install different versions of Python.
- ✓ Learners will be able to infer the coding sequence by looking at the output result using only the print function.
- ✓ Learners will be able to distinguish between syntax errors and runtime errors.

## 1. Sequence

### 1.1. Definition and Importance of Sequence

- | Sequence is the order in which a computer executes its codes.
- | Since the program is very fast, it may look like codes are being executed simultaneously, but in reality the computer processes commands in sequence.
- | Naturally, even with the same commands, if the order changes, then a completely different result is produced.
- | When writing a code in Python, the priority and purpose of the commands should be considered before deciding their order.

Sequence is important in everyday life

put toothpaste on toothbrush



use toothbrush to clean teeth



rinse toothbrush

How could the changed order affect the result?

rinse toothbrush



put toothpaste on toothbrush



use toothbrush to clean teeth

## 2. Basic Terminology

### 2.1. Using the Glossary

- A glossary is provided as a learning aid. If a new term appears, it is important to note them and check the definition.
- A tutorial on Python's official website is also recommended.

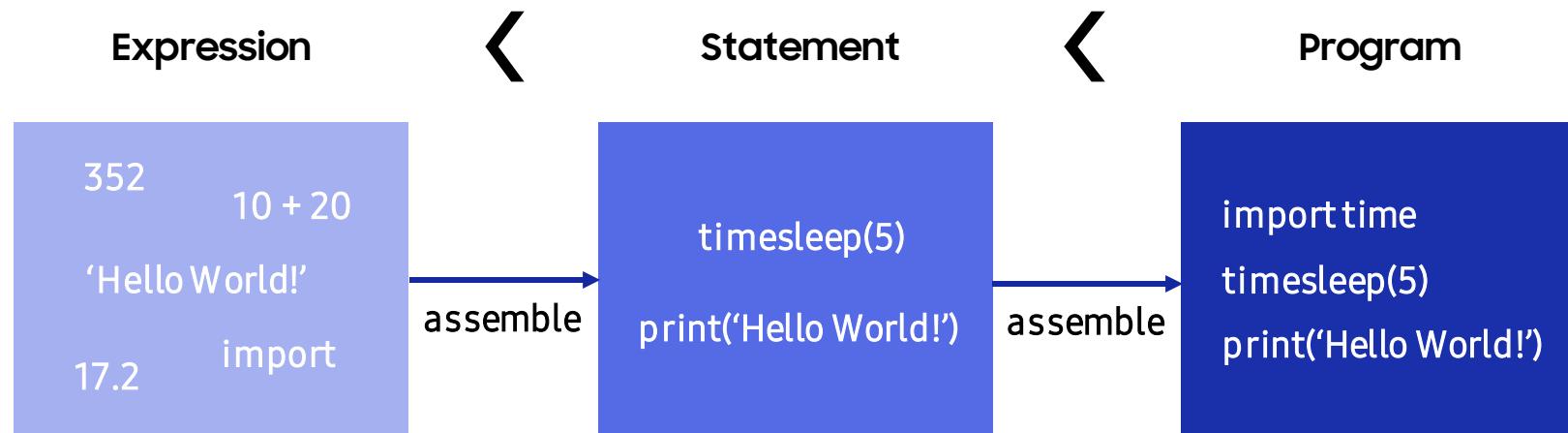
The screenshot shows the Python 3.9.7 documentation website. At the top, there is a navigation bar with links for "Python", "English", "3.9.7", "Documentation", "Quick search", "Go", "modules", and "index". On the left side, there is a sidebar with links for "Download", "Docs by version" (listing versions from 3.11 down to 2.7), and "Other resources" (listing PEP Index, Beginner's Guide, Book List, and AudioVisual Talks). The main content area has a title "Python 3.9.7 documentation" and a welcome message: "Welcome! This is the official documentation for Python 3.9.7." Below this, there is a section titled "Parts of the documentation:" with links to "What's new in Python 3.9?", "Tutorial" (with a "start here" link), "Library Reference" (with a "keep this under your pillow" link), and "Language Reference". To the right of these, there are two more sections: "Installing Python Modules" (with a link to "installing from the Python Package Index & other sources") and "Distributing Python Modules" (with a link to "publishing modules for installation by others").

<https://docs.python.org/3/>

## 2. Basic Terminology

### 2.2. Expression, Statement, and Program

Expression is a simple code that produces values such as numbers, formulas, and strings. A statement is composed of more than one expressions and a program is composed of more than one statement.



## 2. Basic Terminology

### 2.3. Annotation

- | Annotation is a sentence used to describe the function of the code in the program. The interpreter skips execution of the annotation.
- | Annotation is very important in programming. When others modify your codes, annotations help other programmers understand your code. For complex codes, comments are also helpful for future references.
- | There are two main ways to add annotations in Python. First, let's look at how to make a single line annotation.
- | Adding the # sign at the beginning of the sentence will turn the line into an annotation. You haven't learned the code below yet, but reading the comments will help you understand them.

```
1 #define the variable as a radius
2 radius = 4.0
3
4 #print the radius, area, and circumference of the circle
5 print('Radius', radius)
6 print('Area', 3.14 * radius * radius)      #Apply the formula to solve for the area of a circle
7 print('Circumference', 2.0 * 3.14 * radius) #Apply the formula to solve for the circumference of the circle
```

Radius 4.0  
Area 50.24  
Circumference 25.12

## 2. Basic Terminology

### 2.3. Annotation

- | To create multiple-line annotations, add three small or large quotation marks at the front and back lines.
- | For "~~" or "~~~" lines, ~~ portions are not executed.

1	'''
2	You can create a multiple-line annotations using double quotation marks.
3	
4	It allows you to make multiple lines of comments.
5	'''

1	"""
2	It allows you to make multiple lines of comments.
3	
4	It allows you to make multiple lines of comments.
5	"""

## 2. Basic Terminology

### 2.4. Reserved Word

- A word reserved to perform a predetermined role is called a keyword or a reserved word.
- Since keywords can only execute the predetermined role in Python, they cannot be used as identifiers such as variable name and class name. If used as an identifier, the role will overlap with the command and cannot execute.

False	class	finally	is	return
None	continue	for	lambda	try
True	def	from	nonlocal	while
and	del	global	not	with
as	elif	if	or	yield
assert	else	import	pass	
break	except	in	raise	

# 1. Setting Up

## 1.1. Installing Anaconda

- We will set up the workspace using the Anaconda. If this step is deemed unnecessary, the instructor can decide to skip to the next section. (But starting with the virtual environment configuration part, the lesson must be taught in order.)
- First, go to the Anaconda website ([www.anaconda.com](http://www.anaconda.com)) and download the Anaconda Individual Edition. Choose an installer compatible with your computer's operating system. (This lecture provides an example in the Windows system.)

The image shows two screenshots from the Anaconda website. On the left, the 'Individual Edition' page for the 'Your data science toolkit' is displayed. It features a green header bar, a large green button labeled 'Download' with a Windows icon, and a 'For Windows' section showing 'Python 3.8 • 64-Bit Graphical Installer • 477 MB'. Below this is a 'Get Additional Installers' section with icons for Windows, Mac, and Linux. A blue arrow points from this page to the right. On the right, the 'Anaconda Installers' page is shown, which lists installers for Windows, Mac OS, and Linux. Each platform has a section with links for Python 3.8 installers: Windows has '64-Bit Graphical Installer (477 MB)' and '32-Bit Graphical Installer (409 MB)'; Mac OS has '64-Bit Graphical Installer (440 MB)' and '64-Bit Command Line Installer (433 MB)'; Linux has '64-Bit (x86) Installer (544 MB)' and '64-Bit (Power8 and Power9) Installer (285 MB)'. At the bottom, a blue button contains the URL <https://www.anaconda.com/products/individual>.

Individual Edition

Your data science toolkit

With over 25 million users worldwide, the open-source Individual Edition (Distribution) is the easiest way to perform Python/R data science and machine learning on a single machine. Developed for solo practitioners, it is the toolkit that equips you to work with thousands of open-source packages and libraries.

Anaconda Individual Edition

Download

For Windows  
Python 3.8 • 64-Bit Graphical Installer • 477 MB

Get Additional Installers

Anaconda Installers

Windows

MacOS

Linux

Python 3.8  
64-Bit Graphical Installer (477 MB)  
32-Bit Graphical Installer (409 MB)

Python 3.8  
64-Bit Graphical Installer (440 MB)  
64-Bit Command Line Installer (433 MB)

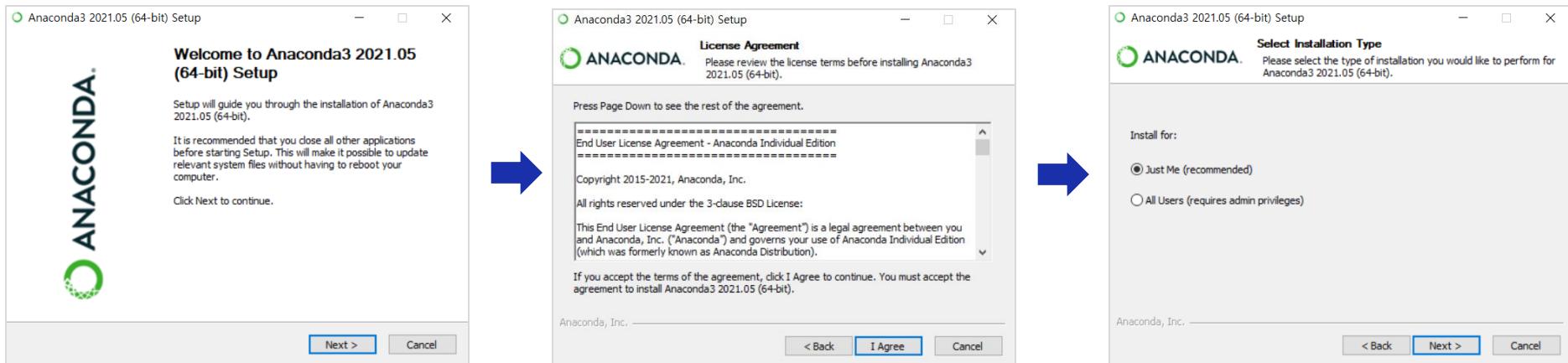
Python 3.8  
64-Bit (x86) Installer (544 MB)  
64-Bit (Power8 and Power9) Installer (285 MB)

<https://www.anaconda.com/products/individual>

# 1. Setting Up

## 1.1. Installing Anaconda

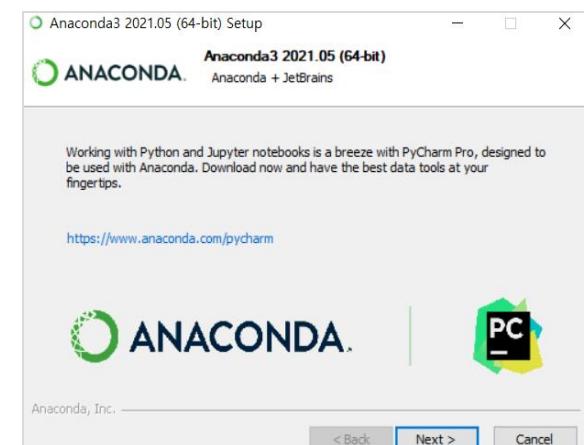
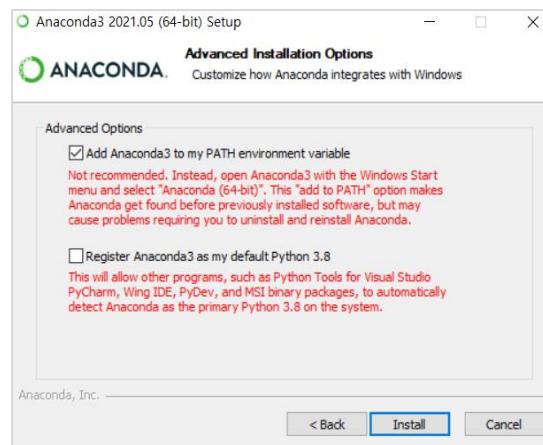
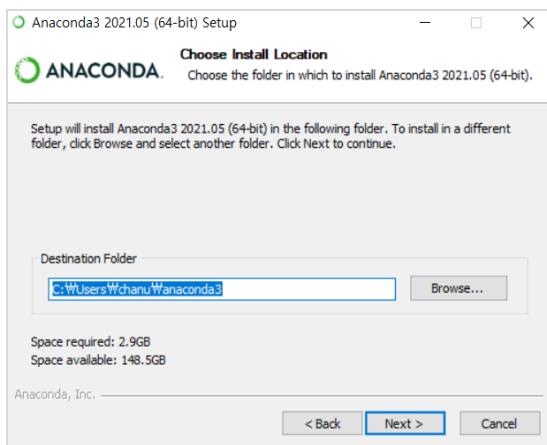
After downloading and executing the anaconda installation file, follow the procedure below to install the program.



# 1. Setting Up

## 1.1. Installing Anaconda

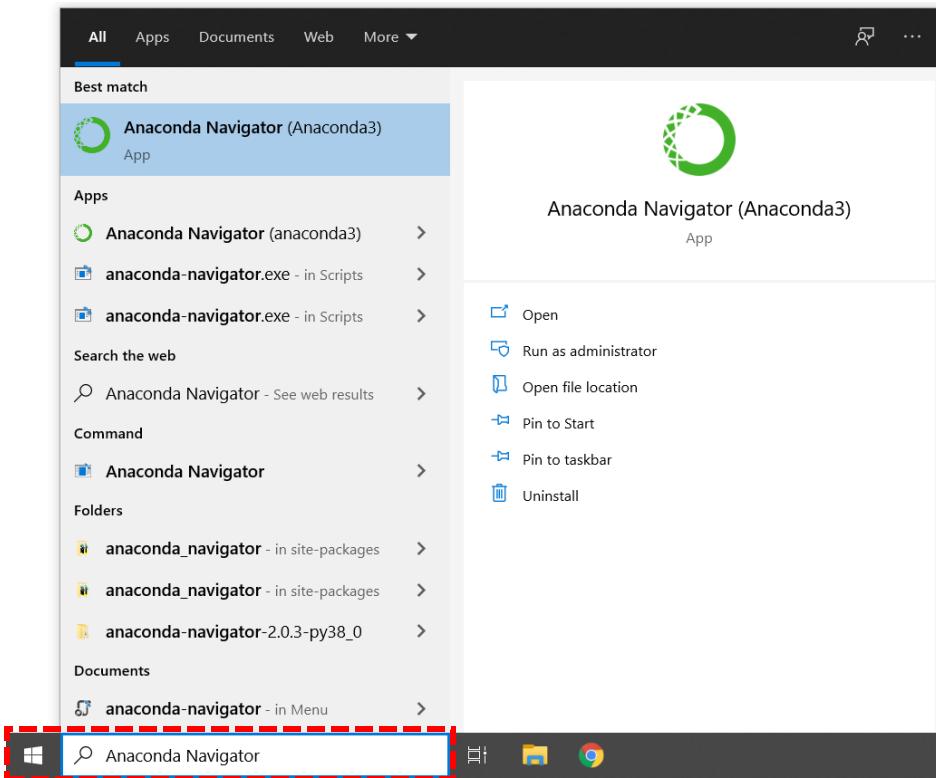
- When installing Anaconda, the computer's username must be in English. Otherwise, the program will not install.
- If a separate Python program has been previously installed, it will overlap with Python.exe of Anaconda. Go to Advanced Options and uncheck Add Anaconda3 to my PATH environment variable. In this case, you can execute the program by clicking on the Anaconda Prompt in the start menu.



# 1. Setting Up

## 1.2. Installing Jupyter Notebook

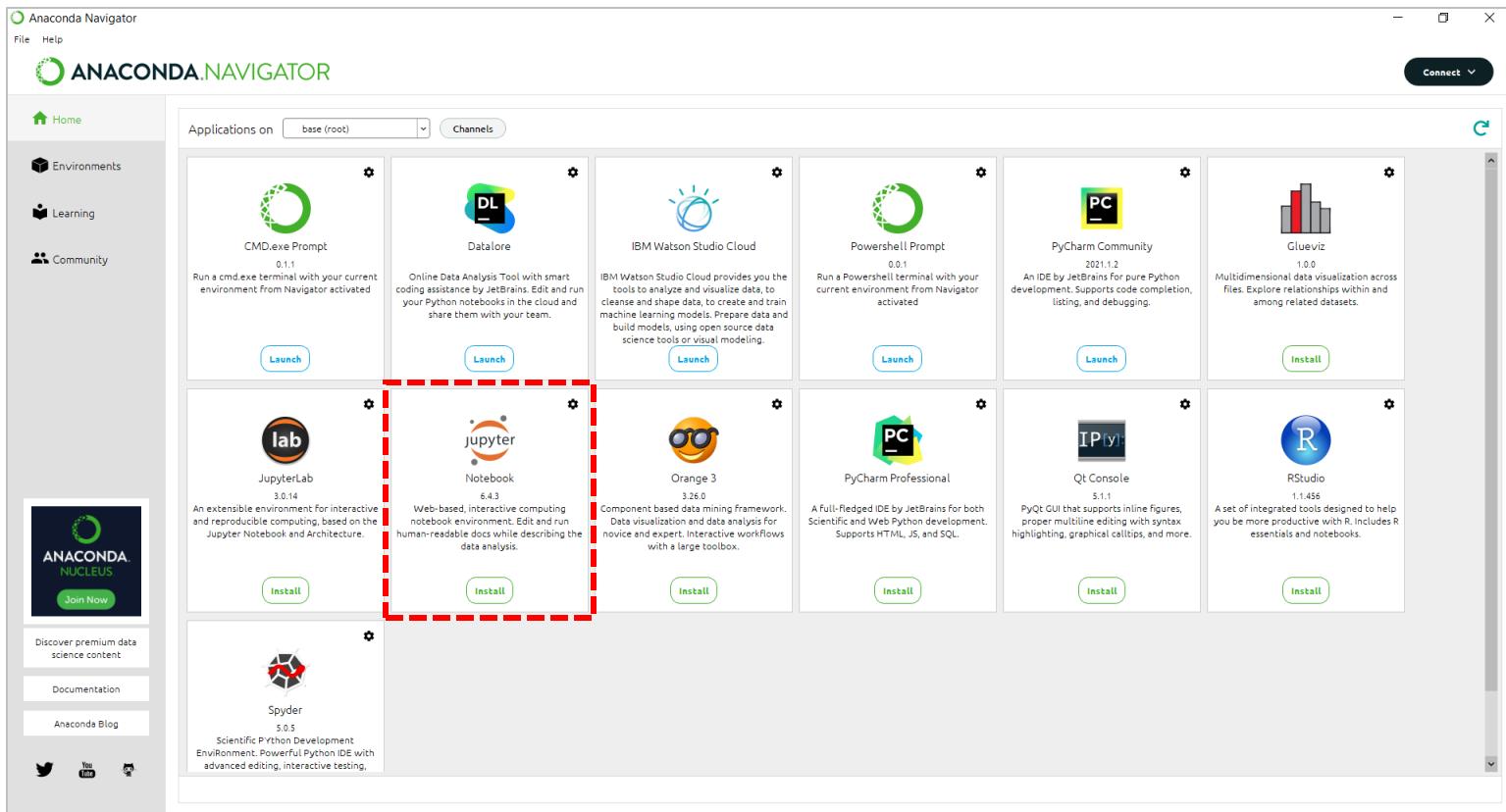
After installing Anaconda, let's install Jupyter Notebook. First, search and run Anaconda Navigator in the Windows Start menu.



# 1. Setting Up

## 1.2. Installing Jupyter Notebook

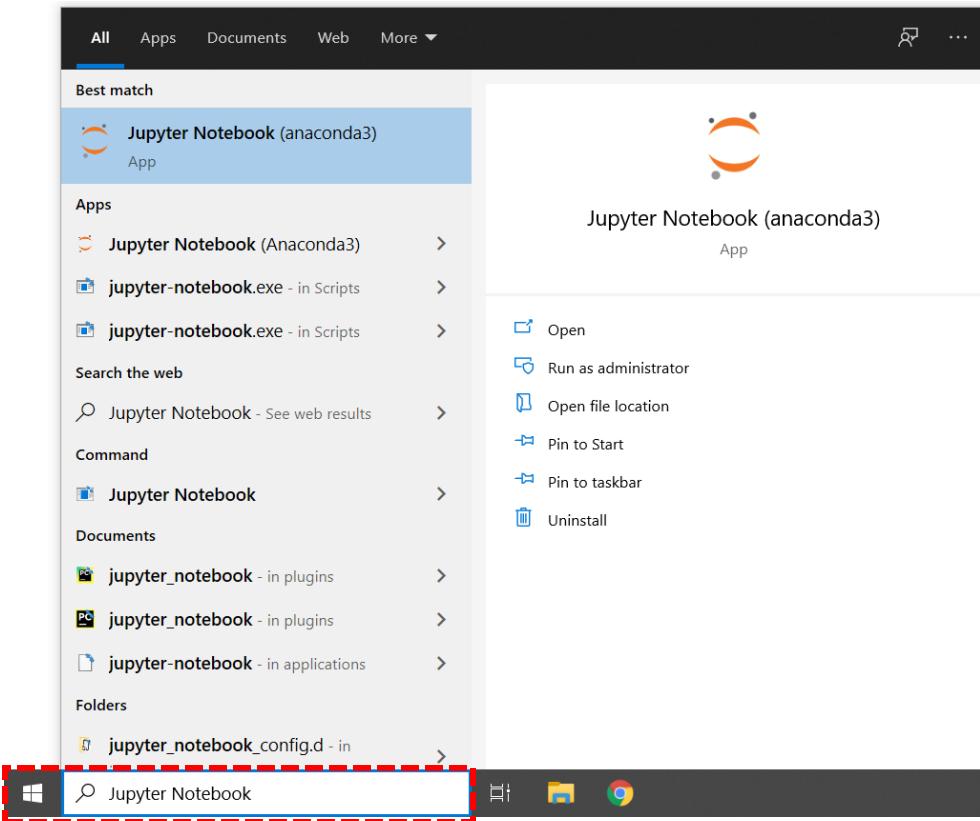
In the Anaconda Navigator as shown below, click the Notebook's Install button complete the installation.



## 3. How To Use Jupyter Notebook

### 3.1. Running the Jupyter Notebook

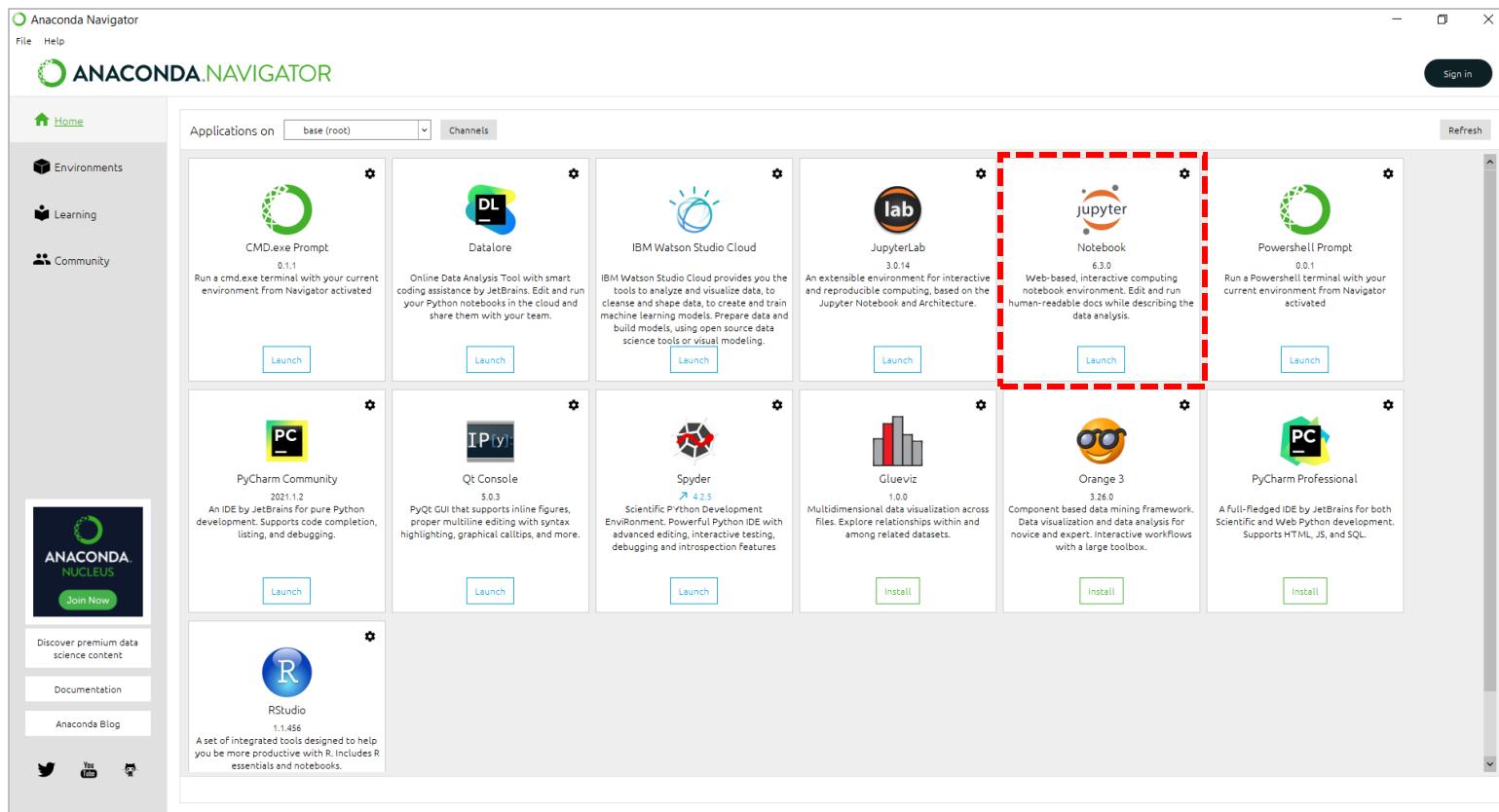
There are three main ways you can run Jupyter Notebook. First, go to Windows Start menu and search Jupyter Notebook.



## 3. How To Use Jupyter Notebook

### 3.1. Running the Jupyter Notebook

I Second, select the Jupiter Notebook from the Anaconda Navigator and click the Launch button.



## 3. How To Use Jupyter Notebook

### 3.1. Running the Jupyter Notebook

I Third, enter the command "jupyter notebook" in Anaconda Prompt.

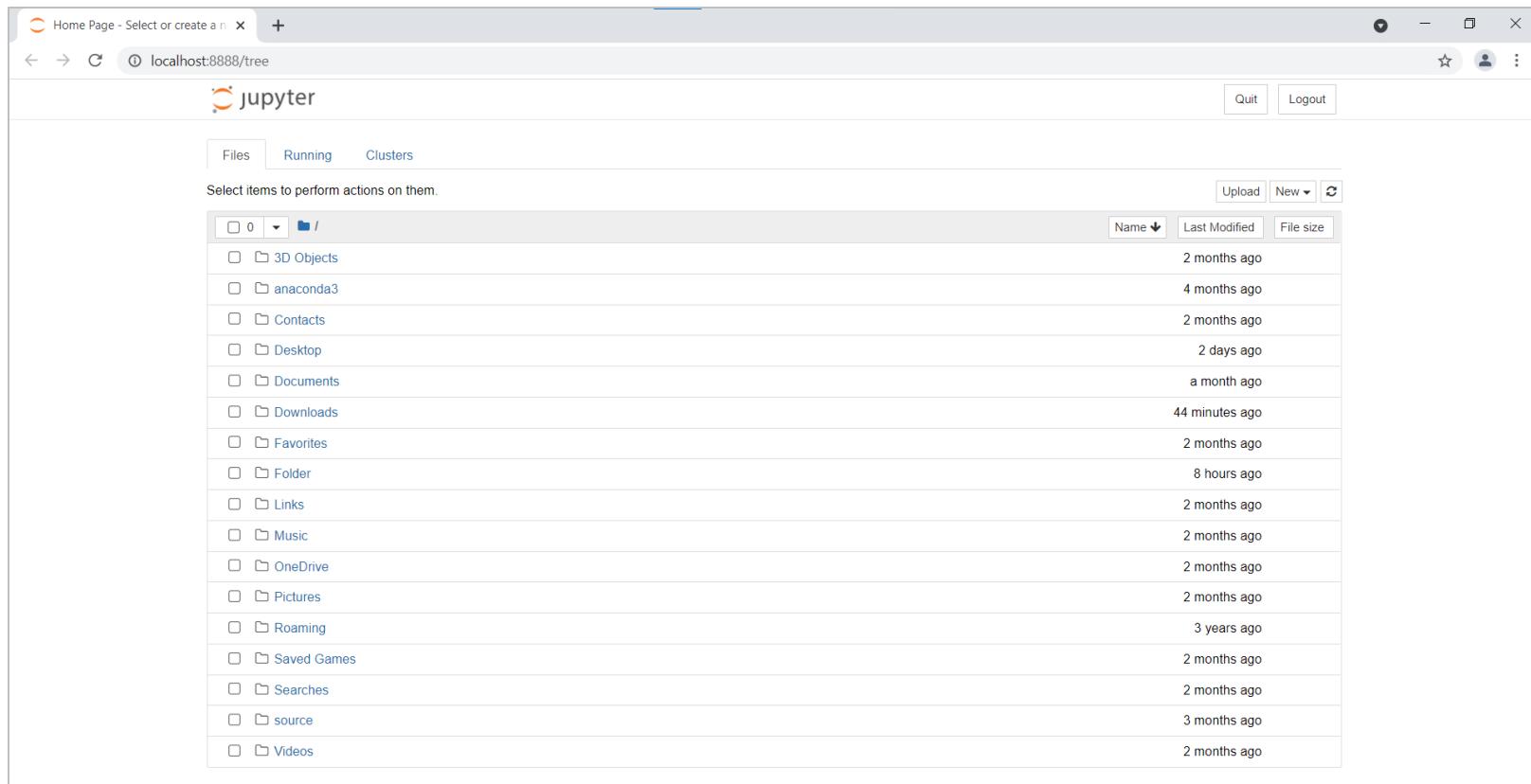
```
Administrator: Anaconda Prompt (Anaconda3) - jupyter notebook
(base) C:\Users\Administrator\jupyter notebook
[1 2021-09-30 16:39:39.738 LabApp] JupyterLab extension loaded from C:\ProgramData\Anaconda3\lib\site-packages\jupyterlab
[1 2021-09-30 16:39:39.738 LabApp] JupyterLab application directory is C:\ProgramData\Anaconda3\share\jupyter\lab
[1 16:39:39.742 NotebookApp] Serving notebooks from local directory: C:\Users\Administrator
[1 16:39:39.742 NotebookApp] Jupyter Notebook 6.3.0 is running at:
[1 16:39:39.743 NotebookApp] http://localhost:8888/?token=b023c3ed6ecce5b3f297033b1ec5d50ba7a223b523f1a877
[1 16:39:39.743 NotebookApp] or http://127.0.0.1:8888/?token=b023c3ed6ecce5b3f297033b1ec5d50ba7a223b523f1a877
[1 16:39:39.743 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 16:39:39.900 NotebookApp]

To access the notebook, open this file in a browser:
file:///C:/Users/Administrator/AppData/Roaming/jupyter/runtime/nbserver-3980-open.html
Or copy and paste one of these URLs:
http://localhost:8888/?token=b023c3ed6ecce5b3f297033b1ec5d50ba7a223b523f1a877
or http://127.0.0.1:8888/?token=b023c3ed6ecce5b3f297033b1ec5d50ba7a223b523f1a877
```

## 3. How To Use Jupyter Notebook

### 3.2. Jupyter Notebook UI

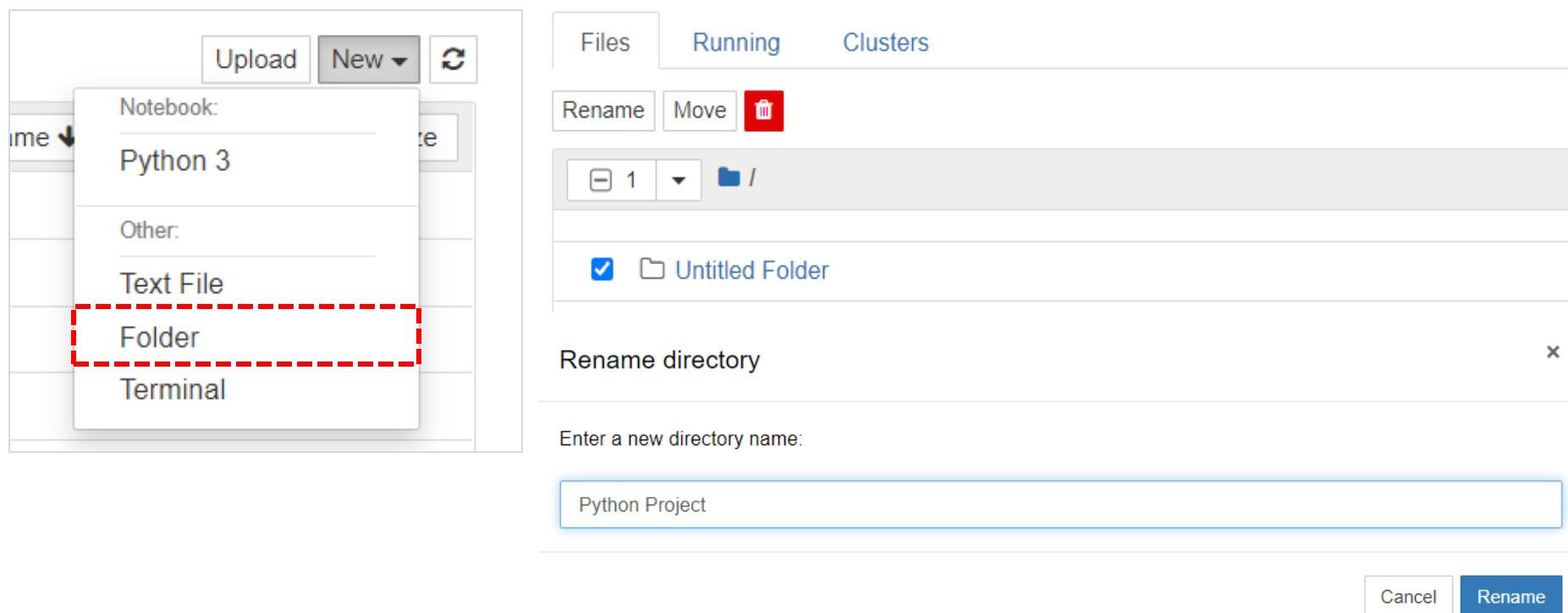
| You will find a screen below when you run Jupyter Notebook.



### 3. How To Use Jupyter Notebook

#### 3.2. Jupyter Notebook UI

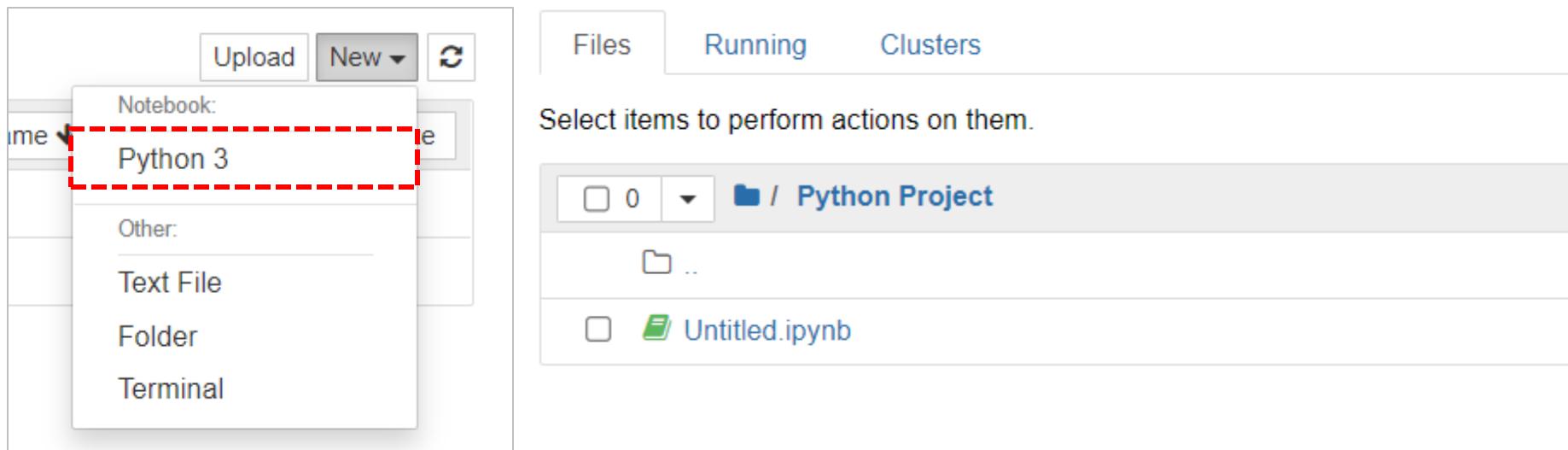
Press the New button at the top right for a drop-down menu. Click Folder to create a new folder. When a new folder is created, click the check box. Then, you can rename, relocate, or delete the folder. Try renaming the folder to “Python Project.”



### 3. How To Use Jupyter Notebook

#### 3.2. Jupyter Notebook UI

- Enter the newly created folder then click on the New button again. Click on the Python 3 to create a new ipynb file. ipynb stands for IPython Notebook, and IPython is an interactive work environment.
- In the example below, a new ipynb file called Untitled was created under the Python Project folder. Alternatively, you can click on the Upload button on the upper right corner to upload the ipynb file from the computer.



## 3. How To Use Jupyter Notebook

### 3.2. Jupyter Notebook UI

A green icon appear when the ipynb file is running. To end the file running, you can 1) click on the checkbox and then click on the Shutdown button under the Files tab, or 2) click on the Shutdown button at the bottom right corner of the Running tab. Running tab shows you the files that are currently running.

The screenshot shows the Jupyter Notebook interface with the 'Files' tab selected. At the top, there are tabs for 'Files', 'Running', and 'Clusters'. Below the tabs is a toolbar with buttons for 'Duplicate', 'Shutdown' (which is highlighted in orange), 'View', 'Edit', and a trash bin icon. The main area displays a file tree under the path '/ Python Project'. A file named 'Untitled.ipynb' is selected, indicated by a checked checkbox next to its name. To the right of the file list are filters for 'Name', 'Last Modified', and 'File size'. Below the file list, it shows 'seconds ago' and 'Running 4 minutes ago 711 B'.

The screenshot shows the Jupyter Notebook interface with the 'Running' tab selected. At the top, there are tabs for 'Files', 'Running', and 'Clusters'. Below the tabs, it says 'Currently running Jupyter processes'. Under the 'Terminals' section, it says 'There are no terminals running.' Under the 'Notebooks' section, it shows a list with 'Python Project/Untitled.ipynb'. To the right of this list, it says 'Python 3' and has a 'Shutdown' button, followed by 'seconds ago'.

### 3. How To Use Jupyter Notebook

#### 3.2. Jupyter Notebook UI

- When you click on the Shutdown button, the icon next to the file will turn gray. The Running tab will show that "There are no notebooks running.
- When you finish running the file, you can Rename, Move, or Download the ipynb file from the Files tab.

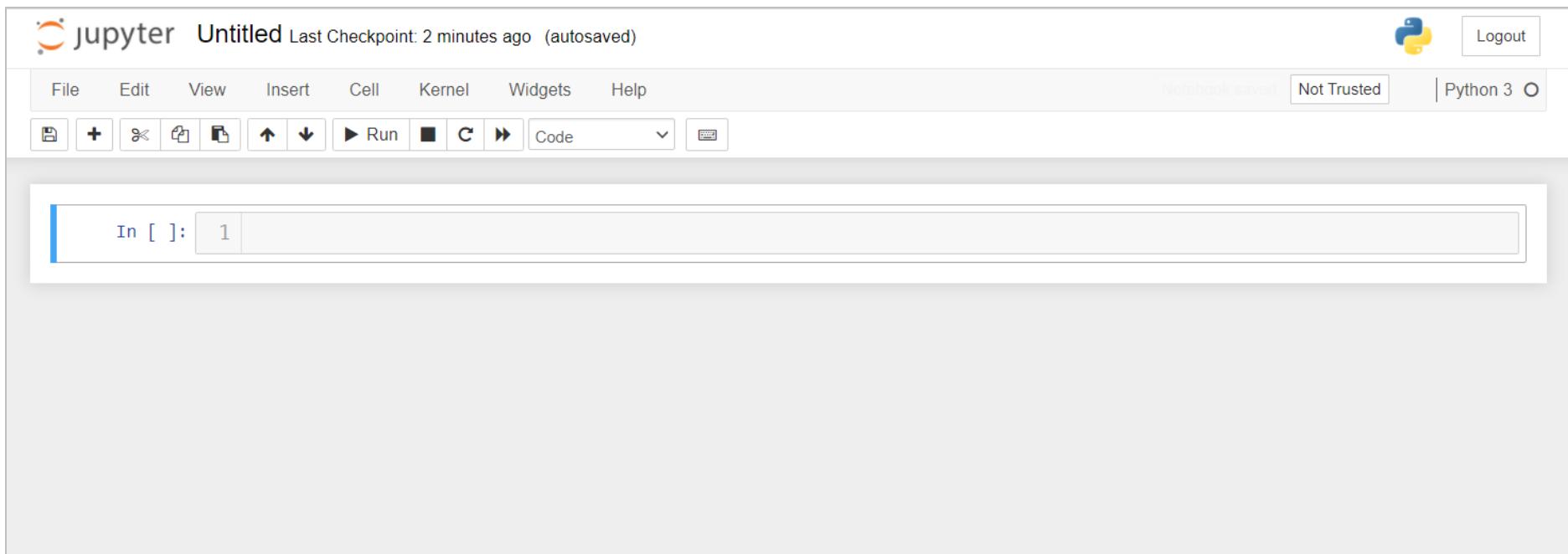
The screenshot shows the Jupyter Notebook interface with the 'Files' tab selected. At the top, there are tabs for 'Files', 'Running', and 'Clusters'. Below the tabs is a toolbar with buttons for 'Duplicate', 'Rename', 'Move', 'Download', 'View', 'Edit', and a trash icon. On the right side of the toolbar are 'Upload', 'New', and a refresh icon. The main area displays a file list under the heading 'Python Project'. There is one item listed: 'Untitled.ipynb' (checkbox checked, folder icon), with a last modified time of 'seconds ago' and a size of '728 B'. There is also a '..' entry. On the far right of the file list are buttons for 'Name', 'Last Modified', and 'File size'.

The screenshot shows the Jupyter Notebook interface with the 'Running' tab selected. At the top, there are tabs for 'Files', 'Running', and 'Clusters'. Below the tabs is a section titled 'Currently running Jupyter processes' which is empty. Underneath it, there are two expandable sections: 'Terminals' (which shows 'There are no terminals running.') and 'Notebooks' (which shows 'There are no notebooks running.').

## 3. How To Use Jupyter Notebook

### 3.2. Jupyter Notebook UI

- When you click on the ipynb file called Untitled, you will be a screen where you can code. Write a code in the input window next to "In [ ]:" and click the Run button to execute. This input window is called a cell, and you can use the Edit tab to copy, paste, merge, or move cells.
- To activate the cell line number, click "Toggle Line Numbers" on the View tab.



## 4. print() Function

### 4.1. The Basics

print( ) code is a command that outputs values inside the ( )

**print("Hello World!")**

Both single quote and double quote marks work, but the start and end symbol has to match.

The value between the open quotation and close quotation marks are called a **string**.



TIP

- Unlike other languages, Python automatically breaks a line after outputting a string through print().

## 4. print() Function

### 4.2. Printing String and Value

- | To print a string using the print( ) function, you must enclose both sides of the string with quotation marks.

```
1 print('Hello World!')  
2 print("Hello World!")
```

Hello World!  
Hello World!

- | To print a numerical value, enter the numbers without quotation marks. If you print a number with the quotation mark, the data type will be an integer.

```
1 print(10)  
2 print(7.5)
```

10  
7.5

- | You can also perform an arithmetic operations in the print( ) function. The arithmetic operations of number data types will be covered in Unit 3.

```
1 print(10 + 7.5)  
2 print(10 - 7.5)
```

17.5  
2.5

## 4. print() Function

### 4.3. Adding a New Line

- | Python automatically breaks the line after printing a string through a print( ) function. If so, could lines break between sentences within the same print( ) function?
- | Using an \n escape sequence within a string adds a new line. The escape sequence refers to a special character used for controlling a string output.

```
1 print('Hello\nWorld!')
```

Hello  
World!

## 4. print() Function

### 4.3. Adding a New Line

- In the print( ) function, the beginning and end of a string may be grouped with three double quotes ("") or three single quotes (''). This will display the grouped string, including the line break, on the screen.

```
1 print('''Hello  
2 World!''')
```

Hello  
World!

- However, if you put a backslash(\) at the end of a grouped string with three quotes, the line will not be broken, and the next line will be connected to the same line.

```
1 print('''Hello\  
2 World!'''')
```

HelloWorld!

## 4. print() Function

### 4.4. The Comma(,) Additions(+), Multiplication(\*) Operators

- A comma, addition, or multiplication operators can be used to print more diverse and efficient strings or values.
- When a comma (,) operator is used between the first and second value, it adds a space in between the values then a line break. You may print two or more integers or values by combining it with a comma operator.

```
1 print('Hello', 'World!')  
2 print(10, 20)
```

Hello World!  
10 20

- You may also combine a string and a number with a comma operator.

```
1 print('Hello', 10)
```

Hello 10

## 4. print() Function

### 4.4. The Comma(,) Additions(+), Multiplication(\*) Operators

When strings are combined with an addition (+) operator, the second integer is printed after the first integer without a space then a line break occurs. When an addition (+) operator is used with values, it will print the result of the addition then a line break.

```
1 print('Hello' + 'World!')  
2 print(10 + 20)
```

HelloWorld!  
30

 When using an addition (+) operator, you cannot combine strings and numbers. When a string and a number are combined with an addition operator (+), TypeError occurs.

```
1 print('Hello' + 10)
```

```
-----  
TypeError                                     Traceback (most recent call last)  
<ipython-input-3-62277d5ee2c1> in <module>  
----> 1 print('Hello' + 10)
```

**TypeError:** can only concatenate str (not "int") to str

## 4. print() Function

### 4.4. The Comma(,) Additions(+), Multiplication(\*) Operators

When a string and a positive integer are combined using a multiplication (\*) operator, the string is repeated as many times as the number and without any space, then a line break occurs. When a multiplication (\*) operator is used for a number, a line break occurs after printing the result of the multiplication operation.

```
1 print('Hello' * 3)
2 print(10 * 20)
```

```
HelloHelloHello
200
```

 When a string and a string are combined with a multiplication operator (\*), TypeError occurs.

```
1 print('Hello' * 'World!')
```

```
TypeError                                     Traceback (most recent call last)
<ipython-input-6-b6a7f39bdc2f> in <module>
----> 1 print('Hello' * 'World!')
```

```
TypeError: can't multiply sequence by non-int of type 'str'
```

## 4. print() Function

### 4.5. End and Split Parameters

- When a program encounters a print() function, a line break occurs. However, when a string is specified at the end of the print( ) statement, the string is printed instead of the line break.
- The first code prints 'Hello' then immediately prints 'World!' without breaking the line. The second code prints a comma instead of a line break.

```
1 print('Hello', end = '')  
2 print('World!')
```

HelloWorld!

```
1 print('Hello', end = ',')  
2 print('World!')
```

Hello,World!

## 4. print() Function

### 4.5. End and Split Parameters

- The previously learned comma(,) operator within the print() function prints a space between two strings or values. When a string is specified in the last sep of the print( ) function, the specified string is printed instead of the space.
- The first code prints 'Hello' then immediately prints 'World!' without any spaces. The second code prints a comma instead of a space.

```
1 print('Hello', 'World!', sep = '')
```

HelloWorld!

```
1 print('Hello', 'World!', sep = ',',)
```

Hello,World!

## 4. print() Function

### 4.6. Creating a Resume

| Below is an example code of today's mission. Let's practice coding a resume using operators we learned in this lesson; comma (+), multiplication (\*) operator, and end and split parameter.

```
1 print('Hello! I will introduce myself.')
2 print('Name: David Doe')
3 print('Age: 23')
4 print('Job: Data Scientist')
5 print('Address: Seoul, South Korea')
6 print('Place of Birth: Southern California, USA')
```

| Also try using asterisk (\*) or minus (-) symbols to decorate the resume.

```
1 print('*****')
2 print('-----')
```

## 5. Pythonic way

### 5.1. Why is Pythonic Coding Important?

- | It is important to form a good coding style and habit from the beginning.
- | The main purpose of Pythonic Coding is to write clear and concise codes using characteristics that are different from other programming languages.
- | Utilizing Python coding conventions used by developer communities also help in collaborating with other developers.
- | There are several conventions for the Pythonic way, but we will focus on the essentials for the beginners.

#### PEP 8 -- Style Guide for Python Code

PEP:	8
Title:	Style Guide for Python Code
Author:	Guido van Rossum <guido at python.org>, Barry Warsaw <barry at python.org>, Nick Coghlan <ncoghlan at gmail.com>
Status:	Active
Type:	Process
Created:	05-Jul-2001
Post-History:	05-Jul-2001, 01-Aug-2013

- ▶ There are two main Python style guides.
- ▶ PEP8 of Python's official community and Google's Python Style Guide.
- ▶ This unit utilizes the PEP8 guide.

<https://www.python.org/dev/peps/pep-0008/>

## 5. Pythonic way

### 5.2. Code lay out

| Python allows writing multiple sentences in one line using semicolons as shown below. This style of coding, however, does not have a good readability.

```
1 a = 10; a += 10; print(a) # Bad case
```

20

| When coding with Python, it is recommended to use one sentence in each line as shown below.

```
1 a = 10      # Good case
2 a += 10
3 print(a)
```

20

## 5. Pythonic way

### 5.2. Code lay out

| A long code like below is very difficult to read.

```
1 if width == 0 and height == 0 and color == 'red' and emphasis == 'strong' or highlight > 100:  
2     print('Sorry, you lose.')
```

| When coding with Python, if a sentence gets too long, try making a line break. Use a backslash(\) to add a line.

```
1 if width == 0 and height == 0 and color == 'red' and \  
2     emphasis == 'strong' or highlight > 100:  
3     print('Sorry, you lose.')
```

| Finally, operators and operands have spacing in between them. When listing elements, put a space after a comma.

```
1 a=[1,2,3]      # Bad
```

```
1 a = [1, 2, 3]  # Good
```

## 5. Pythonic way

### 5.3. Naming convention

- | The names used in programming, like variables, functions, classes and etc., are called identifiers. The types of naming styles of identifiers are shown in the table below.
- | In Python, the names of variables and functions generally follow the Snake style, and the class names follow the Pascal style.

Style	Description	Example
Snake	The first letter starts with a lowercase letter and uses _ (underscore) instead of spaces between words. It's called Snake Style because the underscores looks like snakes.	hello_world
Camel	The first letter starts with a lowercase letter, and the start of the next word is capitalized instead of spacing. It's called the Camel style because the capital letters in the middle looks like a camel's back.	helloWorld
Pascal	Almost the same as the Camel style, but the first alphabet is also capitalized.	HelloWorld

## 5. Pythonic way

### 5.3. Naming convention

| Python's conventions for naming identifiers are as follows:

1. Generally consists of English characters, numbers, and underscores (\_).
  2. There should be no spaces in the middle of the text.
  3. The first word must start with an English letter or an under score (\_).
  4. Upper and lower case letters are distinguished. For example, Count and count are different identifiers.
  5. There is no limit to the length of the identifier.
- 6. Keywords cannot be used as identifiers.**
7. Identifiers starting with \_\_(two underscores) refer to special attributes or use only as the name of specific methods. Therefore, it is not used as the name of a general function or variable.

## 5. Pythonic way

### 5.3. Naming convention

I Examples of the **correct** identifiers:

Correct Identifiers	Reasons
number4	Numbers can be used after starting with English characters.
my_list	Underscores can be used anywhere in the identifier, like the rest of the characters.
__code__	The name of a specific method can start with __ (two underscores).
for_loop	A keyword can be used by connecting with other characters.

## 5. Pythonic way

### 5.3. Naming convention

I Examples of the **incorrect** identifiers:

Incorrect Identifiers	Reasons
1st_variable	Cannot be used because it starts with a number.
my list	Cannot be used because there is a space.
global	Python keywords cannot be used as identifiers.
ver2.9	A special character other than the underscore cannot be used.
num&co	A special character other than the underscore cannot be used.

## 6. Error

### 6.1. Syntax Error

- | Programming languages are used under strict rules and conventions. Writing a code that strays from the grammatical rules results in a syntax error.
- | To prevent syntax errors, you must learn and practice Python grammar and code according to the set rules.
- | To print the string 'Hello', quotation marks must be used on both sides of the string. However, in the code below, a syntax error occurred because the string was not quoted.

```
1 print('Hello')

File "<ipython-input-2-db8c9988558c>", line 1
print('Hello')
^
SyntaxError: EOL while scanning string literal
```

## 6. Error

### 6.2. Runtime Error

- Unlike the syntax errors, grammatically correct codes can still result in error during the execution. This type of error is called runtime error.
- To prevent runtime errors, the developer should consider the possibility of user entering an incorrect data and suggest a user-friendly request to encourage entering a correct data. Also, various exception cases usually occur, so testing the code in various conditions is highly recommended.
- In the code below, the user entered a string two instead of an integer 2, and a runtime error occurred.

```
1 num = int(input('Input an integer: '))
```

```
Input an integer: two
```

```
-----  
ValueError                                Traceback (most recent call last)  
<ipython-input-3-2c1961434081> in <module>  
----> 1 num = int(input('Input an integer: '))  
  
ValueError: invalid literal for int() with base 10: 'two'
```



## One more step

### 1. Runtime Error and Resolution

- A example of a common runtime error is ZeroDivisionError, an error that divides numbers by zero. In order to solve such runtime errors, exceptions can be added using try ~ except statements.
- When the user correctly enters two integers as follows, the two division operations are executed without errors as intended by the program.

```
1 try:  
2     a, b = input('Enter two integers.').split()  
3     result = int(a) / int(b)  
4     print('{}/{} = {}'.format(a, b, result))  
5 except :  
6     print('Check if the integers are correctly entered.')
```

```
Enter two integers. 10 2  
10/2 = 5.0
```

 One more step

## 1. Runtime Error and Resolution

- If the user incorrectly enters a string instead of an integer, or causes a case where a number is divided by zero like the second example below, the except statement will handle the exception case.

```
1 try:  
2     a, b = input('Enter two integers.').split()  
3     result = int(a) / int(b)  
4     print('{}/{} = {}'.format(a, b, result))  
5 except :  
6     print('Check if the integers are correctly entered.')
```

Enter two integers. 10 two  
Check if the integers are correctly entered.

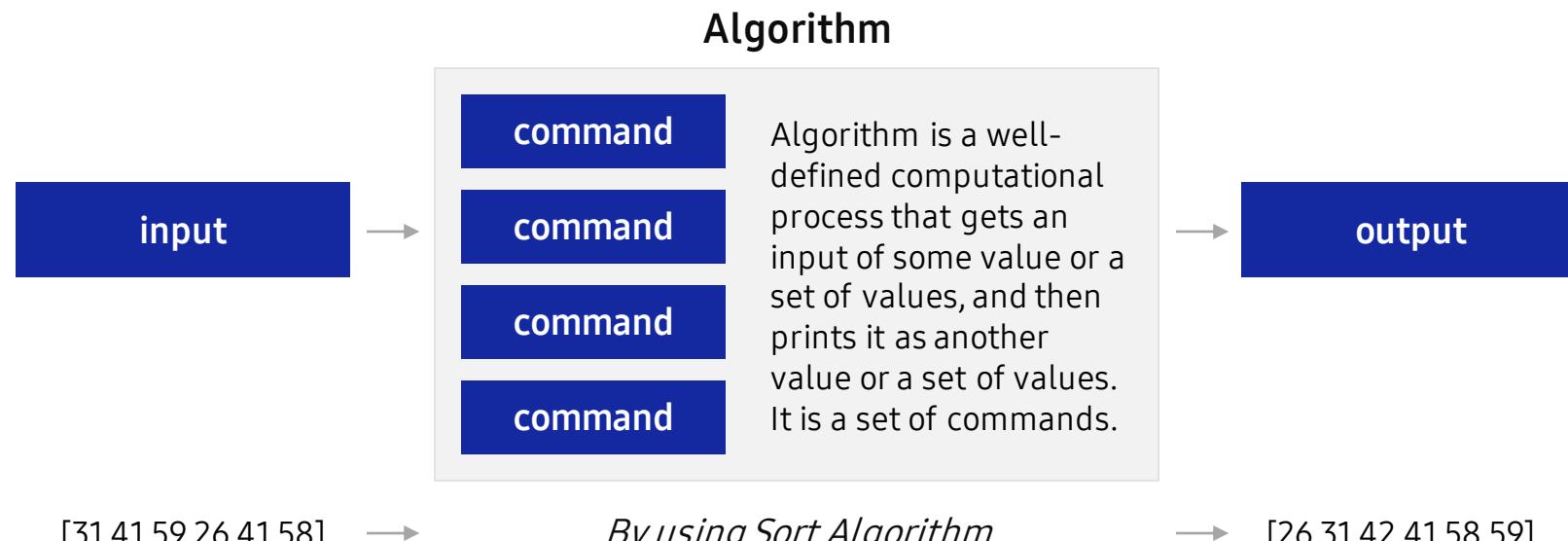
```
1 try:  
2     a, b = input('Enter two integers.').split()  
3     result = int(a) / int(b)  
4     print('{}/{} = {}'.format(a, b, result))  
5 except :  
6     print('Check if the integers are correctly entered.')
```

Enter two integers. 10 0  
Check if the integers are correctly entered.

## 1. Algorithm

### 1.1. The definition of an algorithm

- | An algorithm is a set of instructions for doing something.
- | An algorithm should have a reasonable sequence of computations and tasks to solve a problem, not just a set of commands.
- | We can plan for creating an algorithm, which is a procedure consisted of commands for problem-solving. The planning process can be made up with a pseudocode and flowchart.



## 2. Algorithm representation and planning

### 2.1. Pseudocode

- | Sequence is that a computer sequentially executes the commands from codes.
- | Pseudocode is a representation of an algorithm using daily language. Pseudocode can be a good start for developing algorithm when you do not have enough background knowledge about programming.
- | Pseudocode plays a role as a bridge for completing a flowchart and an actual code. It is often mostly used by programmers as a basic guideline when they implement the actual code.
- | It's the most essential that you should clearly explain and plan what each line of codes should perform.

This program will allow the user to check the number whether it's even or odd.

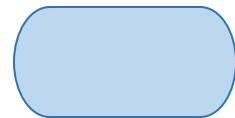
```
If "4"  
    print result  
    "This number is even"  
  
If "3"  
    print result  
    "This number is odd"
```

- ▶ Write a sentence that describes the purpose of the algorithm in the beginning.
- ▶ Write concisely.
- ▶ Create and use rules of simple and distinguishable names to be used.
- ▶ Indentation and whitespace are key points
- ▶ Express sequence structures, selection structures, and repeating structures using if, for, while.

## 2. Algorithm representation and planning

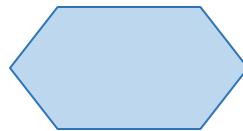
### 2.2. Flowchart

- Flowchart expresses commands with several standardized shapes and the flow of work using arrows.
- For shapes and symbols, those set by the International Organization for Standardization (ISO) are used.
- It is important that you draw them from top to bottom and from left to right without overlapping each other.



**Start, End**

Represent the start and end of a flowchart



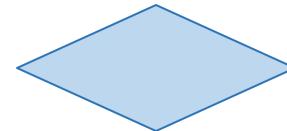
**Preparation**

Initialize data



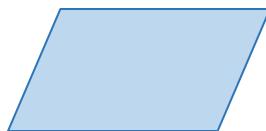
**Action or Process**

Arithmetic operations/processing of a single step



**Decision**

Used to branch to different parts of a flowchart based on judging a condition



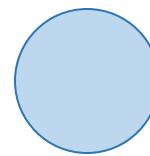
**Input, Output**

Used for input and output data



**Document**

Represent printed document or report



**On-page Connector**

Connect to this symbol of another flowchart (Junction)



**Flow Arrow**

Represent the flow of processing

# 💡 One more step

## 1. Introduction of Python learning references

- | There are good learning references to study Python by yourself. If you want to learn more about Python, visit the websites below.
- | Python Software Foundation

The screenshot shows the Python.org homepage. At the top, there are navigation links for Python, PSF, Docs, PyPI, Jobs, and Community. Below the header, there is a search bar and a "Socialize" button. The main content area features a code snippet for generating a Fibonacci series:

```
# Python 3: Fibonacci series up to n
>>> def fib(n):
>>>     a, b = 0, 1
>>>     while a < n:
>>>         print(a, end=' ')
>>>         a, b = b, a+b
>>>     print()
>>> fib(1000)
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987
```

Below the code, there is a section titled "Functions Defined" with a brief description of what it means to define a function in Python. At the bottom of the main content area, there is a footer with links for "Get Started", "Download", "Docs", and "Jobs".

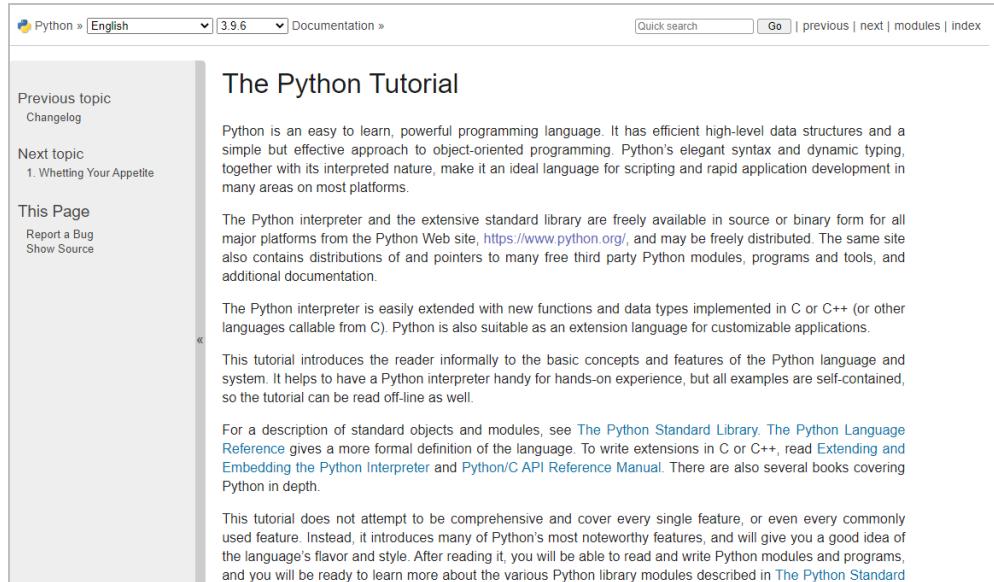
- ▶ A non-profit foundation responsible for the development and distribution of Python.
- ▶ Be able to download the latest version of Python for free.
- ▶ Provide Tutorials and documents for learning Python.
- ▶ Introduce the latest news and events.
- ▶ Provide some successful examples of great Python projects.

<https://python.org/>

# 💡 One more step

## 1. Introduction of Python learning references

### Python Software Foundation



The screenshot shows the Python Tutorial page. The top navigation bar includes links for English, version 3.9.6, Documentation, Quick search, Go, previous, next, modules, and index. On the left, there's a sidebar with links for Previous topic (Changelog), Next topic (1. Whetting Your Appetite), and This Page (Report a Bug, Show Source). The main content area is titled "The Python Tutorial". It starts with a paragraph about Python's ease of learning and its use in various applications. It then discusses the availability of the Python interpreter and standard library, mentioning the Python Web site. The tutorial is described as introductory, suitable for both online and offline reading. It also points to the Python Standard Library and Language Reference. A note at the bottom states that the tutorial does not cover every feature but introduces many noteworthy ones.

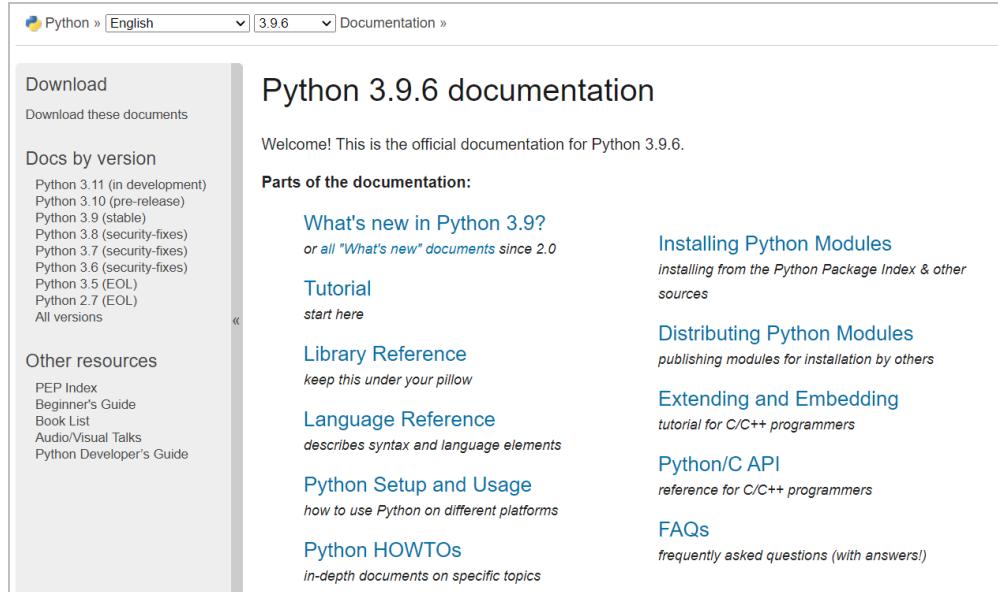
<https://docs.python.org/3/tutorial/index.html>

- ▶ Be able to learn the basics of Python in the Python Tutorial page.

# 💡 One more step

## 1. Introduction of Python learning references

### Python Software Foundation



The screenshot shows the Python 3.9.6 documentation homepage. The left sidebar contains links for 'Download' (including versions 3.11, 3.10, 3.9, 3.8, 3.7, 3.6, 3.5, 3.4, and all versions), 'Docs by version' (links to Python 3.11, 3.10, 3.9, 3.8, 3.7, 3.6, 3.5, 3.4, and EOL), and 'Other resources' (links to PEP Index, Beginner's Guide, Book List, Audio/Visual Talks, and Python Developer's Guide). The main content area features the title 'Python 3.9.6 documentation' and a welcome message: 'Welcome! This is the official documentation for Python 3.9.6.' Below this, there are several sections: 'Parts of the documentation:' (links to 'What's new in Python 3.9?' and 'Tutorial'), 'Installing Python Modules' (with a link to 'start here'), 'Distributing Python Modules' (with a link to 'publishing modules for installation by others'), 'Extending and Embedding' (with a link to 'tutorial for C/C++ programmers'), 'Python/C API' (with a link to 'reference for C/C++ programmers'), and 'FAQs' (with a link to 'frequently asked questions (with answers!)'). Each section includes a small descriptive subtitle.

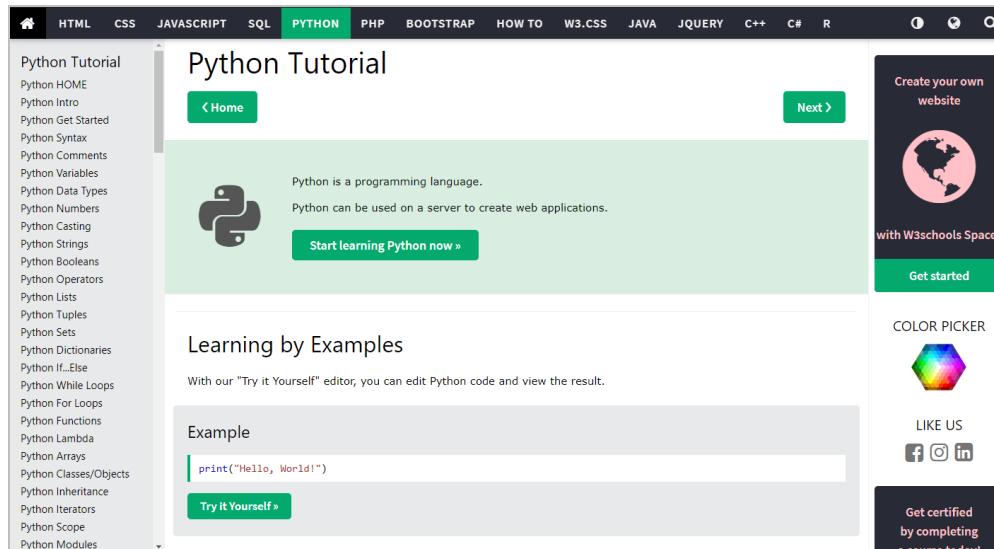
<https://docs.python.org/3/>

- ▶ The Python documentation page provides detailed documentation about Python.

# 💡 One more step

## 1. Introduction of Python learning references

### W3Schools



The screenshot shows the W3Schools Python Tutorial homepage. The top navigation bar includes links for HTML, CSS, JAVASCRIPT, SQL, PYTHON (which is highlighted in green), PHP, BOOTSTRAP, HOW TO, W3.CSS, JAVA, JQUERY, C++, C#, and R. The main content area features a large Python logo and text explaining that Python is a programming language used for web applications. A "Start learning Python now" button is visible. To the right, there's a sidebar with options to "Create your own website with W3schools Spaces", a "COLOR PICKER", and social media links for "LIKE US" on Facebook, Instagram, and LinkedIn. A "Get certified by completing" section is also present. On the left, a sidebar lists various Python topics: Python HOME, Python Intro, Python Get Started, Python Syntax, Python Comments, Python Variables, Python Data Types, Python Numbers, Python Casting, Python Strings, Python Booleans, Python Operators, Python Lists, Python Tuples, Python Sets, Python Dictionaries, Python If...Else, Python While Loops, Python For Loops, Python Functions, Python Lambda, Python Arrays, Python Classes/Objects, Python Inheritance, Python Iterators, Python Scope, and Python Modules.

<https://www.w3schools.com>

- ▶ How can we find out the materials that are helpful for applying Python more extensively?
- ▶ At W3Schools, you can learn various programming languages with Python.
- ▶ In particular, you can directly run Python code examples in a web environment.

# 💡 One more step

## 1. Introduction of Python learning references

### Stack Overflow

The screenshot shows a Stack Overflow question page. The title is "Reversing list element in python". The question was asked 1 month ago and has been viewed 55 times. The question text asks how to reverse list elements one by one in Python, mentioning nested loops and displaying the result as "Result". Below the question, there are input and result fields showing the numbers 102, 346, 589 and 201, 643, 7985 respectively. The tags listed are python, list, loops, numbers, and reverse. On the right side of the page, there is a sidebar with links to "The Overflow Blog", "Featured on Meta", and "Hot Meta Posts". There are also advertisements for "AnkiDroid Flashcards" and "Contribute on Q&A". At the bottom of the page, there is a link to "Create a free Team".

<https://stackoverflow.com/>

- ▶ While writing a program, you will encounter many errors and problems that are difficult to solve. How to solve these problems?
- ▶ Stack Overflow is the world's most popular information sharing website for developers.
- ▶ Search for “reverse list element in python” on this website. There will be many questions and discussions related to the list you will learn later.

Unit 3.

# **Basic of Numeric Data Types and Arithmetic Operation**

## Learning objectives

- ✓ Be able to do addition, subtraction, multiplication, division, and power operation using python. Also, understand and be able to use python's advanced function of finding quotient and remainder in division.
- ✓ Understand there are different data types for data processing and be able to execute numerical calculation using relationship between data types and operators.
- ✓ Learn how to deal with strings and be able to print data value from a code.
- ✓ Be able to write equations using operator precedence.

## Learning overview

- ✓ Perform math operations including arithmetic operation, power, etc. using numeric data types.
- ✓ Find out data types of objects and learn how operators work depending on data types.
- ✓ Make an output using string data types.

## Concepts you will need to know from previous units

- ✓ Printing an output on the screen using `print` and take input values from the users using `input`.
- ✓ How to run Python using Jupyter Notebook. Python is a Python Shell for interactive Python coding and should be able to write a Python code in the condition.

# Keywords

Data Types

Numeric Data  
Types

Arithmetic  
Operations

Strings

String Indexing

## 3. Mission

### 3.1. Print the circumference and area of the circle

- | In this mission, get an input of circle's radius.
- | Calculate circumference and area of the circle using operators we will learn in this unit.
- | Print calculated circumference and area of the circle on the screen.
- | Put various numbers like 4, 5, 6, 10, 20, 33 for the radius.

### 3. Mission

#### 3.2. How printing the circumference and area of the circle works

```
1 PI = 3.14
2 radius = float(input('Enter the radius :'))
3 circum = PI * radius * radius
4 area = 2.0 * PI * radius
5
6 print('Radius of the circle', radius)
7 print('Area of the circle', circum)
8 print('Circumference of the circle', area)
```

Enter the radius : 4.0

Radius of the circle 4.0

Area of the circle 50.24

Circumference of the circle 25.12

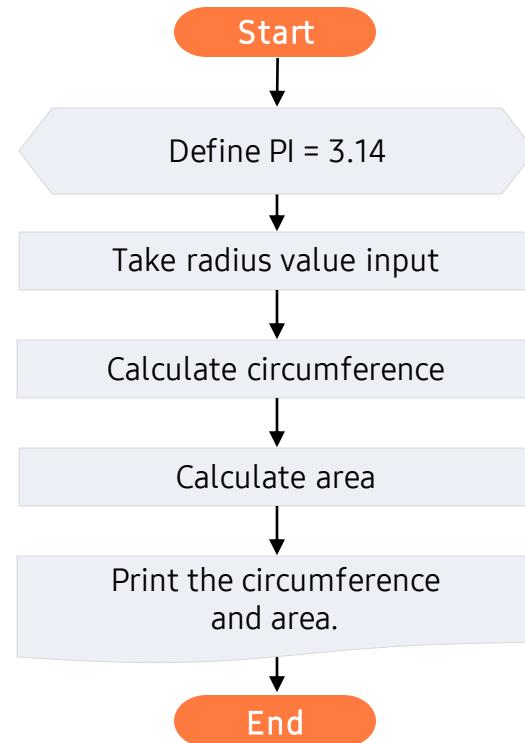
## 3. Mission

### 3.3. Programming plan

#### Pseudocode

- [1] Start
- [2] Define value of PI as 3.14
- [3] Take the radius value input
- [4] Calculate circumference using the radius input value
- [5] Calculate area using the radius input value
- [6] Print the circumference and area on the screen
- [7] End

#### Flow chart



### 3. Mission

#### 3.4. Printing the circumference and area of the circle: final code

```
1 PI = 3.14
2 radius = float(input('Enter the radius :'))
3 circum = PI * radius * radius
4 area = 2.0 * PI * radius
5
6 print('Radius of the circle', radius)
7 print('Area of the circle', circum)
8 print('Circumference of the circle', area)
```

# | Key concept

# 1. Numeric Data Types, Arithmetic and Division

## 1.1. Computation using numerical information

In programming, operations using numerical information and then using the data frequently occur.

For example, information such as a person's height and weight is expressed as numerical values such as 177cm and 58kg.

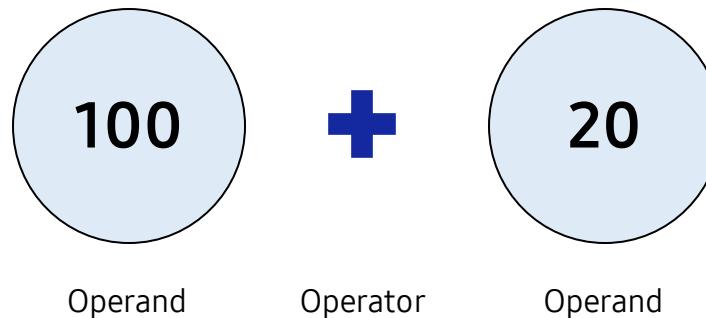
- Python expresses numerical data through numbers.
  - Various mathematical operators are provided for this numerical data.
- Ex** Addition operations of the numbers 100 and 20  
can be performed by  $100 + 20$ .



## 1. Numeric Data Types, Arithmetic and Division

### 1.1. Computation using numerical information

Let's learn the terms for arithmetic operation. Data values or variables subject to the operation are called operands. The subject of the operation is called the operator. Take an example of an operation of  $100 + 20$ .



# 1. Numeric Data Types, Arithmetic and Division

## 1.2. Operators

The table below explains the operator of Python and the operation of the operator.

Operator	Meaning	Action
+	Addition	Add the left operand and the right operand.
-	Subtraction	Subtract the right operand from the left operand.
*	Multiplication	Multiply the right operand from the left operand.
/	Division	Divide the left operand into the right operand. Python's division basically returns the real number value.
//	Floor Division(quotient)	Unlike /, the result of division is used to discard the decimal point or less and obtain only the integer part.
%	Remainder	It is read as a modulo operator and has nothing to do with the percentage that means ratio. Finds the remainder of the division.
**	Power	Multiply the left operand to the right operand. In the case of **0.5, a square root operation is performed.

# 1. Numeric Data Types, Arithmetic and Division

## 1.3. Example code using operator

- | Perform an operation using numerical data types. The operation may be performed using numerical data and operator +, -, \*, /, //, %.

```
1 | 100 + 20 # Find addition of numerical value of 100 and 20
```

```
120
```

```
1 | 100 * 20 # Find multiplication of numerical value of 100 and 20
```

```
2000
```

```
1 | 100 - 20 # Find subtraction of numerical value of 100 and 20
```

```
80
```

```
1 | 100 / 20 # Divide numerical value of 100 with 20
```

```
5.0
```

```
1 | 100 // 20 # Divide numerical value of 100 with 20 and find quotient
```

```
5
```

```
1 | 100 % 20 # Divide numerical value of 100 with 20 and find quotient
```

```
0
```

## 2. Object Data Types, Operator Priorities, and Data Type Conversion

### 2.1. Classification of data types

- | In Python, the distinction of data types is very important.
- | Characters and numbers have the same form, but their properties can be very different.
  - | Ex The number 10 can be calculated. However, the letter "10" is impossible to calculate.
  - | Depending on the data type, the available operators are different. In the following case, a grammar error occurs because the string "10" is divided by 2.

 **TypeError**

```
1 | 10 / 2
5.0

1 | '10' / 2  String 10 divided by number 2
```

---

```
TypeError                                     Traceback (most recent call last)
<ipython-input-40-67c5b32d2496> in <module>
----> 1 '10' / 2 # String 10 divided by number 2

TypeError: unsupported operand type(s) for /: 'str' and 'int'
```

## 2. Object Data Types, Operator Priorities, and Data Type Conversion

### 2.1. Classification of data types

- In Python, the operator applies differently depending on the data type.
- The result of the multiplication (\*) operation of the numbers 100 and 2 is 200.
- However, the result of the multiplication (\*) operation of the letters '100' and 2 is '100100'. In other words, the result of the multiplication (\*) operation of the letter 'Hello' and 2 is 'HelloHello', but the multiplication operation on the letter is different from the multiplication operation on the numerical value.

```
1 | 100 * 2      # Find multiplication of numerical value 100 and 2
```

```
200
```

```
1 | 'Hello' * 2  # Find multiplication of string 'Hello' and 2
```

```
'HelloHello'
```

```
1 | '100' * 2    # Find multiplication of string '100' and 2
```

```
'100100'
```

## 2. Object Data Types, Operator Priorities, and Data Type Conversion

### 2.2. Basic data types in Python

| The basic data types and examples of Python, which are widely used, are as follows.

Data Types	Example
Integers(int)	..., -3, -2, -1, 0, 1, 2, 3, ...
Real Number(float)	3.14, 4.28, 0.01, 123.432
String(str)	'Hello World', '123', "Hi"
Bool(bool)	True, False

## 2. Object Data Types, Operator Priorities, and Data Type Conversion

### 2.2. Basic data types in Python

Here are the important data structure of Python and types of variables. These will be used in coding.

Number	Boolean	String	List	Tuple	Dictionary
int ex : 10, 200	ex: True, False	ex : 'hello' '100'	ex: [10, 20, 30]	ex: (10, 20, 30)	ex: {'name': 'David', 'age': 23, 'height': 56.5}
float ex : 3.14					
complex ex : 4 + 3j					

## 2. Object Data Types, Operator Priorities, and Data Type Conversion

### 2.3. How to distinguish data types

- | The terms referring to numerical values, functions, and classes, which are components of the Python program, is object.
- | The role of the operator differs depending on whether the object's data type is integer or string.
- | Type function returns the type of an object. Let's check the data type using the code below.

```
1 type(100) # Let you know data type of 100  
int
```

```
1 type(100.0) # Let you know data type of 100
```

float

Numeric data type of 100 and 100.0 are different from each other.

```
1 type('100') # Let you know data type of 100
```

str

Write complex number in form of x + y j.

```
1 type(10 + 3j) # Complex number data type in the form of real + imaginary numbers like 10 + 3j
```

complex

## 2. Object Data Types, Operator Priorities, and Data Type Conversion

### 2.4. Precedence of operators

- | If the addition and multiplication operators appear in one sentence, the multiplication operation is performed first.
- | When parentheses appear in a sentence, the operators in parentheses are treated first.
- | The calculation results in the sentence vary according to the operator processing order.

```
1 10 + 20 * 30    # Multiplication is calculated first.
```

610

```
1 (10 + 20) * 30  # Operator in parentheses is calculated first.
```

900

## 2. Object Data Types, Operator Priorities, and Data Type Conversion

### 2.5. Operator precedence chart

| Let's look at Python operators and precedence. These are the contents to be learned in this lecture.



Operator	Explanation
()	parenthesis operator
**	power operator
~, +, -	monadic operator
* , / , % , //	multiplication, division, remainder operator
+ , -	addition, subtraction
>> , <<	bitwise movement operator
&	bitwise AND operator
^ ,	bitwise XOR operator, bit OR operator
<= , < , > , >=	comparison operator
== , !=	equal operator
= , %= , /= , //= , -= , += , *= , **=	assignment operator, complex assignment operator
is , is not	identity operator
in , not in	membership test operator
not , or , and	logical operator

The higher the table,  
the higher the priority.

## 2. Object Data Types, Operator Priorities, and Data Type Conversion

### 2.6. Data type conversion

- Python data can be calculated by converting its data type. The string '10' can be int-type converted to int('10'). Therefore, int('10')/2 is executed without error.
- In order to change the number 10 from the Python data to the string '10', str(10) is used. This converted string can be written with the string as an addition operation.

```
1 int('10') / 2 # Convert string '10' into integer and divide by 2.
```

```
5.0
```

```
1 'I like number ' + str(10) # Convert number 10 into string and use + operator
```

```
'I like number 10'
```

```
1 a = None  
2 type(a)
```

```
NoneType
```



## One More Step

- ▶ None can be used if the data type of the Python object is not determined.
- ▶ The data type of variable a with a None value is `NoneType`.
- ▶ None seems unnecessary because there is no data value. However, it is often necessary because it is the only way to indicate the absence of a value.

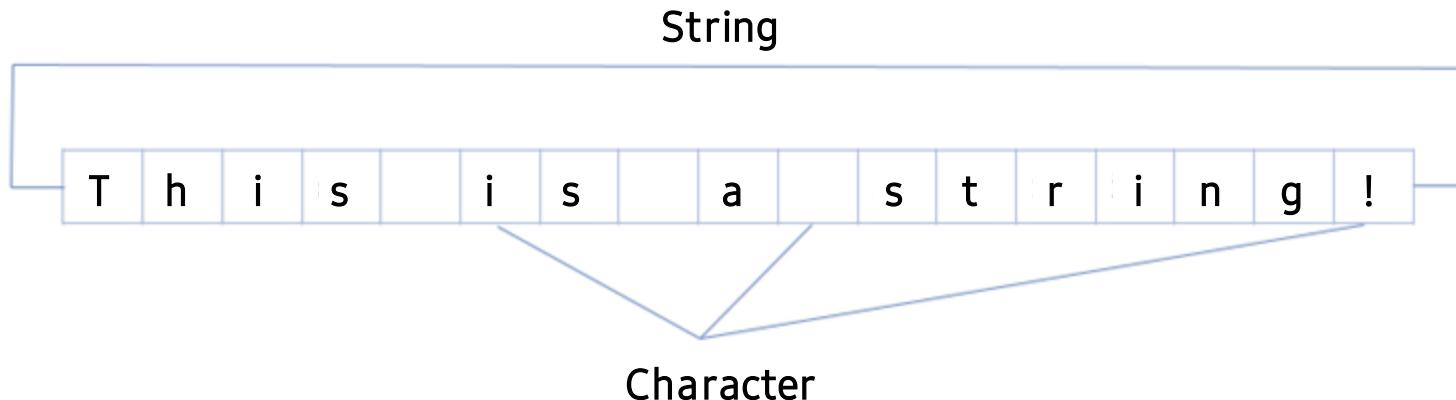
```
1 a = None  
2 type(a)
```

```
NoneType
```

## 3. String Application

### 3.1. Take a look at string

- | In Python, the string str is very suitable for displaying information in software.
  - | All srt files can be converted to texts. However, as it is a text, it is impossible to apply arithmetic calculation.
- Ex** The user's name, ID, password, etc. may be expressed as a string.
- | Python's string is a form in which each letter is connected.
  - | Addition and multiplication operations are possible for Python's strings.



## 3. String Application

### 3.2. String operator +, \*

Let's look at the operations + and \* available in the string. The + operation of the string connects the two strings. The \* operation of the string must be an integer. That is, the string is generated by repeating the integer n.

```
1 'I' + ' Love' + ' Python!' # String can be added to strings.  
1 'Python ' * 10 # Generates string of 'Python' repeated 10 times  
1 '*' * 50      #Generates string of '*' repeated 50 times  
1 str(100) * 10 #String '100' repeats 10 times
```



The \* operation of the string must be an integer. Otherwise, an error will occur.

#### TypeError

```
1 '*' * 3.1
```

---

Traceback (most recent call last)  
<ipython-input-5-5c37ed9b20aa> in <module>  
----> 1 '\*' \* 3.1

**TypeError**: can't multiply sequence by non-int of type 'float'



## One More Step

- ▶ Python is divided into **letters, numbers, sequences, and object data types**.
- ▶ Among them, both strings and numbers can use additional operators, but problems arise when values of different data types are added.
- ▶ Therefore, it is necessary to convert it into the same data type and proceed with the operation.

### ! TypeError

```
1 "A" + 1
```

Addition of number and string causes an error.

```
TypeError
<ipython-input-6-8c1ba8ce860b> in <module>
----> 1 "A" + 1
```

TypeError: can only concatenate str (not "int") to str

```
1 "10" + 10
```

"10" looks like a number, but since it is a character in the quotation mark, an error such as the above occurs.

```
TypeError
<ipython-input-7-f19a135c2b59> in <module>
----> 1 "10" + 10
```

TypeError: can only concatenate str (not "int") to str

# | Paper coding

**Try to fully understand the basic concept before moving on to the next step.**

**Lack of understanding basic concepts will increase your burden in learning this course, which may make you fail the course.**

**It may be difficult now, but for successful completion of this course we suggest you to fully understand the concept and move on to the next step.**

**Q1.** Use the int function to output the number 100 by adding the letter "50" and the number 50.  
Also, use the str function to output the result of the addition "5050".

Condition for Execution

The first line outputs the number 100, and the second line outputs the letter "5050".

Time

5minutes



Write the entire code and the expected output results in the note.

## Q2.

One letter '1' and three '0' are given. Use these two to make the number 1000. Here, only addition operations can be used between strings, and the int() function can be used only once.

Condition for Execution

Output the number 1000.

Time

5minutes



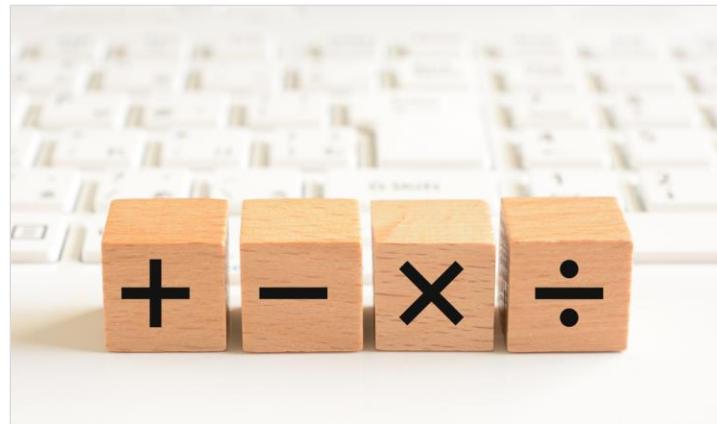
Write the entire code and the expected output results in the note.

## Unit 03. Basic of Numeric Data Types and Arithmetic Operation

| Let's code

# 1. Numeric Data Types, Arithmetic and Division

## 1.1. Arithmetic operation



- ▶ Implement several mathematical operations in Python, including addition, subtraction, multiplication, division, and square.

# 1. Numeric Data Types, Arithmetic and Division

## 1.1. Arithmetic operation

I Find out arithmetic calculation through coding.

1	1 + 5	6
2	1 - 8	-7
3	4 * 5	20
4	5 // 3	1

 Line 1

- Calculate addition

 Line 2

- Calculate subtraction

# 1. Numeric Data Types, Arithmetic and Division

## 1.1. Arithmetic operation

1	1 + 5
	6
2	1 - 8
	-7
3	4 * 5
	20
4	5 // 3
	1

### Line 3

- Calculate multiplication

### Line 4

- Floor Division(quotient): it returns 1, which is the quotient of dividing 5 by 3.

# 1. Numeric Data Types, Arithmetic and Division

## 1.2. Power, remainder

1	2	**	10
1024			
2	9	%	5
4			

### Line 1~2

- 1: Power,  $2^{10}$  is performed to return 1024.
- 2: It is called modulo. It returns the remainder of 9 divided by 5 which is 4.

# 1. Numeric Data Types, Arithmetic and Division

## 1.3. Mixed operation

1	1 + 2 * 5
	11
2	(1 + 2) * 5
	15



Line 1~2

- 1: Number operations have the same precedence as general arithmetic operations. In other words, multiplication and division are performed before addition.
- 2: If there is a parenthesis, it is calculated first.

# 1. Numeric Data Types, Arithmetic and Division

## 1.4. Complex number operation

```
1 ( 3 + 4j ) + ( 5 + 2j ) # Addition of complex numbers  
(8+6j)
```

```
2 ( 3 + 4j ) - ( 5 + 2j ) # Subtraction of complex numbers  
(-2+2j)
```

```
3 ( 3 + 4j ) * ( 5 + 2j ) # Multiplication of complex numbers  
(7+26j)
```

```
4 ( 3 + 4j ) / ( 5 + 2j ) # Division of complex numbers  
(0.793103448275862+0.48275862068965514j)
```

Line 1~4

- Addition, subtraction, multiplication, and division of complex numbers

## 2. Object Data Types, Operator Priorities, and Data Type Conversion

### 2.1. Identifying and converting data types

Identifying and converting a certain value of data type is a very important issue.

```
int("10") # Convert string "10" to integer 10
```



The data type is critical because the operator to be used varies depending on the data type.

## 2. Object Data Types, Operator Priorities, and Data Type Conversion

### 2.1. Identifying and converting data types

Find out data types using type built-in functions

```
1 type(10)
```

int

```
2 type("10")
```

str

```
3 type(10.5)
```

float

#### Line 1

- When a number (integer) is an input, it is marked as int.

#### Line 2

- When a string is an input, it is marked as str(string).

## 2. Object Data Types, Operator Priorities, and Data Type Conversion

### 2.1. Identifying and converting data types

| Find out data types using type built-in functions

```
1 type(10)
```

int

```
2 type("10")
```

str

```
3 type(10.5)
```

float

Line 3

- When a number (real number) with decimal points is an input, it is marked as float.



## One More Step

- In the cell, type (100) outputs int, but print (type (100)) outputs <class'int>. Class is a keyword defining an object, which is dealt with in detail in Unit 21.

```
1 type(10)      # Cell output is int
```

```
int
```

```
1 print(type(10)) # Cell output is <class 'int'>
```

```
<class 'int'>
```

## 2. Object Data Types, Operator Priorities, and Data Type Conversion

### 2.2. How to convert a data type

```
1 int("50")
```

```
50
```

```
2 int("50") + 20
```

```
70
```

```
3 "10" + str(10)
```

```
'1010'
```

```
4 float("10.5")
```

```
10.5
```

#### Line 1~2

- 1: Use the int function to replace letters with numbers.
- 2: Since the string "50" has been changed to an integer by int ("50"), +20 operation is performed.

## 2. Object Data Types, Operator Priorities, and Data Type Conversion

### 2.2. How to convert a data type

```
1 int("50")
```

```
50
```

```
2 int("50") + 20
```

```
70
```

```
3 "10" + str(10)
```

```
'1010'
```

```
4 float("10.5")
```

```
10.5
```

#### Line 3 ~ 4

- 3: To replace numbers with letters, use the str function.
- 4: Use the float function to replace the letter with a real number.

## 3. Strings and Operators

### 3.1. Addition and Multiplication Operators of String

- 1 The string may use operators such as addition and multiplication. Subtraction and division operators cannot be used.

```
1 "Hello" + "World"
```

```
'HelloWorld'
```

```
2 "text" + "repeat" * 10
```

```
'textrepeatrepeatrepeatrepeatrepeatrepeatrepeatrepeatrepeat'
```

Line 1

- String cannot be subtracted, but it can be added. Two strings can be added together to connect.

## 3. Strings and Operators

### 3.1. Addition and Multiplication Operators of String

| Find out the addition and multiplication operators of strings.

1 "Hello" + "World"

'HelloWorld'

2 "text" + "repeat" \* 10

'textrepeatrepeatrepeatrepeatrepeatrepeatrepeatrepeatrepeat'

Line2

- When a string is multiplied by an integer, the string repeats. Here, 'repeat' is repeated 10 times.
- However, division and subtraction do not work.

## 3. Strings and Operators

### 3.2. Single and Double Quotation Marks

- | When printing a string, the same result is obtained whether using single quotes such as 'Hello Python!!' or double quotes such as "Hello Python!!".

```
1 print("Hello World!")
2 print('Hello World!')
```

Hello World!  
Hello World!

#### Line 1~2

- The strings "~~" and "~~" show the same output result.

## 3. Strings and Operators

### 3.3. Using comma

- | When a string is to be an output, string and numerical value is printed using a comma.

```
1 print('I am', 23, 'years old!')  
2 print('I', 'really' * 3, 'love', 'Python!')
```

```
I am 23 years old!  
I really really really love Python!
```

#### Line 1~2

- String and numerical values were printed using commas.
- The comma automatically adds a space when outputting.

## 3. Strings and Operators

### 3.4. Escape sequence

I Escape sequence: \" or \' may be used to output double or small quotes on the screen.

```
1 print(\"double quotation marks\")
```

"double quotation marks"

```
1 print(\"'single quotation marks'\")
```

'single quotation marks'



#### Line 1~2

- String should be enclosed in double or double quotes.
- Therefore, it is difficult to indicate quotation marks. In this case, you can use \" or \'.
- An escape sequence refers to a special character used to control a computer, and since a double quote or a single quote is defined for defining a string, it is output using the control character \' or \' to output it.

## 3. Strings and Operators

### 3.4. Escape sequence

I Escape sequence: \n is used to change lines.

```
1 print("line\nbreak")
```

line  
break

Line 1

- <Inverse Slash+Character> is used to display special characters such as line change. \n is a line-switching character, and \t is a tab character.

## 3. Strings and Operators

### 3.4. Escape sequence

Escape sequence: \t is used when a tab character is to be input.

```
1 print("Enter a \t tab in the middle of the sentence.")
```

Enter a tab in the middle of the sentence.



Line 1

- When entering a tab character in the middle of a sentence, enter \t at the corresponding location.

## 3. Strings and Operators

### 3.4. Escape sequence

I Escape sequence: \t is used when a tab character is to be input.

```
1 print("\tEnter a tab at first!")
```

Enter a tab at first!

#### Line 1

- When entering a tab character in the start of a sentence, enter \t as a first output character.

## 3. Strings and Operators

### 3.5. String indexing

- | String indexing: String indexing may be used to obtain only some of the long strings. Use large brackets [] and numbers.

```
1 "hello"[0]
```

```
'h'
```

```
2 "hello"[2]
```

```
'l'
```

#### Line 1

- String indexing starts from 0.
- That is, when zero is indexed, h, which is the first letter of "hello", is indexed.

#### Line 2

- When 2 is used as an index in string indexing, you can see that the third letter l is indexed.

## 3. Strings and Operators

### 3.6. Negative string indexing

- String indexing: String indexing may be used to obtain only some of the long strings. Use large brackets [] and numbers.

```
1 "hello"[-1]  
'o'
```

Line1

- Python supports negative indexing, and when indexing -1, the last character is indexed.

## 3. Strings and Operators

### 3.6. Negative string indexing

String indexing: String indexing may be used to obtain only some of the long strings. It will be covered in detail later.

```
1 "hello"[-5]
```

```
'h'
```

Line 1

- Python supports negative indexing, and when indexing -5, the very first character is indexed.

## 3. Strings and Operators

### 3.6. Negative string indexing



Pay attention to the range of indexing.

```
: 1 "hello"[5]
```

```
IndexError Traceback (most recent call last)
<ipython-input-3-cdd7a2ddb114> in <module>
----> 1 "hello"[5]
```

IndexError: string index out of range



Line 1

- The string, 'hello' has 5 characters. Therefore indexing range is from 0 to 4.  
caution: index that is out of the range prints error.

Unit 4.

# Variables and Input

## Learning objectives

- ✓ Understand the concept of variables and be able to print variables and constants.
- ✓ Understand the data types of variables and the use of assignment operators and be able to use variables.
- ✓ Understand the concept and types of compound assignment operators and be able to do cumulative sum of variables.
- ✓ Be able to assign values and calculate variables using arithmetic operators.

## Learning overview

- ✓ Learn the concept of variables and their operations.
- ✓ Learn the data types of variables and strings.
- ✓ Learn the concept of arithmetic operators and the order of operations based on variables.
- ✓ Learn the cumulative sum of values using compound assignment operators .

## Concepts You Will Need to Know From Previous Units

- ✓ How to use the print() to print the data onto the monitor, and use the input() to receive the input value from the user.
- ✓ How to use the str() to change a number data type into a string data type.
- ✓ How to perform basic arithmetic operations using number data types.

# Keywords

Variable

Reserved  
Keywords

Assignment  
Operator

Compound  
Assignment Operator

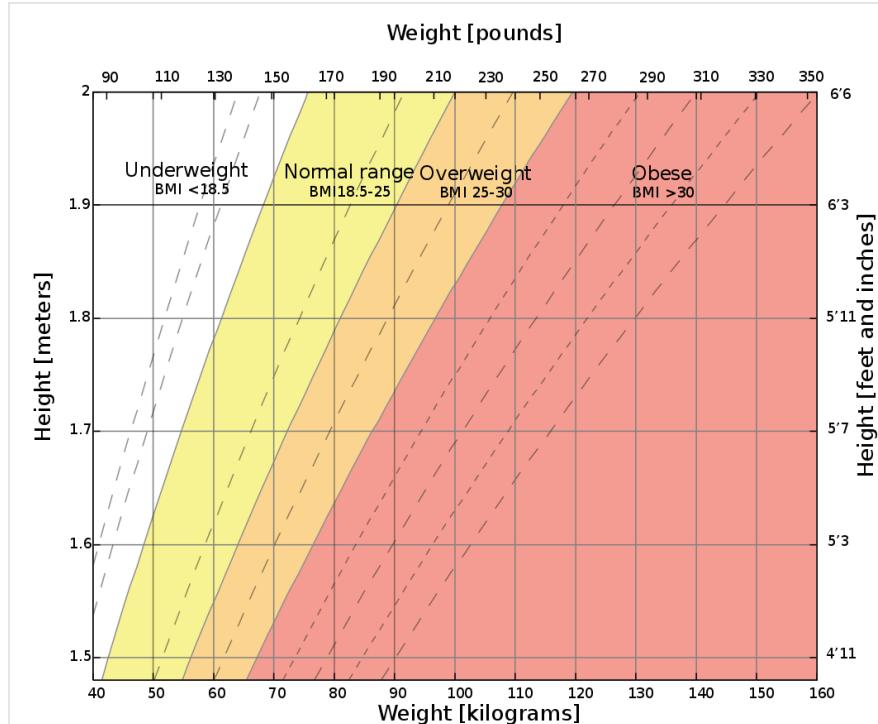
Arithmetic  
Operator

# Mission

## 1. Real world problem

### 1.2. What is the BMI?

| The Body Mass Index (BMI) measures human obesity with a formula that uses a person's height and weight.



[https://commons.wikimedia.org/wiki/  
File:Body\\_mass\\_index\\_chart.svg](https://commons.wikimedia.org/wiki/File:Body_mass_index_chart.svg)

- ▶ The function for calculating for the BMI ( $\text{kg}/\text{m}^2$ ) is  $(\text{weight})/(\text{height})^2$ . The units are kg for weight, and m for height.
  - Ex** The BMI of a person with a weight of 55 kg and a height of 168 cm( $=1.68 \text{ m}$ ) is  $55/(1.68 \text{ m})^2 = 19.4$ .
- ▶ The standard of obesity based on the BMI is mostly universal in the Western Hemisphere; however, the standard differs slightly in East Asian countries.

## 1. Real world problem

### 1.3. WHO BMI Standard Chart

WHO's standard for a normal BMI range is 18.5-24.9, and it can differ according to the person's country, region, and age.

BMI	Nutritional status
Below 18.5	Underweight
18.5–24.9	Normal weight
25.0–29.9	Pre-obesity
30.0–34.9	Obesity class I
35.0–39.9	Obesity class II
Above 40	Obesity class III

<https://www.euro.who.int/en/health-topics/disease-prevention/nutrition/a-healthy-lifestyle/body-mass-index-bmi>

## 2. Mission

### 2.1. What is a BMI CALCULATOR?

If you go to this website, [https://www.nhlbi.nih.gov/health/educational/lose\\_wt/BMI/bmicalc.htm](https://www.nhlbi.nih.gov/health/educational/lose_wt/BMI/bmicalc.htm), you can access the BMI Calculator provided by NIH (Note: it uses different standards than WHO)

**BMI Categories:**  
Underweight = <18.5  
Normal weight = 18.5-24.9  
Overweight = 25-29.9  
Obesity = BMI of 30 or greater

**What Next? Take Action Towards Better Health:**

[Maintain a Healthy Weight](#)

- Maintaining a healthy weight is important for your heart health.
- Learn more about [overweight and obesity](#)

[Increase Physical Activity](#)

- Moving more can lower your risk factors for heart disease.

[Eat a Heart-Healthy Diet](#)

- Eating a healthy diet is the key to heart disease prevention.

[Know and Control Your Heart Health Numbers](#)

- Tracking your heart health stats can help you meet your heart health goals.

Download the BMI calculator app today

- BMI's category of obesity is calculated by dividing the person's weight by the square of their height. The function is  $\text{kg}/\text{m}^2$
- According to this website, a BMI of 18.5 and under is categorized as Underweight, when a BMI is in the range of 18.5-24.9, it is considered Normal, while a BMI range of 25.0-29.9 is Overweight, and 30.0 and above is considered Obese.

**Ex** If Height is 170 cm and Weight is 73 kg,

the formula is:  $73 / (1.7 \times 1.7) = 25.26 \rightarrow \text{Overweight}$

## 2. Mission

### 2.2. How the BMI CALCULATOR works

```
1 name = input("Enter name: ")  
2 height_cm = int(input("Enter height(cm): "))  
3 height_m = height_cm / 100  
4 weight = int(input("Enter weight(kg): "))  
5 bmi = weight / (height_m ** 2)  
6 print(name, "'s BMI is", bmi.)
```

Enter name" David  
Enter height(cm): 170  
Enter weight(kg): 70  
David's BMI is 24.221453287197235.

## 2. Mission

### 2.3. Programming plan

#### Pseudocode

- [1] Start
- [2] Enter Name
- [3] Enter Height (cm)
- [4] Convert the unit of height from (cm) to (m)
- [5] Enter Weight (kg)
- [6] Calculate for BMI
- [7] Print the result on monitor
- [8] End

#### Flowchart



## 2. Mission

### 2.4. BMI CALCULATOR final code

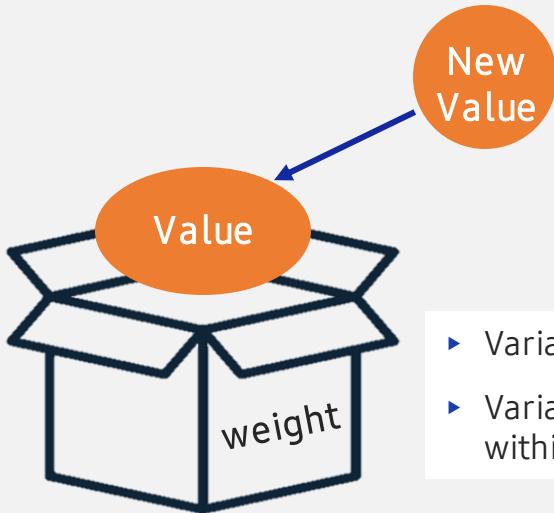
```
1 name = input("Enter name: ")  
2 height_cm = int(input("Enter height(cm): "))  
3 height_m = height_cm / 100  
4 weight = int(input("Enter weight(kg): "))  
5 bmi = weight / (height_m ** 2)  
6 print(name, "'s BMI is", bmi.)
```

# | Key concept

## 1. Variable Declaration

### 1.1. What is a variable

During programming, a single mistake in multiple value conversions can result in an error. The use of **variables** can simplify this process.



- ▶ Variables are a key concept in programming.
- ▶ Variables are used to store certain values within the computer.

The value inside the box can be changed constantly. It is best to understand this value as **data**. This data can hold a single value, such as `weight = 78.7`, and it can also hold one or more values like `person = ('David', 78.7)`.

## 1. Variable Declaration

### 1.2. Grammar of variable declarations

Containers for storing data values are called “Variables.” Unlike other programming languages, such as C or Java that require specific methods, Python does not require a variable data type declaration.

**Ex** Assume that we store one’s weight as the variable “weight.” In order to do this, we perform the following:  
the name and weight can be stored together in a () .

```
1 weight = 78.7  
2 person = ('David', 78.7)
```

#### Line 1, 2

- When the code is executed, a variable with the name “weight” is created in the memory and the value 78.7 is stored.
- In the variable “person”, the values ('David', 78.7) are stored. (These types of data are called Tuples).
- The ‘=’ in the code above does not mean ‘equal’, but ‘to save the value on the right with the name (weight) on the left.’ This operator ‘=’ is called an assignment operator.

## 1. Variable Declaration

### 1.2. Grammar of variable declarations

 Focus Assigning and recalling values in variables minimizes errors and makes editing programs much easier.

```
1 radius = 4.0
2 print('Radius of Circle', radius)
3 print('Area of Circle', 3.14 * radius * radius)
4 print('Circumference of Circle', 2.0 * 3.14 * radius)
```

Radius of Circle 4.0

Area of Circle 50.24

Circumference of Circle 25.12

## 1. Variable Declaration

### 1.2. Grammar of variable declarations

**I Identifier:** We will learn to work with various data, but in order to do so, we need different names to differentiate the data within the code. Just as one differentiates people by using names, such as "James" and "Michael", identifiers play the role of differentiating variables.



- ▶ Identifiers must be written with the following rules.
  - Identifiers can be a combination of letters or digits or an underscore (\_), but they cannot consist of any special symbols.
  - Identifiers cannot start with a digit.
  - Identifiers cannot contain spaces or tabs.
  - Identifiers are case sensitive. Therefore, the variable index and INDEX are different variables.
  - Identifiers cannot use one of the Python Reserved Words (or Keywords)

## 1. Variable Declaration

### 1.3. Variable declarations and Identifiers

**I Example of identifier:** Calculate the area of a rectangle using its width and height, where width is 10, and height is 5. It is possible to use the “width” identifier to save 10 and the “height” identifier to save 5, and then go ahead with the calculation. This way, identifiers make it easier to differentiate variables.

```
1 width = 10  #Identifier to differentiate the width of the rectangle from other data
2 height = 5  #Identifier to differentiate the height of the rectangle from other data
3 rectangle_area = width * height
4 print('rectangle_area =', rectangle_area)
```

```
rectangle_area = 50
```



Line 1, 2

- Using the “width” and “height” identifiers, we can differentiate the width and height of the rectangle from other data.

## 1. Variable Declaration

### 1.4. Python Keywords

While we can freely name identifiers, we cannot use Python's built-in keywords as identifiers. Moreover, keywords cannot be used as variable names.

#### Python's Built-In Keywords which cannot be used as identifiers

False	class	return	is	finally	None
if	for	lambda	continue	True	def
from	while	nonlocal	and	del	global
not	with	as	elif	try	or
yield	assert	else	import	pass	break
except	in	raise			

## 1. Variable Declaration

### 1.4. Python Keywords

I Using a Keyword to create a variable will result in an error, as shown below.



```
1 global = 500
File "<ipython-input-6-6cbed63281ae>", line 1
    global = 500
          ^
SyntaxError: invalid syntax
```

## 1. Variable Declaration

### 1.5. Tips for Python Keywords



TIP

- ▶ How to Search for Python Keywords:
  - The list of Python Keywords can be searched using the following method.  
(※ The "import" function will be covered later.)

Ex

```
1 import keyword
2
3 # Search for Python Keywords
4 print("Python keyword list...")
5 print(keyword.kwlist)
```

Python keyword list...

```
['False', 'None', 'True', 'and', 'as', 'assert', 'async', 'await', 'break', 'class', 'continue', 'def', 'del', 'elif', 'else', 'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']
```



## One More Step

- ▶ How do you name variables?
  - When naming variables, it's important to use a name that best describes its role.
  - Good variable names make the entire program easier to read. On the other hand, careless and spontaneous variable names can make the program very difficult to read.

**Ex** It's more comprehensible to name weight and height as "weight" and "height", rather than x1 and x2.

```
1 weight = 78.2
2 height = 180.0
```



```
1 x1 = 78.2
2 x2 = 100.0
```



## 1. Variable Declaration

### 1.6. String

- | So far, we have only stored integers and real numbers as variables. However, it is also possible to store strings. In the programming world, text data is called a **string**.
- | To create a string, you need to use either single quotes (') or double quotes ("). So, "Hello World!" and 'Hello World!' are the same strings.

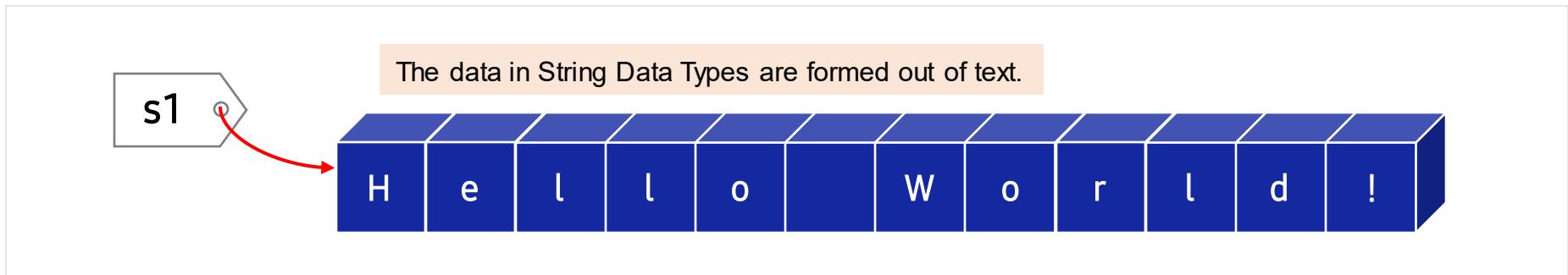
```
1 s1 = 'Hello World'  
2 print(s1)
```

```
Hello World
```

## 1. Variable Declaration

### 1.6. String

- | In Python, string data such as the examples shown above is called as “objects.”
- | In the following example, the variable s1 would be the “tag” that refers to this object. “Objects” refer to the data and functions used in Python Programming.



## 1. Variable Declaration

### 1.6. String

Strings can perform addition operations by combining two strings with “+,” as shown below.

```
1 s1 = "Hello"  
2 s2 = "World!"  
3 print(s1 + s2)
```

HelloWorld!

## 1. Variable Declaration

### 1.7. Python Data Type

| Data Types: Definition and Data Type Check

#### Data Type

- ▶ A Data Type is a type of data that can be processed in programming languages.
- ▶ Basic Data Types include Booleans, Numbers (Integers, Real Numbers, Complex Numbers), Strings, Lists, Tuples, Sets, and Dictionaries.
- ▶ The `type()` function allows one to check the data type of an object.

## 1. Variable Declaration

### 1.7. Python Data Type

| Dynamic typing can be used to assign variables to different data types, such as shown below.

```
1 num = 85
2 print(type(num))
3
4 pi = 3.14159
5 print(type(pi))
6
7 message = "Good morning"
8 print(type(message))
```

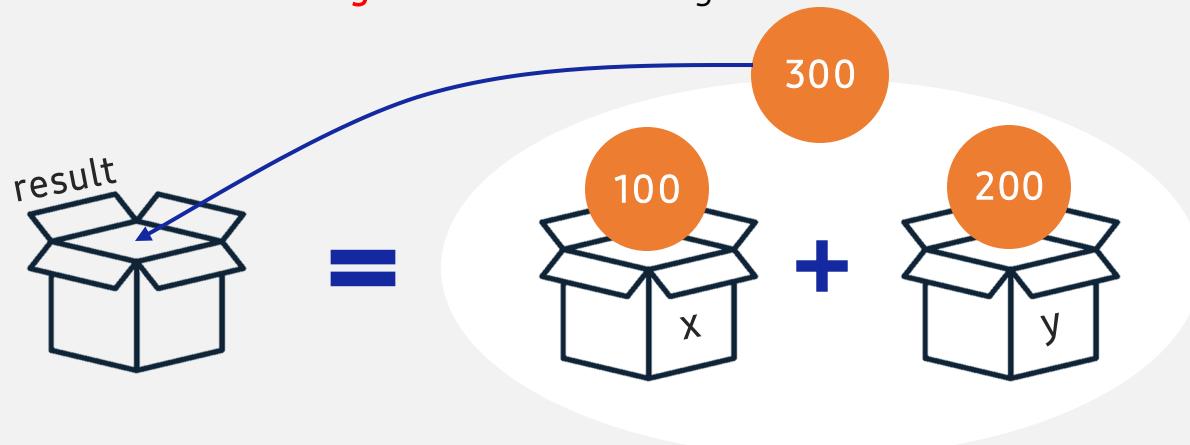
```
<class 'int'>
<class 'float'>
<class 'str'>
```

## 1. Variable Declaration

### 1.8. Assignment operator in Python

Take a look at this expression 'result = 100 + 200'.

This expression means that we **assign** the value on the right to the variable on the left.



The '=' sign in the code 'RESULT = 100+200' does not mean 'equal'. It means 'to store the value on the right of the = operator in the variable *result*' on the left of =.  
This '=' operator is called the **assignment operator**.

## 2. Variable Assignment and Reference

### 2.1. Python simultaneous assignment



As shown in the expression `x, y = 100, 200`, it is possible to assign multiple variables to values in one line simultaneously. This is called **simultaneous assignment**.

```
1 x, y = 100, 200  
2 result = x + y  
3 print(result)
```

300

## 2. Variable Assignment and Reference

### 2.2. Python multiple assignment

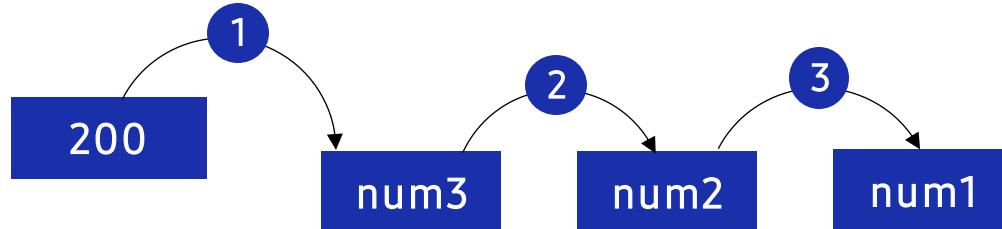
| As shown in num1 = num2 = num3 = 200, you could assign 200 to multiple variables.

```
1 num1 = num2 = num3 = 200  
2 print(num1, num2, num3)
```

200 200 200

#### Line 1

- 200 is assigned to num1, num2, and num3. The order of assignment starts from the right as shown below.



The order of assignment is as shown above.

## 2. Variable Assignment and Reference

### 2.3. Equal sign in Python

 The equal sign does not mean that two values are equal.

```
1 300 = 300
```

```
File "<ipython-input-8-b44a4a79e963>", line 1  
 300 = 300  
^
```

**SyntaxError:** cannot assign to literal

- ▶ The equal sign '=' does not mean two values are equal, but that the value on the right is assigned to the value on the left. Therefore, because the value 300 cannot be assigned to the value 300, it results in an error.
- ▶ Be sure that you do not confuse this with the comparison operator "=="

## 3. Compound Assignment Operators

### 3.1. A variety of compound Assignment Operators in Python

- | There are a variety of compound assignment operators in Python. Even if you are not familiar with compound assignment operators, you can still use arithmetic operators and assignment operators to code without a problem.
- | Not knowing any of the compound assignment operators that programmers use frequently, however, could be a problem when reading someone else's code.

Compound Assignment Operators

Operator	Description	Example
<code>+=</code>	Combination of the Addition Operator and the Simple Assignment Operator	<code>i+=10</code>
<code>-=</code>	Combination of the Subtraction Operator and the Simple Assignment Operator	<code>i-=10</code>
<code>*=</code>	Combination of the Multiplication Operator and the Simple Assignment Operator	<code>i*=10</code>
<code>/=</code>	Combination of the Division Operator and the Simple Assignment Operator	<code>i/=10</code>
<code>^=</code>	Combination of the Bitwise XOR(^) Operator and the Simple Assignment Operator	<code>i ^=10</code>
<code>%=</code>	Combination of the Modulo Operator and the Simple Assignment Operator	<code>i %=10</code>

### 3. Compound Assignment Operators

#### 3.1. A variety of compound Assignment Operators in Python



It's important to know Compound Assignment Operators.

You may not be able to read and understand someone else's code if you do not know any of the Compound Assignment Operators.

```
1 num = 200
2 num += 100
3 print(num)
4 num -= 100
5 print(num)
6 num *= 20
7 print(num)
8 num /= 2
9 print(num)
```

```
300
200
4000
2000.0
```

## 4. Arithmetic Operator

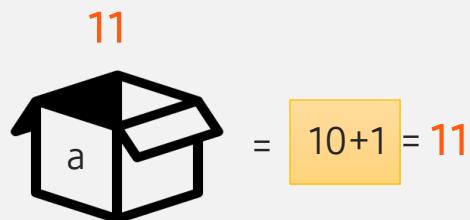
### 4.1. Arithmetic operators in Python

I Arithmetic Operators perform addition, subtraction, multiplication, division, modulo, and exponentiation operations.

Proper numerical expressions can increase calculation efficiency and code simplification.

In an expression such as `a = 10 + 1`, the `10+1` on the right side of the equal sign is the *evaluation value*. This value is calculated and assigned to `a`. Such arithmetic operated by the Arithmetic Operator and assigned to a variable is very common.

$$a = 10+1$$



## 4. Arithmetic Operator

### 4.2. How do arithmetic operators work?

In an expression  $a + b$ , the  $+$  is an operator while  $a, b$  are called operands. The Python Operators and their operations are described in the table below.

Operator	Meaning	Operation
$+$	Addition	Add the left operand and the right operand
$-$	Subtraction	Subtract the right operand from the left operand
$*$	Multiplication	Multiply the left operand and the right operand
$/$	Real Division	Divide the left operand by the right operand Division in Python will return real values by default
$//$	Integer Division(Quotient)	Unlike the $/$ , the Integer Division is used to discard the decimal point of the quotient and only obtain the integer
$\%$	Remainder/Modulo	Read as Modulo Operator. Irrelevant of percentage or ratio. Returns the remainders of division.
$**$	Exponentiation	Multiply the left operand to the power of the right operand

## 4. Arithmetic Operator

### 4.2. How do arithmetic operators work?

Hint: Square root of 4 can be obtained by `4 ** 0.5`. The result is 2.

Operator	Sign	Example	Value Result
Addition	+	<code>7 + 4</code>	11
Subtraction	-	<code>7 - 4</code>	3
Multiplication	*	<code>7 * 4</code>	28
Real Division	/	<code>7 / 4</code>	1.75
Integer Division (Quotient)	//	<code>7 // 4</code>	1
Modulo	%	<code>7 % 4</code>	3
Exponentiation	**	<code>7 ** 2</code>	49

## 4. Arithmetic Operator

### 4.2. How do arithmetic operators work?



Let's take a look at what data type values each variable receives and how it is calculated using that variable.

The following code converts the input value into a float data type. Int(input('...')) is used when converting into an integer type.

```
1 weight = float(input("Enter weight in kg: "))
2 height = float(input("Enter height in m: "))
3
4 bmi = (weight / (height ** 2))
5 print("Your BMI=", bmi)
```

```
Enter weight in kg: 70
Enter height in m: 1.75
Your BMI= 22.857142857142858
```

## 4. Arithmetic Operator

### 4.3. input function of Python and arithmetic operators

| The int(input(' ')) is often used to receive the user's input and calculating it by converting it into an integer.

| Solve the problem of returning change when there are only 500 and 100 won.

※ Won is the Korea currency.

```
1 money = int(input("Money Paid : "))
2 price = int(input("Price of Item : "))
3
4 change = money - price
5 print("Amount of Change : ", change)
6 coin500s = change // 500
7 change = change % 500
8 coin100s = change // 100
9
10 print("Pieces of 500 won coins: ", coin500s)
11 print("Pieces of 100 won coins: ", coin100s)
```

```
Money Paid: 5000
Price of Item: 300
Amount of Change: 4700
Pieces of 500 won coins :9
Pieces of 100 won coins: 2
```

#### Line 1-2

- Receive the amount of money and price from the user and convert them into the int type.
- After that, the problem can be solved by integer operation.

## 4. Arithmetic Operator

### 4.3. input function of Python and arithmetic operators

| After receiving the input, the desired result can be obtained by using the operator.

```
1 money = int(input("Money Paid : "))
2 price = int(input("Price of Item : "))
3
4 change = money - price
5 print("Amount of Change : ", change)
6 coin500s = change // 500
7 change = change % 500
8 coin100s = change // 100
9
10 print("Pieces of 500 won coins: ", coin500s)
11 print("Pieces of 100 won coins: ", coin100s)
```

```
Money Paid: 5000
Price of Item: 300
Amount of Change: 4700
Pieces of 500 won coins :9
Pieces of 100 won coins: 2
```

#### Line 4-8

- The value for change is money – price.
- In coin500s, the change is divided by 500 and then stored.

## 4. Arithmetic Operator

### 4.3. input function of Python and arithmetic operators

| After receiving the input, the desired result can be obtained by using the operator.

```
1 money = int(input("Money Paid : "))
2 price = int(input("Price of Item : "))
3
4 change = money - price
5 print("Amount of Change : ", change)
6 coin500s = change // 500
7 change = change % 500
8 coin100s = change // 100
9
10 print("Pieces of 500 won coins: ", coin500s)
11 print("Pieces of 100 won coins: ", coin100s)
```

```
Money Paid: 5000
Price of Item: 300
Amount of Change: 4700
Pieces of 500 won coins :9
Pieces of 100 won coins: 2
```

#### Line 4-8

- Put the remainder of the change divided by 500 in change.
- In coin100s, change is divided by 100 then stored.

## 4. Arithmetic Operator

### 4.4. Caution for using arithmetic operators

 Using the wrong equation will result in an error.

```
1 num1 = 3
2 num2 = 0
3 print(num1/num2)
```

```
ZeroDivisionError                                 Traceback (most recent call last)
<ipython-input-18-b729e0caa109> in <module>
      1 num1 = 3
      2 num2 = 0
----> 3 print(num1/num2)
```

`ZeroDivisionError`: division by zero

- ▶ When using the arithmetic operator, be sure to check whether there are any mistakes in the equation.
- ▶ If the numerical value is divided by 0, an error will occur. Read the `ZeroDivisionError` carefully.

# | Paper coding

**Try to fully understand the basic concept before moving on to the next step.**

**Lack of understanding basic concepts will increase your burden in learning this course, which may make you fail the course.**

**It may be difficult now, but for successful completion of this course we suggest you to fully understand the concept and move on to the next step.**

**Q1.** Assign the values 30, 60 to the variables width and height, respectively.  
Write a program that uses these two variables to find the area of the rectangle as shown below.

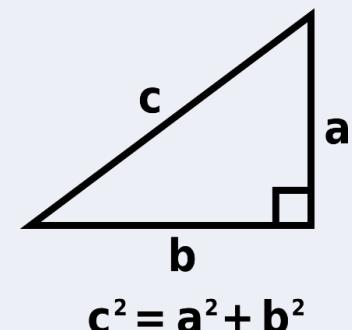
Conditions for Execution	"Print "Area of Rectangle : 1800."
Time	5 Minutes

 Write the entire code and the expected output results in the note.

**Q2.**

The Pythagorean Theorem states that the square of the hypotenuse c for any right triangle is equal to the square of the base a plus the square of the height b. Write code that calculates the length of the hypotenuse by receiving the base and height as integers.

Conditions for Execution	Enter the bottom of the first line and the height of the second line. In the third line, print out the result of calculating the length of the hypotenuse.
Time	5 Minutes
The Pythagorean Theorem	$c^{**2} = a^{**2} + b^{**2}$



- Write the entire code and the expected output results in the note.

| Let's code

## 1. Receiving String Input From User: `input()` function

### 1.1. Using `input` function

| The user's name is stored in the variable name. Use this variable to write a code that will print the following.

```
1 name = input("Enter Name: ")  
2 print("Hello", name)  
3 print("Welcome to the World of Python Programming.")
```

## 1. Receiving String Input From User: `input()` function

### 1.1. Using `input` function

| Check the Results

```
1 name = input("Enter Name: ")  
2 print("Hello",name.)  
3 print("Welcome to the World of Python Programming.")
```

Enter Name: David  
Hello, David.  
Welcome to the World of Python Programming.

## 1. Receiving String Input From User: `input()` function

### 1.2. Receiving Integers From Users

- | Generate a program that executes arithmetic operation by receiving input from the user
- ▶ In the addition program learned earlier, the result of 100+200 was always printed.
  - ▶ It will be a more useful program to input two integers, add them, and then print out the results.

```
1 x = int(input("Enter the first integer: "))
2 y = int(input("Enter the second integer: "))
3 s = x + y
4 print("The sum of, " x, "and" y, "is", s)
```

```
Enter the first integer: 300
Enter the second integer: 400
The sum of 300 and 400 is 700
```

- ▶ The `input()` function is used to input integers. However, the `input()` function always returns the input in the form of a string. Therefore, the string “300” is returned.
- ▶ In order to convert the string “300” into an integer 300, the return value of the `input()` function must be enclosed in `int()`. The `int()` function converts the string into an integer.

## 1. Receiving String Input From User: `input()` function

### 1.2. Receiving Integers From Users

If you enter characters, or a String of characters such as 'hundred' instead of the number 300 as follows, the `int()` function cannot be converted into an integer, and it will result in a Runtime error.

#### Watch Out For the `ValueError`

```
1 #If the characters "hundred" are entered, the int() function fails to convert it into an integer and will print an error.
2 x = int(input("Enter the first integer: "))
3 y = int(input("Enter the second integer: "))
4 s = x + y
5 print("The sum of," x, "and" y, "is", s)
```

Enter the first integer: hundred

```
-----  
ValueError                                     Traceback (most recent call last)  
<ipython-input-2-611617428447> in <module>  
      1 #If the characters "hundred" are entered, the int() function fails to convert it into an integer and will print an error.  
----> 2 x = int(input("Enter the first integer: "))  
      3 y = int(input("Enter the second integer: "))  
      4 s = x + y  
      5 print("The sum of," x, "and" y, "is", s)  
  
ValueError: invalid literal for int() with base 10: 'hundred'
```

 One More Step

- ▶ Python can check the data type with the following type function. This data type is only numbers, so x=int(x) is possible. It is also possible to check in advance whether the data type conversion is possible with x.isdigit() == True. These functions will be covered later.

```
1 x = input("Enter the first integer: ")  
2 print('Input Data Type: ', type(x))
```

Enter the first integer: 100  
Input Data Type: <class 'str'>

```
1 # Let's check whether x is only a number.  
2 if x.isdigit() == True :  
3     x = int(x)  
4     print('Input Data Type: ', type(x))  
5 else:  
6     print('x is not a number type.')
```

Input Data Type: <class 'int'>

## 2. Introduction of Variables

### 2.1. Using variables for efficient programming

| When writing a program, a name for storing data values is needed. The name is what we call a variable.



`radius = 4.0`

`name = input('input your name')`

| By introducing variables, complex tasks can be made simpler. You can store values in variables and then use variables instead of the values.

## 2. Introduction of Variables

### 2.1. Using variables for efficient programming

| Receiving String Input: Find out how to store the string value in the variable name.



## 2. Introduction of Variables

### 2.1. Using variables for efficient programming

| Let's get familiar with using variables through the following codes.

```
1 radius = 4.0
2 print('Radius of Circle', radius)
3 print('Area of Circle', 3.14 * radius * radius)
4 print('Circumference of Circle', 2.0 * 3.14 * radius)
```

Radius of Circle 4.0

Area of Circle 50.24

Circumference of Circle 25.12

```
1 name = input('input your name: ')
```

input your name:

```
1 name = input('input your name: ')
```

input your name: David

## 2. Introduction of Variables

### 2.2. Variable and data type

#### I Variable Type

- ▶ Each variable must secure sufficient space in memory to represent its value.
- ▶ In order to determine the amount of space needed and read/write values in that space, the data type must be determined. There is no need, however, to specify the type as Python determines the most suitable type on its own.

```
1 name = 'David'  
2 age = 27  
3 print(type(name))  
4 print(type(age))
```

```
<class 'str'>  
<class 'int'>
```

## 2. Introduction of Variables

### 2.3. Caution for TypeError when using variables

| A TypeError occurs when a string, such as the variable my\_height, is added to an integer, such as number 1.

#### TypeError

```
1 my_age = 22
2 my_height = '177'
3 my_age = my_age + 1
4 my_height = my_height + 1
5 print(my_age, my_height)
```

```
-----  
TypeError: can only concatenate str (not "int") to str  
<ipython-input-9-1e339f59d7ef> in <module>  
    2 my_height = '177'  
    3 my_age = my_age + 1  
----> 4 my_height = my_height + 1  
    5 print(my_age, my_height)
```

# | Pair programming



# Pair Programming Practice

## Guideline, mechanisms & contingency plan

Preparing pair programming involves establishing guidelines and mechanisms to help students pair properly and to keep them paired. For example, students should take turns “driving the mouse.” Effective preparation requires contingency plans in case one partner is absent or decides not to participate for one reason or another. In these cases, it is important to make it clear that the active student will not be punished because the pairing did not work well.

## Pairing similar, not necessarily equal, abilities as partners

Pair programming can be effective when students of similar, though not necessarily equal, abilities are paired as partners. Pairing mismatched students often can lead to unbalanced participation. Teachers must emphasize that pair programming is not a “divide-and-conquer” strategy, but rather a true collaborative effort in every endeavor for the entire project. Teachers should avoid pairing very weak students with very strong students.

## Motivate students by offering extra incentives

Offering extra incentives can help motivate students to pair, especially with advanced students. Some teachers have found it helpful to require students to pair for only one or two assignments.



# Pair Programming Practice



## | Prevent collaboration cheating

The challenge for the teacher is to find ways to assess individual outcomes, while leveraging the benefits of collaboration. How do you know whether a student learned or cheated? Experts recommend revisiting course design and assessment, as well as explicitly and concretely discussing with the students on behaviors that will be interpreted as cheating. Experts encourage teachers to make assignments meaningful to students and to explain the value of what students will learn by completing them.

## | Collaborative learning environment

A collaborative learning environment occurs anytime an instructor requires students to work together on learning activities. Collaborative learning environments can involve both formal and informal activities and may or may not include direct assessment. For example, pairs of students work on programming assignments; small groups of students discuss possible answers to a professor's question during lecture; and students work together outside of class to learn new concepts. Collaborative learning is distinct from projects where students "divide and conquer." When students divide the work, each is responsible for only part of the problem solving and there are very limited opportunities for working through problems with others. In collaborative environments, students are engaged in intellectual talk with each other.

**Q1.** Receive a radius value from the user and print the circumference and area of a circle with this radius that follows. Use the variable PI= 3.141592 to obtain this value.

**Example  
Result**

Enter the radius of a circle:11

Circumference of a circle= 69.115024, Area of a circle= 280.132632

- ▶ Circumference of a circle:  $2 * \text{radius} * \text{PI}$
- ▶ Area(Surface) of a circle:  $\text{PI} * \text{radius} * \text{radius}$
- ▶ Study the area and circumference of a circle before moving forward.

**Q2.**

Write a program that displays the squared values from 2 to 6 in a table as the following. As shown below,  $a$  can be increased from 2 to 6, and  $n$  has a value of 2. Input the actual value for the part corresponding to  $a^{**}n$ , so that the output result of the equation is  $2^{**}2$ .

Example Result	a	n	a ** n
	2	2	4
	3	2	9
	4	2	16
	5	2	25
	6	2	36

Unit 5.

# **Logic and Comparison Operations**

## Learning objectives

- ✓ Understand the Boolean data type and be able to convert other data types into Booleans.
- ✓ Compare the size relation of two values through comparison operators.
- ✓ Understand the function and role of logical operators and practice using them.
- ✓ Understand and practice comparing number values and string values using comparison and logical operators and determine whether its results are True or False in Boolean value.

## Learning overview

- ✓ Learn about the Boolean data type.
- ✓ Learn about comparison operators, which is a way of comparing whether one value is greater, lesser, or equal to another.
- ✓ Learn the concept of logical operators and its precise role and function.

## Concepts You Will Need to Know From Previous Units

- ✓ Understand integers, real numbers, and string data types and declare/assign appropriate variables.
- ✓ Be familiar with compound arithmetic assignment operation and be able to read the code.
- ✓ Know the priorities of each operator and apply them based on these concepts.

# Keywords

Boolean  
Data Types

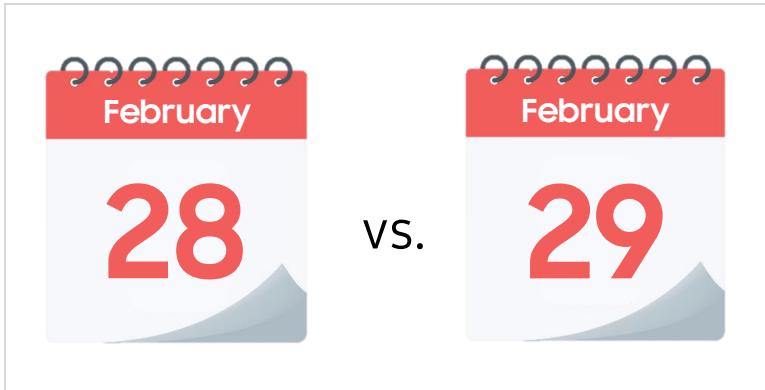
Comparison  
Operator

Logical  
Operator

# Mission

## 1. Real world problem

### 1.1. Origin of leap year



- ▶ The length of one year is not exactly 365 days. A leap year is a year in which an extra day is added to match to the actual solar year. February has 28 days, but in the leap year, it has 29 days.
- ▶ The current Gregorian calendar used in most countries around the world complements the Julian calendar, which adds one day every four years (February 29<sup>th</sup>), subtracting three days (three leap years) from the 400<sup>th</sup> to reduce the deviation from the solar calendar.
- ▶ Let's make a [Leap Year Calculator](#)

## 1. Real world problem

### 1.2. Rules of Leap Years for the Leap Year Calculator

- | If the year is divisible by 4, it is a leap year.
- | Even if the number of years is divided by 4, it should not be divided by 100.
- | The year in which the number of years is divisible by 400 is unconditionally a leap year.

## 2. Mission

### 2.1. How the Leap Year Calculator Works

```
1 year = int(input("Enter the year: "))
2 leap_year = (year % 4 == 0 and year % 100 != 0) or year % 400 == 0
3 print(leap_year)
```

```
Enter the year: 2021
False
```

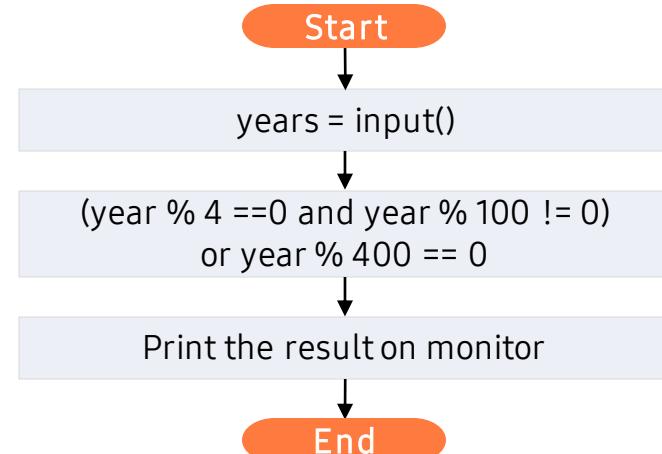
## 2. Mission

### 2.2. Programming plan

#### Pseudocode

- [1] Start
- [2] Enter the year
- [3] Use the leap year formula to determine whether it is true or false. The formula is  $(\text{year} \% 4 == 0 \text{ and } \text{year} \% 100 != 0) \text{ or } \text{year} \% 400 == 0$
- [4] If any of them is false, it will print False.
- [5] End

#### Flowchart



## 2. Mission

### 2.3. Leap year calculation final code

```
1 year = int(input("Enter the year: "))
2 leap_year = (year % 4 == 0 and year % 100 != 0) or year % 400 == 0
3 print(leap_year)
```

# | Key concept

## 1. Boolean Operation: Logic and Comparison Operators

### 1.1. Comparison operator

**Comparison Operator** takes the result of the comparison operation of numerical data, such as “greater than” or “less than,” and returns it as True or False.



## 1. Boolean Operation: Logic and Comparison Operators

### 1.1. Comparison operator

- | **Comparison Operator** examines the size relationship of two comparable operands, such as numbers and strings, in terms such as "greater than" or "less than."
- | This result is returned as True or False. The data type that has True or False values is called Booleans.

```
1 print('Result of 10 > 5: ', 10 > 5)
2 print('Result of 5 < 1: ', 5 > 1)
3 print('Result of 5 == 5: ', 5 == 5)
4 print('Result of 5 != 5: ', 5 != 5)
5 print("Result of 'a' > 'b': ", 'a' > 'b')
```

Result of 10 > 5: True  
Result of 5 < 1: False  
Result of 5 == 5: True  
Result of 5 != 5: False  
Result of 'a' > 'b': False

## 1. Boolean Operation: Logic and Comparison Operators

### 1.2. Boolean data type

| **Boolean**: a data type consisting of True and False values.

| It is also called the logical data type and is used to indicate True and False. As seen below, in Python the integer 1 is converted into True and 0 is converted into False.

```
1 print(bool(1))
2 print(bool(0))
3 print(bool(True))
4 print(bool(False))
```

```
True
False
True
False
```

 One More Step

- ▶ Boolean Data Type Print Test

```
1 print(bool(-1))
2 print(bool(0))
3 print(bool(None))
4 print(bool(''))
5 print(bool('hello'))
6 print(bool([]))
7 print(bool([10, 20]))
```

True  
False  
False  
False  
True  
False  
True

- ▶ First, the number 0, when converted into a Boolean type, becomes False. All the other numbers except 0, when converted into a Boolean type, become True.
- ▶ White spaces such as '' are returned as False in Boolean. All other characters besides the white space are returned as True.
- ▶ None is a keyword that means there is no value.
- ▶ [] is an object called a list. If there is no value in this object, it is returned as False.

## 1. Boolean Operation: Logic and Comparison Operators

### 1.3. Description about comparison operators

| Let's take a closer look at comparison operators and their explanations and results.

Operator	Description	a = 100 b = 200
==	If two operands have the same value, return True	a == b is False
!=	If two operands have different values, return True	a != b is True
>	If the left operand is greater than the right operand, return True	a > b is False
<	If the left operand is less than the right operand, return True	a < b is True
>=	If the left operand is greater than or equal to the right operand, return True	a >= b is False
<=	If the left operand is less than or equal to the right operand, return True	a <= b is True

## 1. Boolean Operation: Logic and Comparison Operators

### 1.4. Code examples of comparison operators

 Focus The returned values are Boolean values when comparison operators have been applied.

```
1 a = 100
2 b = 200
3 print(a == b)
4 print(a != b)
5 print(a > b)
6 print(a < b)
7 print(a >= b)
8 print(a <= b)
```

```
False
True
False
True
False
True
```

- | The == operator returns True only if the values are the same. For example, in the statement a==b, the variable a is 100 and variable b is 200, so the values are not the same. Therefore, False is returned.
- | The second comparison operator (!=) returns True if the values are not the same. So, in the example, variables a and b, with values 100 and 200 respectively, are not equal. Therefore a != b returns True as a result.
- | As such, comparison operators allow values to be compared in greater than, less than, greater than or equal to, and less than or equal to.

## 1. Boolean Operation: Logic and Comparison Operators

### 1.4. Code examples of comparison operators

| Comparison operators return Boolean data types.

```
1 n = int(input('Enter the integer: '))
2 print('Is the integer even?', n % 2 == 0)
```

Enter the integer: 60  
Is the integer even? True

 Line 2

- If the remainder divided by 2 is 0, return True; if not, return False.

## 1. Boolean Operation: Logic and Comparison Operators

### 1.5. String and comparison operators

In a string, == returns True if the two strings match completely.

```
1 print('aaa' == 'aaa')
2 print('aaa' == 'bbb')
3 print('aaa' != 'aaa')
4 print('aaa' != 'bbb')
```

True  
False  
False  
True

#### Line 1-2

- The first line returns True because the two strings are completely equal.
- The second line returns False because the two strings are not equal.

## 1. Boolean Operation: Logic and Comparison Operators

### 1.5. String and comparison operators

In a string, != returns True if the two strings do not match completely.

```
1 print('aaa' == 'aaa')
2 print('aaa' == 'bbb')
3 print('aaa' != 'aaa')
4 print('aaa' != 'bbb')
```

True  
False  
False  
True

#### Lines 3-4

- The third line returns False because the two strings match completely.
- The fourth line returns True because the two strings do not match.

## 1. Boolean Operation: Logic and Comparison Operators

### 1.6. Difference between == and is

The command "is" checks whether the two objects are the same.

```
1 a = 1
2 b = 1.0
3 print(a == b)
4 print(a is b)
```

True  
False

#### Line 3-4

- The third line uses the == operator to check to see whether the two values are equal. The two values are equal, so True is returned.
- The fourth line asks whether the two objects are the same. Since a is an integer while b is a real number, the two objects are not the same. In other words, they are stored in different memories, and therefore, False is returned.

 One More Step

- ▶ The in operator checks to see if strings partially match.

```
1 print('aaa' in 'aaa-bbb-ccc')
2 print('bbb' in 'aaa-bbb-ccc')
3 print('ddd' in 'aaa-bbb-ccc')
```

True

True

False

- ▶ If you want to determine whether there are any matching characters within a string, you can use the in operator to obtain True and False values.
- ▶ In the code above, since the string aaa is included in the string aaa-bbb-ccc, True is returned through the operation.



## One More Step

- ▶ Python can also use two or more comparison operators as shown below.
- ▶ In the code below, n is 100, so  $0 < n < 200$  is True. The if statement is a conditional statement that executes when the condition is True, but we will learn more about this later in the course.

```
1 n = 100
2 print('n =', n)
3 if 0 < n < 200 :
4     print('n is between 0 and 200.')
```

n = 100  
n is between 0 and 200.

## 1. Boolean Operation: Logic and Comparison Operators

### 1.7. Caution when using comparison operators



An error can occur when the order of operators is changed.

```
1 num1 = 100
2 num2 = 200
3 num1 =< num2
```

```
File "<ipython-input-34-efe2ac59bee3>", line 3
    num1 =< num2
      ^
```

SyntaxError: invalid syntax

- ▶ The `<=` operator should not be used as `=<`.
- ▶ There should not be any spaces between `!` and `=` in `!=`, nor should there be a space between `>` and `=` in `>=` operations.

## 2. Logical Operators: Determining Logical Expressions

### 2.1. Logical operators : and, or, not

Logical Operators determine logical expressions and return True or False as results.

$x \text{ and } y$

$x$	$y$	$x \text{ and } y$
False	False	False
False	True	False
True	False	False
True	True	True



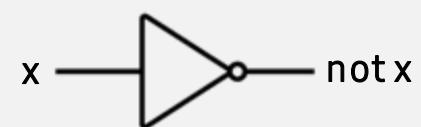
$x \text{ or } y$

$x$	$y$	$x \text{ or } y$
False	False	False
False	True	True
True	False	True
True	True	True



$\text{not } x$

$x$	$\text{not } x$
False	True
True	False



This figure above is a symbol for logical gates widely used in electric engineering and computers science. Each is a symbol that means and, or, and not, respectively.

## 2. Logical Operators: Determining Logical Expressions

### 2.1. Logical operators : and, or, not

Operator	Description
x and y	If any one of x or y is False, then it is False. It is True only when all of them are True.
x or y	If any one of x or y is True, then it is True. It is False only when all of them are False.
not x	If x is True, it is False. If x is False, it is True.



When logical operators are applied, the results are returned in Boolean types.

```
1 x = True
2 y = False
3 print(x and y)
4 print(x or y)
5 print(not x)
6 print(not y)
```

False

True

False

True

## 2. Logical Operators: Determining Logical Expressions

### 2.1. Logical operators : and, or, not

| Logical Operation combines Boolean values to create new Boolean values.

```
1 num = int(input('Enter an Integer: '))
2 result = ( num >= 0 and num <= 100 and num % 2 == 0 )
3 print('Is the input an even integer between 0 and 100?', result)
```

```
Enter an Integer: 99
Is the input an even integer between 0 and 100? False
```

#### Line 2

- num is greater than 0 but less than 100, so the first and second equations return True.
- In the third equation, num has a remainder of 1, so 1==0 is returned as False.
- The and logical operator returns True when all values are True; therefore, it returns False.

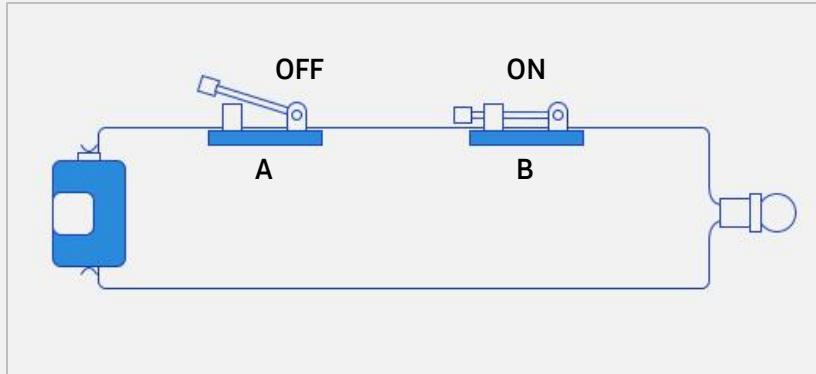
## 2. Logical Operators: Determining Logical Expressions

### 2.2. Comparison of 'and' to 'or'

#### I Logical Operators: and, or

When using the logical operator “and”, the output value is affected by a False input. It returns True if both statements are true.

If any of the switches is OFF, then the bulb is off (False).



and

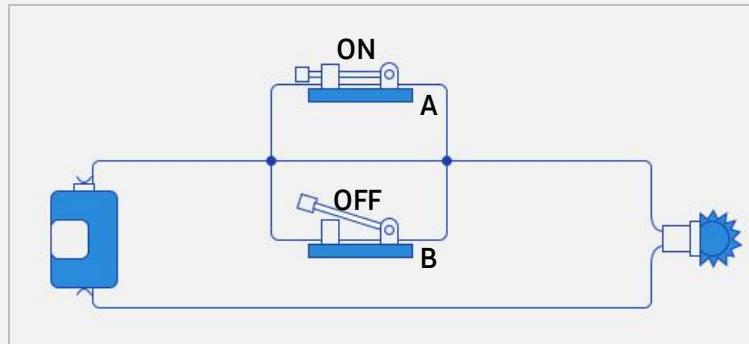
Input		Output
A	B	
True	True	True
True	False	False
False	True	False
False	False	False

## 2. Logical Operators: Determining Logical Expressions

### 2.2. Comparison of 'and' to 'or'

When using the logical operator “or”, the output value is affected by a True input. It returns True if one of the statements is true.

If any of the switches is ON, then the bulb is on (True)



		or
Input		Output
A	B	
True	True	True
True	False	True
False	True	True
False	False	False

# | Paper coding

**Try to fully understand the basic concept before moving on to the next step.**

**Lack of understanding basic concepts will increase your burden in learning this course, which may make you fail the course.**

**It may be difficult now, but for successful completion of this course we suggest you to fully understand the concept and move on to the next step.**

**Q1.**

Write code that receives the value n through the user's keyboard input.  
Return True if the given integer n is odd and return False if the integer is even.  
For cases where n is 20 and 21, print the following.

Conditions  
for Execution

Enter an Integer : 20  
Is the integer odd?: False  
Enter an Integer : 21  
Is the integer odd?: True

Time

5 Minutes



Write the entire code and the expected output results in the note.

**Q2.** Write code that takes the user's input and determines whether the integer value n is an even number within the range of 0 to 100 or not. The result of execution should look as follows:

Conditions for Execution	Enter an integer: 12 Is the input an even integer between 0 and 100? False
Time	5 Minutes



Write the entire code and the expected output results in the note.

## Unit 05. Logic and Comparison Operations

| Let's code

## 1. Logical Operators: AND, OR, NOT

### 1.1. AND, OR, NOT return true/false values

- | Python supports and, or, not operators, commonly known as logical operators. These operations return Boolean values of either True or False through logical operations and, or, and not.
- | Booleans are a data type that have True and False values. Other data types that are not originally Booleans can be converted into Booleans.
- | First, the number 0 is converted and returned as False in Boolean. All other numbers beside 0 are converted and returned as True. For example, if a is 0 and b is 1, then a and b is interpreted as False and True.
- | White spaces such as " " are converted and returned as False in Boolean while all other letters are returned as True.
- | None is a keyword that means there is no value. Therefore, it is converted and returned as False.

## 1. Logical Operators: AND, OR, NOT

### 1.1. AND, OR, NOT return true/false values

```
1 print(10 > 20) # 10 is less than 20, so 10 > 20 is False
2 print(10 < 20) # 10 is less than 20, so 10 < 20 is True
3 print(bool(9)) # 9 is not 0, so it is True
4 print(bool(-1)) # -1 too is not 0, so it is True
5 print(bool(0)) # 0 is the only number value that is False
6 print(bool(None)) # None means there is no value, so it is False
7 print(bool('')) # White space is a string, so it is False
8 print(bool('hello')) # It holds string values, so it is True
9
```

False

True

True

True

False

False

False

## 1. Logical Operators: AND, OR, NOT

### 1.2. bool function returns true/false values

- | From the result of the previous exercise, it can be observed that 9 and -1 are considered True, while 0 is considered False. It can also be observed that none of the white spaces are considered False.
- | Here, None is a Python keyword expressing that there is no value. Be sure to check that 0 and white spaces are considered False and that any other values other than 0 is True.

## 1. Logical Operators: AND, OR, NOT

### 1.3. logical operators return true/false values

An operation that can be applied to Boolean data types in such a way is a logical operation. The logical operation combines Boolean values to create new Boolean values.

#### Logical operators in Python

Operator	Description
x and y	If either one of x or y is False, then it is False. It is True only when all of them are True.
x or y	If any one of x or y is True, it is True. It is False only when all of them are False.
not x	If x is True, it is False. If x is False, it is True.

## 1. Logical Operators: AND, OR, NOT

### 1.4. Truth table

The values returned by the logical operators depending on the values of x and y are as shown in the table below.

x and y

x	y	x and y
False	False	False
False	True	False
True	False	False
True	True	True

x or y

x	y	x and y
False	False	False
False	True	False
True	False	True
True	True	True

not x

x	not x
False	True
True	False

## 2. Order of Operation

### 2.1. Precedence of Python Operators

- We use many operators when writing a program, and the order in which they are combined is important.

$$x + y \times z$$

A diagram illustrating operator precedence. On the left, the expression  $x + y \times z$  is shown. A horizontal bracket underneath the terms  $y \times z$  is labeled with a red circled ①, indicating that multiplication is performed before addition. A larger horizontal bracket underneath the entire expression  $x + y \times z$  is labeled with a red circled ②, indicating the final result of the addition.

$$(x + y) \times z$$

A diagram illustrating operator precedence. On the right, the expression  $(x + y) \times z$  is shown. A horizontal bracket underneath the terms  $x + y$  is labeled with a red circled ①, indicating that addition is performed first. A larger horizontal bracket underneath the entire expression  $(x + y) \times z$  is labeled with a red circled ②, indicating the final result of the multiplication.

- As we learned in math class, multiplication and division should be done before addition and subtraction. Determining its precedence is the order of operation.
- This is a rule that determines which of the many operations should be performed first. Each operator is sequenced. Multiplication and division have higher priorities than addition and subtraction.

## 2. Order of Operation

### 2.2. Operator precedence table

Operator	Description
**	Exponentiation Operator
~, +, -	Unary Operator
*, /, %, //	Multiplication, Division, Modulo Operator
+, -	Addition, Subtraction
>>, <<	Bitwise Operator
&	Bitwise AND Operator
^,	Bitwise XOR, Bitwise OR Operator
<=, <, >, >=	Comparison Operator
==, !=	Equality Operator
=, %=, /=, //=, -=, +=, *=, **=	Assignment Operator, Compound Assignment Operator
is, is not	Identity Operator
in, not in	Containment Operator
not, or, and	Logical Operator

- ▶ The higher the position of the operator on the table, the higher its precedence

## 2. Order of Operation

### 2.3. Code examples of operator precedence

```
1 x = int(input("Enter the First Equation: "))
2 y = int(input("Enter the Second Equation: "))
3 z = int(input("Enter the Third Equation: "))
4
5 avg = (x + y + z) / 3
6 print("Average =", avg)
```

```
Enter the First Equation: 10
Enter the Second Equation: 20
Enter the Third Equation: 30
Average = 20.0
```

#### Line 5

- If we code  $x + y + z / 3$ , then it will calculate  $z / 3$  first. Therefore, we need to add it in parentheses first, such as  $(x + y + z) / 3$
- Just like the formula used in math class, the values in parentheses have a higher precedence.

# | Pair programming



# Pair Programming Practice

## Guideline, mechanisms & contingency plan

Preparing pair programming involves establishing guidelines and mechanisms to help students pair properly and to keep them paired. For example, students should take turns “driving the mouse.” Effective preparation requires contingency plans in case one partner is absent or decides not to participate for one reason or another. In these cases, it is important to make it clear that the active student will not be punished because the pairing did not work well.

## Pairing similar, not necessarily equal, abilities as partners

Pair programming can be effective when students of similar, though not necessarily equal, abilities are paired as partners. Pairing mismatched students often can lead to unbalanced participation. Teachers must emphasize that pair programming is not a “divide-and-conquer” strategy, but rather a true collaborative effort in every endeavor for the entire project. Teachers should avoid pairing very weak students with very strong students.

## Motivate students by offering extra incentives

Offering extra incentives can help motivate students to pair, especially with advanced students. Some teachers have found it helpful to require students to pair for only one or two assignments.



# Pair Programming Practice

## | Prevent collaboration cheating

The challenge for the teacher is to find ways to assess individual outcomes, while leveraging the benefits of collaboration. How do you know whether a student learned or cheated? Experts recommend revisiting course design and assessment, as well as explicitly and concretely discussing with the students on behaviors that will be interpreted as cheating. Experts encourage teachers to make assignments meaningful to students and to explain the value of what students will learn by completing them.

## | Collaborative learning environment

A collaborative learning environment occurs anytime an instructor requires students to work together on learning activities. Collaborative learning environments can involve both formal and informal activities and may or may not include direct assessment. For example, pairs of students work on programming assignments; small groups of students discuss possible answers to a professor's question during lecture; and students work together outside of class to learn new concepts. Collaborative learning is distinct from projects where students "divide and conquer." When students divide the work, each is responsible for only part of the problem solving and there are very limited opportunities for working through problems with others. In collaborative environments, students are engaged in intellectual talk with each other.

**Q1.**

Receive a 3-digit integer from the user. If the hundredth digit of the integer n is 3, return True. If not, return False.

Example  
Result

Enter a 3-digit integer: 321  
True

- ▶ Hint: You must be familiar with the // operator. If not, please go back and review it.

**Q2.** Receive one integer. If the integer is a multiple of 5, return True. If not, return False.

Example  
Result

Enter an integer: 125  
True

Unit 6.

# **Conditional Statement-1**

## **: Conditions and Decision Making**

## Learning objectives

- ✓ Understand the necessity of conditional statements and utilize conditional statements.
- ✓ Understand and utilize the differences between control statements such as conditional statements and loop statements and sequential statements.
- ✓ Understand syntax of the if conditional statement and construct complex conditional statements.
- ✓ Understand syntax of blocks and indentations and construct overlapping blocks.

## Lesson overview

- ✓ Learn about control statements that operate according to the sequential statement and conditions.
- ✓ Using the if statement, write a program that runs differently according to the conditions
- ✓ Learn how to use conditional expressions that return Boolean types, which are True and False, in a conditional statement.

## Concepts You Will Need to Know From Previous Units

- ✓ Understand the notion of a variable's data type, define variables in desired types, and compute without error.
- ✓ Receive input from the user using an input function and print using the print function.
- ✓ Execute various computations using arithmetic, comparison and logical operators.

# Keywords

Sequential  
Statement

Conditional  
Statement

if  
Statement

Boolean  
Data Type

Conditional  
Expression

# Mission

## 1. Real world problem

### 1.2 Lottery simulation with 3 integers



- ▶ In this mission, we will write a program that notifies you of winning the lottery of a simple three-digit number.
- ▶ This mission has a lottery winning number of integers 2, 3, and 9, and the user who does not know these numbers enter three numbers.
- ▶ Here, the user's inputs can only be in single-digit natural number and are divided by blanks. If the user correctly matches the three numbers regardless of their order, such as in 2 3 9 or 9 3 2, print 'You won the lottery'.
- ▶ Through this mission, we will learn the principles of programs that operate under complex conditions.

## 2. Mission

### 2.1. How the lottery machine works

```
1 n1, n2, n3 = 2, 3, 9
2
3 n = 0
4 a, b, c = input('Enter a three-digit lottery number : ').split()
5 a = int(a)
6 b = int(b)
7 c = int(c)
8
9 if a == n1 or a == n2 or a == n3: # If one of the numbers match the winning number
10    n += 1                         #Increase n
11 if b == n1 or b == n2 or b == n3:
12    n += 1
13 if c == n1 or c == n2 or c == n3:
14    n += 1
15
16 if n == 3:                      # If n is 3, three numbers matched
17    print('You won the lottery!')
```

Enter a three-digit lottery number :  
You won the lottery!

## 2. Mission

### 2.2. Mechanism of automatic lottery machine

```
1 n1, n2, n3 = 2, 3, 9
2
3 n = 0
4 a, b, c = input('Enter a three-digit lottery number : ').split()
5 a = int(a)
6 b = int(b)
7 c = int(c)
8
9 if a == n1 or a == n2 or a == n3: # If one of the numbers match the winning number
10    n += 1                         #Increase n
11 if b == n1 or b == n2 or b == n3:
12    n += 1
13 if c == n1 or c == n2 or c == n3:
14    n += 1
15
16 if n == 3:                      # If n is 3, three numbers matched
17    print('You won the lottery!')
```

Enter a three-digit lottery number : 1, 2, 9

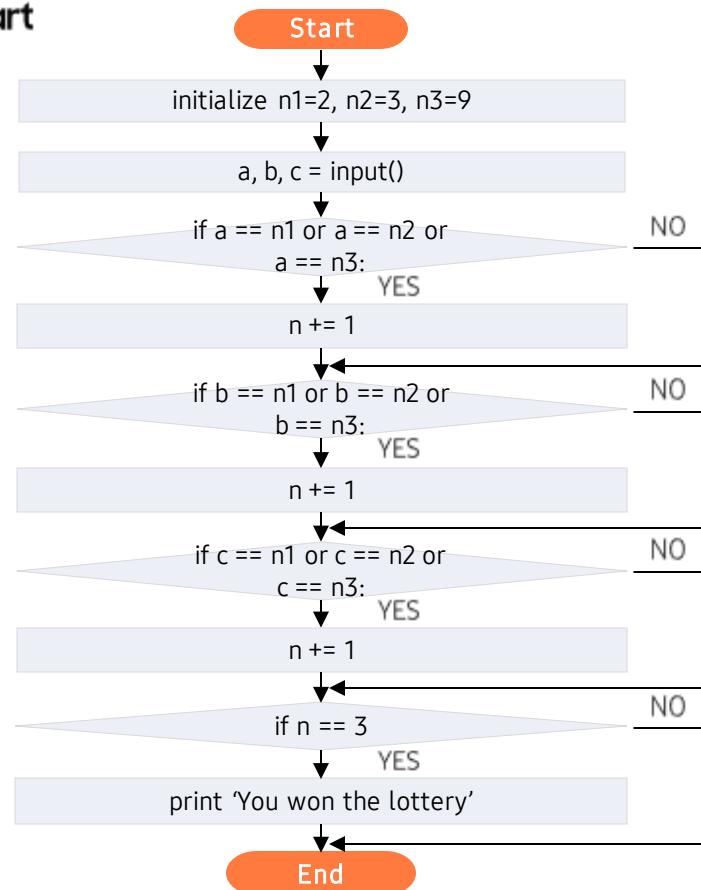
## 2. Mission

### 2.3. Programming Plan

#### Pseudocode

```
[1] Start  
[2] Generate three numbers, 2, 3, 9.  
[3] Reset with n = 0.  
[4] Receive input for three numbers a, b, c.  
[5] if a matches one of 2, 3, 9, then {  
    [6] Add 1 to n }  
[7] if b matches one of 2, 3, 9 then {  
    [8] Add 1 to n }  
[9] if c matches one of 2, 3, 9 then {  
    [10] Add 1 to n }  
[11] if n is 3, print 'You won the lottery!'  
[12] End
```

#### Flowchart



## 2. Mission

### 2.4. Automatic lottery machine final code

```
1 n1, n2, n3 = 2, 3, 9
2
3 n = 0
4 a, b, c = input('Enter a three-digit lottery number : ').split()
5 a = int(a)
6 b = int(b)
7 c = int(c)
8
9 if a == n1 or a == n2 or a == n3: # If one of the numbers match the winning number
10    n += 1                         #Increase n
11 if b == n1 or b == n2 or b == n3:
12    n += 1
13 if c == n1 or c == n2 or c == n3:
14    n += 1
15
16 if n == 3:                      # If n is 3, three numbers matched
17    print('You won the lottery!')
```

# | Key concept

## 1. if Conditional Statements

### 1.1 Conditional statements that have different flows depending on the conditions

- | The Python program we have written so far is of a sequential structure in which code written first are executed first.
- | The sequential execution structure is a general structure that most programming languages as well as Python use.
- | However, in programming languages, sequential flow is not always the case.
- | Certain commands may be performed only when certain conditions are satisfied, or different tasks may be required depending on the conditions.
- | This unit will learn conditional statements to solve more complex problems.

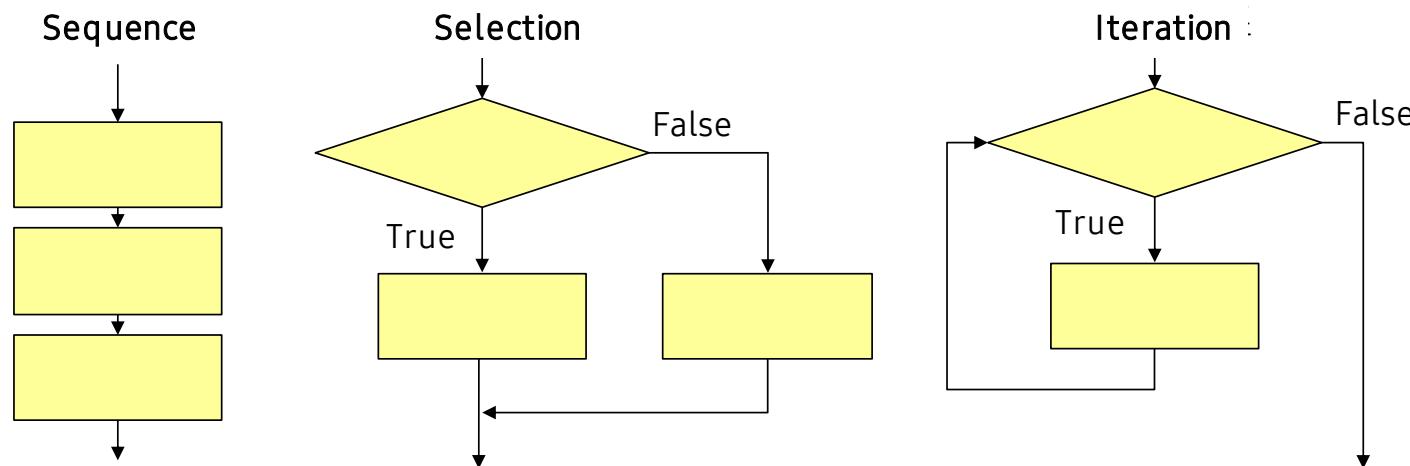
## 1. if Conditional Statements

### 1.2. Basic control structures of a program

| There are three main control structures of a program.

- ▶ Sequence - a structure in which commands are executed sequentially.
- ▶ Selection - a structure in which one of several instructions is selected and executed.
- ▶ Iteration - a structure in which the same command is executed repeatedly.

| The flowcharts representing the above three structures are shown below.



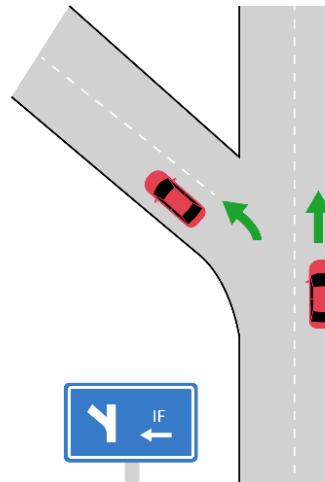
## 1. if Conditional Statements

### 1.2. Basic control structures

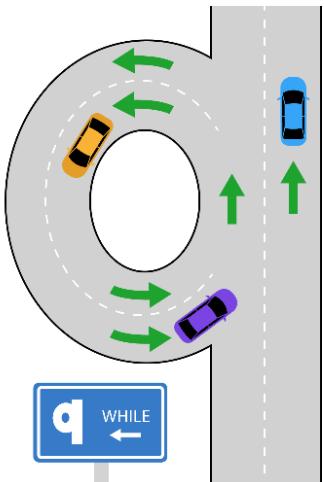
- | The basic control structure of a program resembles that of the basic Lego blocks. Almost all Lego works are made using only a few basic blocks.
- | The same goes for programs. Even a very complex program can be created with only three basic blocks.
- | To easily understand the basic blocks of the program, you can think of them as roads on which cars drive.
  - ▶ The Sequence can be considered as a road through which a car goes straight as shown in Figure 1.
  - ▶ The Selection is an intersection where a car selects one of the two roads and drives, as shown in Figure 2.
  - ▶ The iteration as shown in Figure 3, can be said to be a roundabout where the vehicle rotates and travels. This structure can iterate a statement multiple times.



<Figure 1>



<Figure 2>



<Figure 3>

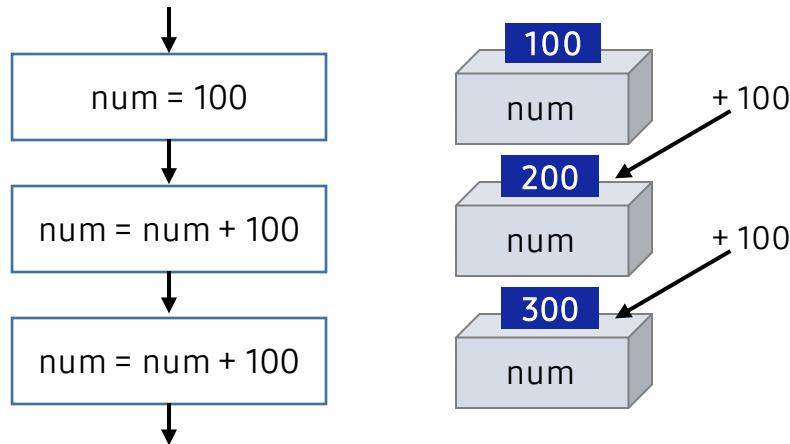
## 1. if Conditional Statements

### 1.3. Sequential statements

**I Sequential statements:** This is a structure in which the code written first is executed first. Each time `num = num + 100` is executed, `num` increases by 100.

```
1 num = 100
2 print('num = ', num)      # 100 is printed
3 num = num + 100
4 print('num = ', num)      # 100 is added to num and 200 is printed
5 num = num + 100
6 print('num = ', num)      # 100 is added again to num and 300 is printed
```

```
num = 100
num = 200
num = 300
```



A structure in which commands are executed sequentially

## 1. if Conditional Statements

### 1.4. Sequential statements and various flow statements

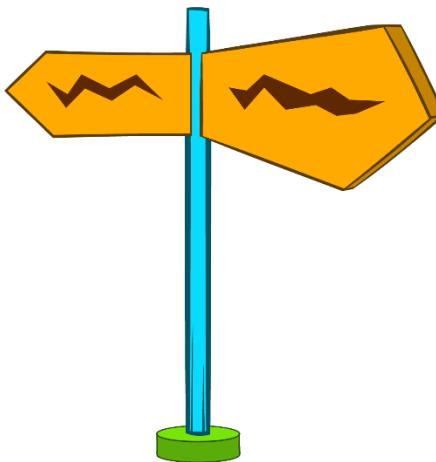
- | Let's look at flow statements that are not sequential statements.
- | **Control statements:** Statements that control the flow of a program.
- | There are three types of control sentences as follows.
  - ▶ **Conditional statement** : if statement, if-else statement, if-elif-else statement
  - ▶ **Loop statement** : for statement, while statement
  - ▶ **Commands that change the flow of a loop statement** : break, continue

## 1. if Conditional Statements

### 1.5. Why we need a selection structure

| Why do we need selection structure?

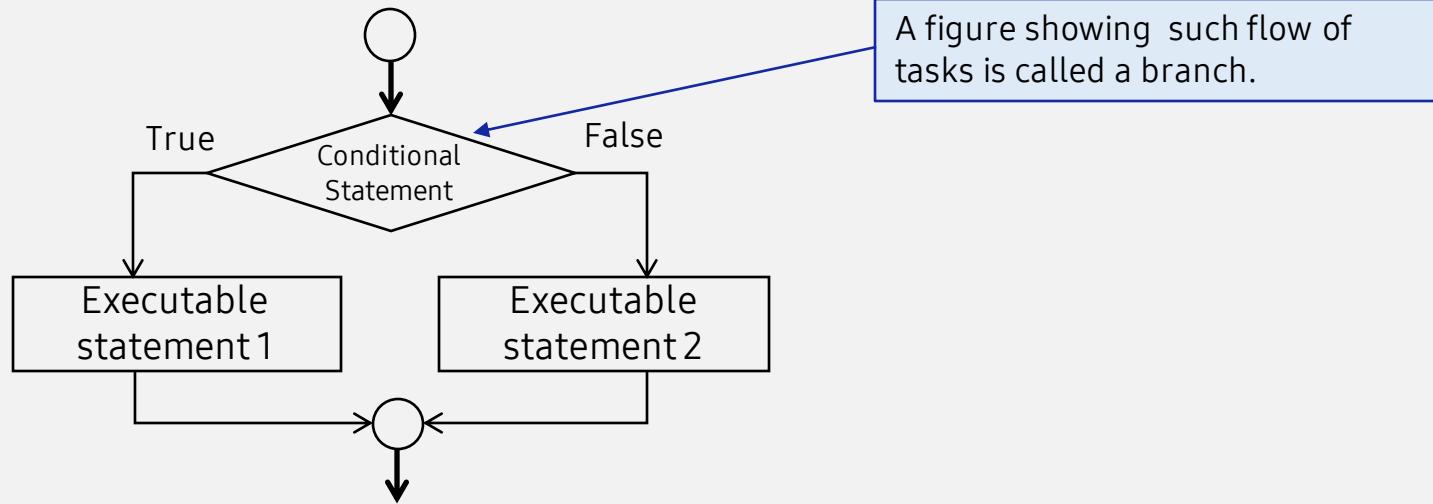
- ▶ At some stages of the program, there are more than one paths to proceed, among which we have to select one.
- ▶ If there is no selection structure, the program will always repeat the same action. If the program always does the same thing, it will always reach a fixed conclusion.



## 1. if Conditional Statements

### 1.6. Flow of conditional statement

How can we choose to run only one of several executable statements? Let's express this in a flowchart.



- <Executable Statement 1> or <Executable Statement 2> is executed according to specific conditions.
- Conditions are determined by <Conditional Statement>. The conditional statement must have a True or False value.

## 1. if Conditional Statements

### 1.7. If statement and related terms that are executed in specific circumstances

- | Study if statements and related terms that are executed in specific circumstances.
- | Create a code that operates according to the situations written below.

Situation 1: print "youth discount" if the age is under 20.

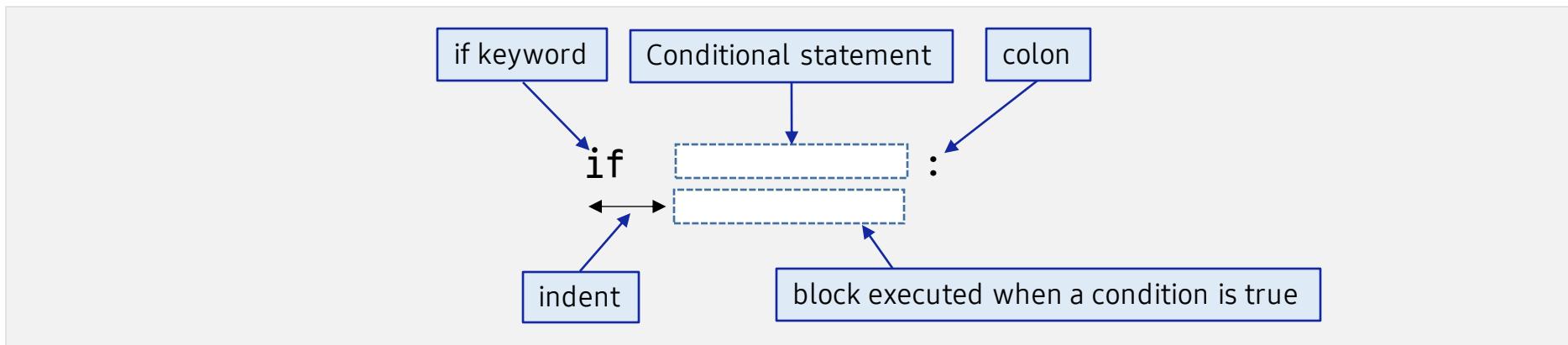
Situation 2: print "goal achieved" if step count over 1,000.

- ▶ Implementing this code requires an expression that determines whether the **condition** is satisfied or not.
- ▶ This is called a **conditional expression**. And this conditional expression is evaluated in Boolean type, which contains True or False values.
- ▶ The relational operator from the previous chapter is used to compare two operands, and the result of the **relational operator** is shown in True or False value. Thus, it can be used in conditional expressions.

## 1. if Conditional Statements

### 1.8. if statement syntax

| Study syntax of the if statement.



**Situation 1:**  
print "youth discount" if  
the age is under 20

↓  
if age < 20:  
    print('youth discount')

**Situation 2:**  
print "goal achieved" if  
stepcount(walk\_count) is  
over 1,000.

↓  
if walk\_count >= 1000:  
    print('goal achieved')

- ▶ (Situation 1) - In the conditional sentence that appears before the colon (:), print ('youth discount') is executed only when the age is less than 20 years old using < operator.
- ▶ (Situation 2) - In the conditional sentence, use the >= operator to execute a code, print ('goal achievement'), when the walk\_count reaches 1,000 or more.

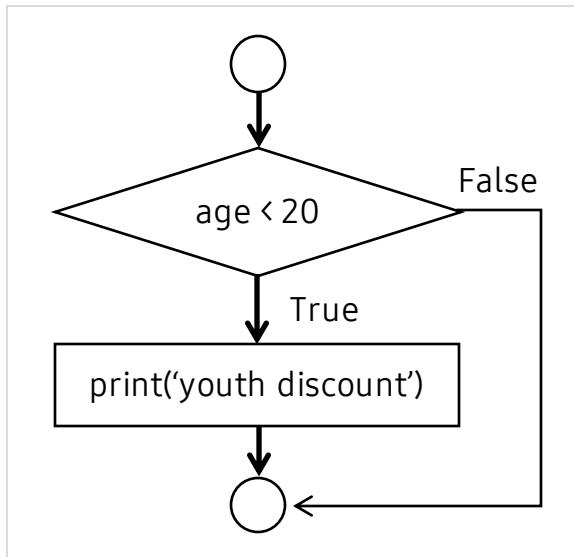
## 1. if Conditional Statements

### 1.9. if statement example code

| Write a code using an if statement.

```
1 age = 18          # if age is 18
2
3 if age < 20:      # result of age < 20 is True
4     print('youth discount')
```

youth discount

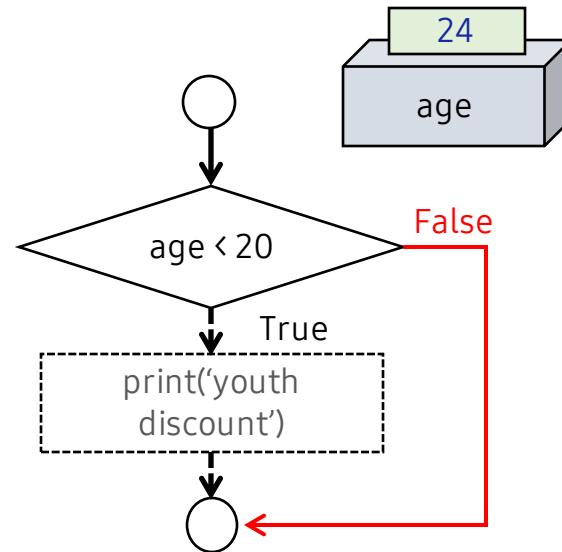
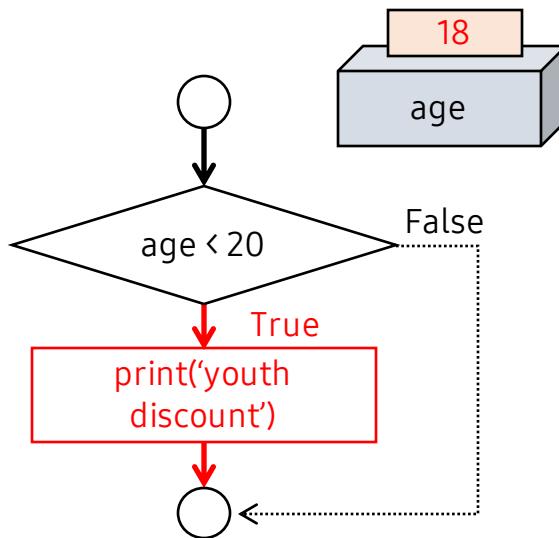


- ▶ When the age value is 18, the conditional statement of `age < 20` becomes True, so 'youth discount' is printed on the screen.
- ▶ When the age value is 20 or more, no output is printed.

# 1. if Conditional Statements

## 1.9. if statement example code

Example of converting to a different flow according to the value of the age variable



```
1 age = 18          # if age is 18
2
3 if age < 20:      # result of age < 20 is True
4     print('youth discount')
```

youth discount

```
1 age = 24          # if age is 24
2
3 if age < 20:      # result of age < 20 is False
4     print('youth discount')
```

## 1. if Conditional Statements

### 1.10. Block of if statement



#### Conditional statement and block

- ▶ Learn the necessity of indentation and the term block.
- ▶ The code below prints an error.

```
1 age = 18
2
3 if age < 20:
4     print('youth discount')
```

```
File "<ipython-input-11-897b539e10ef>", line 4
    print('youth discount')
^
```

**IndentationError:** expected an indented block

- ▶ A chunk of code that can be executed when a condition is true is called a block.
- ▶ The conditional block of a conditional statement must be indented. Otherwise, an error occurs as above.

## 1. if Conditional Statements

### 1.11. Indentation

 Focus Python is a programming language in which indentation is significant. Depending on the indentation, the same code produces different results. Let's take a closer look at the output from indented codes from 1 to 5.

- ▶ [Indentation code 1] If the conditions of age <20 is satisfied

```
1 age = 18
2 if age < 20:
3     print('youth discount')
4 print('Welcome')
```

youth discount  
Welcome

- ▶ [Indentation code 2] If the conditions of age <20 is not satisfied

```
1 age = 24
2 if age < 20:
3     print('youth discount')
4 print('Welcome')
```

Welcome

## 1. if Conditional Statements

### 1.11. Indentation

- ▶ [Indentation code 3] If the condition 'age <20' is satisfied

```
1 age = 18
2 if age < 20:
3     print('age', age)
4     print('youth welcome')
5     print('youth discount')
```

Age 18  
youth welcome  
youth discount

#### Line 3-5

- Since age is 18, age <20 is True and Line 3-5 are printed.

## 1. if Conditional Statements

### 1.11. Indentation

- ▶ [Indentation code 4] If the condition age <20 is not satisfied

```
1 age = 24
2 if age < 20:
3     print('age', age)
4     print('youth welcome')
5     print('youth discount')
```

No output

#### Line 3-5

- Since the age is 24, age <20 is False and Line 3-5 are not printed.



## One More Step

- ▶ Blocks are called code blocks.
  - It refers to a chunk of code that can be tied together with the source code.
  - Python must have an indented code block after an if statement, if the if statement is followed by a : (colon). else, elif, for, while, def, class, etc. must also be indented.
  - If the number of indentations is not identical within the same block, as in [indentation code 5], an indentation error called "IndentationError: unexpected indent" occurs.
  - By the Pythonic way, 4 spaces is the recommended size of indentation.



### Wrong Indentation

```
1 age = 18
2 if age < 20:
3     print('age', age)
4     print('youth welcome')
5     print('youth discount')

File "<tokenize>", line 4
    print ('youth welcome')
^
```

IndentationError: unindent does not match any outer indentation level

 One More Step

- ▶ The rules of block indent
  - The PEP-8 guideline recommends writing code in the line after the line of colon.
  - Most Python Integrated Development Environments (IDE) provide automatic indentation.
  - Also, a space key, rather than a tab, is recommended to indent. Blocks can be written in multiple lines.
  - For multiple lines of commands, the number of indentations must all be the same.

```
1 age = 18
2 if age < 20: print('youth discount') # Bad coding
```

youth discount

```
1 age = 18
2 if age < 20:
3     print('youth discount') # Good coding
```

youth discount

## 2. Applying Numeric and String Conditions to Conditional Statements

### 2.1. Code to verify if a number is a multiple of 3

| Write a code that receives input from the user and verifies if the number is a multiple of 3.xx

```
1 number = int(input('Enter an integer: '))      # Change input value to integer type
2 if (number % 3) == 0:
3     print(number, 'is a multiple of 3.')
```

Enter an integer : 6  
6 is a multiple of 3.



- If the number is divided by 3 and the remainder is all 0, line 3 is executed (If the value of module 3 is 0, it's a multiple of 3)
- In this way, it is possible to determine whether the number is a multiple of 3.

## 2. Applying Numeric and String Conditions to Conditional Statements

### 2.2. Code to verify if a number is a multiple of 3 and 5

| Verify if a number is a multiple of 3 and 5

```
1 number = int(input('Enter an integer: '))
2 if (number % 3) == 0 and (number % 5) == 0:
3     print(number, 'is a multiple of 3 and 5.')
```

Enter an integer : 15  
15 is a multiple of 3 and 5.



#### Line 2,5

- If the remainder is zero when the number is divided by 3, and the remainder is zero when divided by 5, line 3 is executed.
- In this way, it is possible to determine whether the number is a multiple of 3 as well as 5.

## 2. Applying Numeric and String Conditions to Conditional Statements

### 2.3. == operator

A code that verifies whether the input string matches my\_id is shown below. That is, we can verify whether the stored string matches the input string using ==.

```
1 my_id = 'david'  
2  
3 s = input('Enter your id: ')  
4 if s == my_id:  
5     print('ID matches.')
```

Enter your id: david  
ID matches.

#### Line 2,5

- If the string input is 'david', the two strings s and my\_id match, and Line 5 is executed. When another string is given, Line 5 is not executed.

### 3. Various scenarios of conditions

#### 3.1. Even-number discrimination scenario

| Below is an even-number discrimination scenario.

```
1 n = int(input("Enter an integer : "))
2 print("n =", n)
3 if n % 2 == 0 :
4     print(n , "is an even number.")
```

Enter an integer : 50

n = 50

50 is an even number.

##### Line 3

- Checks if the remainder is 0 when the input n is divided by 2. If the input value is 50, the equation is True, so line 4 is executed.

### 3. Various scenarios of conditions

#### 3.2. Scenario verifying if two strings match

| Check if the string matches.

```
1 str1 = 'aaa'  
2 str2 = 'bbb'  
3 if str1 == str2:  
4     print('Two strings match.')  
5 if str1 != str2:  
6     print('Two strings are different.')
```

Two strings are different.



Line 3

- Runs Line 4 block if the two strings are identical. In this case, it is not printed since the values of str1 and str2 are different.

### 3. Various scenarios of conditions

#### 3.2. Scenario verifying if two strings match

| Check if the string matches.

```
1 str1 = 'aaa'  
2 str2 = 'bbb'  
3 if str1 == str2:  
4     print('Two strings match.')  
5 if str1 != str2:  
6     print('Two strings are different.')
```

Two strings are different.



Line 5

- If the values of the two strings are different, Line 6 block is executed. In this case, it is printed because the values of str1 and str2 are different.

## 4. pass Keyword

### 4.1. Role of pass statement

- Python's pass statement is used to omit code in a block.
- It is mainly used when you wish to write code later because the exact execution of the conditional statement has not been determined yet.

```
1 num = 2
2 if num % 2 == 0:
3     print('Even number')
4 if num % 3 == 0:
5     pass
```

Even number



#### Line 5

- It leaves a block empty to complete the functionality later.

## 4. pass Keyword

### 4.1. Role of pass statement

- | Python's pass statement can be used to define the name and implement the functionalities later. We will learn about the functions later.
- | The function func below defines only its name and intends to implement the function later.

```
1 def func():  
2     pass  
3  
4 func()
```

#### Line 4

- func function does not do anything because there is no code inside the func() function.

## 4. pass Keyword

### 4.2. pass statement's role

! pass statement plays an important role.

```
1 def func():
2
3
4 func()
```

```
File "<ipython-input-3-7c72fccf7a42>", line 4
    func()
    ^
```

**IndentationError:** expected an indented block

- ▶ Although it does not have a functionality, the pass statement plays a role as a placeholder.
- ▶ Therefore, if the pass statement is omitted inside the function in the above sentence, an error occurs.



## One More Step

### ▶ pass statement and function

- The print, input, etc. that we use are bundles of codes that perform defined actions.
- This function performs a specific action whenever it is called.
- Sometimes we have to construct the function ourselves.(You'll learn this later)
- You can simply create a name and test it without a code, and even in this case, write the pass keyword to specify that the function has not yet been constructed.

```
def func_name(x1, x2, ... ) :  
    code1  
    code2  
    ...  
    return n1[, n2,...]
```

The general form of a function

```
def func_name(x1, x2, ... ) :  
    pass
```

A function with no body constructed

# | Paper coding

**Try to fully understand the basic concept before moving on to the next step.**

**Lack of understanding basic concepts will increase your burden in learning this course, which may make you fail the course.**

**It may be difficult now, but for successful completion of this course we suggest you to fully understand the concept and move on to the next step.**

## Q1.

If a game user's game\_score is over 1000 points, print 'You are a master.'

Example Output

```
Enter game score : 1500  
game_score = 1500  
You are a master.
```

or

```
Enter game score : 100  
game_score = 100
```

Time

5min



Write the entire code and the expected output results in the note.

**Q2.** Write a program that receives any integer  $x$  between -100 and 100 and 1) prints  $x$  on the screen, and 2) prints "...is a natural number" if  $x$  is an integer greater than zero. Otherwise, let it simply print  $x$  as in  $x = -10$ .

## Example Output

```
Enter integer:  
x = 50  
50 is a natural number.
```

or

```
Enter integer:  
x = -10
```

## Time

5min



Write the entire code and the expected output results in the note.

| Let's code

## 1. If Statement

### 1.1. If statements that are executed when certain conditions are satisfied

| Below is an if statement that is executed when a certain condition is satisfied.

```
1 x = 100
2 if x > 1:
3     print("x is greater than 1.")
```

x is greater than 1.

- ▶ Check the function of the if statement with an actual code. Put 100 in the variable x and write the if statement as above. In this case, since the conditional expression  $x>1$  returns True, the following execution is run.
- ▶ Therefore, 'x is greater than 1' is printed on the screen. Here, the line below :(colon) must be indented.

## 1. If Statement

### 1.2. Relational operator =, ==



**Focus** Assignment operator = and relational operator == are different operators.

- ▶ This part is mentioned repeatedly, because beginners often make these mistakes. The = operator is an operation that puts the value of right to the variable on the left. The == operator returns True or False depending on whether the left and right values are identical or not.

## 1. If Statement

### 1.3. Indentation in Python

Indentation is an important element of coding.

- ▶ How can we print that one is accepted and qualified for a scholarship if the score is 90 or higher? In this case, sentences can be grouped by using indentations as shown below.
- ▶ If the value of score is 90 or higher, two sentences calling print() are executed in the code below. Note that these sentences have the same number of spaces. All of these belong to the same block.

```
if score >= 90 :
```

```
    print("You are accepted.")  
    print("You are qualified for a scholarship.")
```

Block: a group of sentences.



## 1. If Statement

### 1.3. Indentation in Python

- Indentation is a very important factor in Python. Blocks are completed by indentation.
- All sentences belonging to one block are executed together. The sentences in the block have four indentation spaces in front of them when compared to the sentences above them. These spaces are denoted by - as shown below.

```
1 score = int(input("Enter the score: "))
2 if score >= 90 :           # if the condition is true, block indent is all executed
3     print("Congratulations.")
4     print("You are accepted.")
5     print("You are qualified for a scholarship.")
```

Enter the score: 95  
Congratulations.  
You are accepted.  
You are qualified for a scholarship.

#### Line 2-5

- Line 3, 4 and 5 are executed because the score is given as 95.

## 1. If Statement

### 1.3. Indentation in Python

In Python, if there is the same number of spaces in front of the sentence, these sentences belong to one block. If the number of spaces is changed by mistake, an error occurs.

#### IndentationError

```
1 score = int(input("Enter the score: "))
2 if score >= 90 :      # Caution : if indentation is different an error occurs
3     -print("Congratulations.")
4     --print("You are accepted.")
5     ---print("You are qualified for a scholarship.")
```

```
File "<ipython-input-16-0e653e9e7f6e>", line 4
    print("You are accepted.
          ^
IndentationError: unexpected indent
```

#### Line 3-5

- Error occurs because the numbers of indented spaces are different.

## 2. Use of String and Spilt Method

### 2.1. split method

- The split method separates string objects. A string list is generated as a result.

```
1 words = 'Python is beautiful'  
2 word_list = words.split()  
3 print(word_list)
```

```
['Python', 'is', 'beautiful']
```

- The split uses space as the default value. So, each value is inserted using the space as a separator.
- In this way, a character that divides a string into one or more individual strings is called a [separator](#).

## 2. Use of String and Spilt Method

### 2.2. input function and split method

- The input function receives only strings as input. The split method cuts the input string in space units. To divide by commas, use split(',') .
- In order to perform numerical operations on this, it is necessary to convert it into an integer type or a real number type using int and float functions.

```
1 str1 = input('Enter three integers divided by space: ')
2 print('Input: ', str1)
3 print('Input separated by split() method: ', str1.split())
```

Enter three integers separated by space: **10 20 30** separated by spaces

Input: 10 20 30

Input separated by split() method: ['10', '20', '30']

```
1 str1 = input('Enter three integers separated by ,: ')
2 print('Input: ', str1)
3 print('Input separated by split() method: ', str1.split(','))
```

Enter three integers separated by ,: **10 20 30** separated by commas

Input: 10,20,30

Input separated by split() method: ['10', '20', '30']

- ▶ Make a list of string using string received by split function.
- ▶ After that, the elements in this list can be converted to an integer type.

## 2. Use of String and Spilt Method

### 2.3. How to process input values with split and int functions

Numbers separated by split methods are strings in the list. Therefore, numerical operations are possible only when they are converted into integer types using int function.

```
1 num1, num2, num3 = input('Enter three integers separated by ,: ').split(',')
2 num1, num2, num3 = int(num1), int(num2), int(num3)
3 print(num1, num2, num3)
```

Enter three integers separated by ,:



Enter three integers separated by ,: 5.6.7

```
1 num1, num2, num3 = input('Enter three integers separated by ,: ').split(',')
2 num1, num2, num3 = int(num1), int(num2), int(num3)
3 print(num1, num2, num3)
```

Enter three integers separated by ,: 5,6,7  
5 6 7

# | Pair programming



# Pair Programming Practice

## Guideline, mechanisms & contingency plan

Preparing pair programming involves establishing guidelines and mechanisms to help students pair properly and to keep them paired. For example, students should take turns “driving the mouse.” Effective preparation requires contingency plans in case one partner is absent or decides not to participate for one reason or another. In these cases, it is important to make it clear that the active student will not be punished because the pairing did not work well.

## Pairing similar, not necessarily equal, abilities as partners

Pair programming can be effective when students of similar, though not necessarily equal, abilities are paired as partners. Pairing mismatched students often can lead to unbalanced participation. Teachers must emphasize that pair programming is not a “divide-and-conquer” strategy, but rather a true collaborative effort in every endeavor for the entire project. Teachers should avoid pairing very weak students with very strong students.

## Motivate students by offering extra incentives

Offering extra incentives can help motivate students to pair, especially with advanced students. Some teachers have found it helpful to require students to pair for only one or two assignments.



# Pair Programming Practice

## | Prevent collaboration cheating

The challenge for the teacher is to find ways to assess individual outcomes, while leveraging the benefits of collaboration. How do you know whether a student learned or cheated? Experts recommend revisiting course design and assessment, as well as explicitly and concretely discussing with the students on behaviors that will be interpreted as cheating. Experts encourage teachers to make assignments meaningful to students and to explain the value of what students will learn by completing them.

## | Collaborative learning environment

A collaborative learning environment occurs anytime an instructor requires students to work together on learning activities. Collaborative learning environments can involve both formal and informal activities and may or may not include direct assessment. For example, pairs of students work on programming assignments; small groups of students discuss possible answers to a professor's question during lecture; and students work together outside of class to learn new concepts. Collaborative learning is distinct from projects where students "divide and conquer." When students divide the work, each is responsible for only part of the problem solving and there are very limited opportunities for working through problems with others. In collaborative environments, students are engaged in intellectual talk with each other.

## Q1.

Write a program that receives age as input and prints "Adult" if age is 20 or over, "Youth" if under 20 and equal to or over 10, and "Kid" if under 10.

### Output example

```
Enter age: 16  
Youth
```

```
Enter age: 33  
Adult
```

```
Enter age: 5  
Kid
```

**Q2.** Write a program that prints "Adult" if the age is equal to or over 20, "Youth" if under 20 and equal to or over 10 and "Kid" if under 10.

## Output example

```
Enter age: 20
Enter height in cm: 180
You can enter.
```

Unit 7.

# **Conditional Statement-2**

## **: Making decisions in two directions and applying conditional statements**

## Learning objectives

- ✓ Understand various conditional statements such as if statement, if-else statement, and if-else-if and code according to the situation.
- ✓ Solve more complex problems using double if-else statement.
- ✓ Solve and apply various problems through if-else statement and logic, comparison operators.
- ✓ Generate arbitrary random numbers using random modules.

## Lesson overview

- ✓ Use appropriate conditional expressions for if, elif statements.
- ✓ Check elif conditional expression when the conditional expression of if statement is not true.
- ✓ Understand the execution order of if-elif-else and apply them to solve problems.
- ✓ Understand that conditional expressions have values in Boolean data types, which are True and False , and solve the problems with various operators.

## Concepts You Will Need to Know From Previous Units

- ✓ Understand control statements executed according to conditions other than sequential execution.
- ✓ Understand that conditional statement is a statement executed only when a specific condition is satisfied.
- ✓ Understand the execution procedure of Boolean operators.

# Keywords

**if - else**

**if-else-elif**

**Double Conditional  
Statement**

**Random  
Numbers**

# Mission

## 1. Real world problem

### 1.3. Penalty shootout rules



- ▶ Penalty shootouts is a rule used in soccer when the score is tied even after overtime. In some cases, it is called PK, an abbreviation for penalty kicks, but it the exact term is penalty shoot-out. Therefore, it is accurate to mark it as PS or PSO.
- ▶ We will simulate such games using computers.
- ▶ Suppose a user tries a penalty kick against the computer. In this case, it is assumed that the user may select one of the three areas, left, center, and right, and perform a penalty kick.
- ▶ The computer will be a goalkeeper and defend, creating random numbers to defend one of the three areas.

## 2. Mission

### 2.1. Process of penalty shootout game

```
1 import random
2
3 n = random.randint(1, 3) # randomly generates numbers among 1,2,3
4 if n == 1:
5     computer_choice = "Left"
6 elif n == 2:
7     computer_choice = "Middle"
8 else:
9     computer_choice = "Right"
10
11 user_choice = input("Which side will you attack?(left, middle, right) : ")
12 if computer_choice == user_choice:
13     print("Offense failed.")
14 else :
15     print("Congrats!! Offense succeeded.")
16 print('computer defense position :', computer_choice)
```

Which side will you attack?(left, middle, right) : right

Congrats! You made it.

computer defense position : left

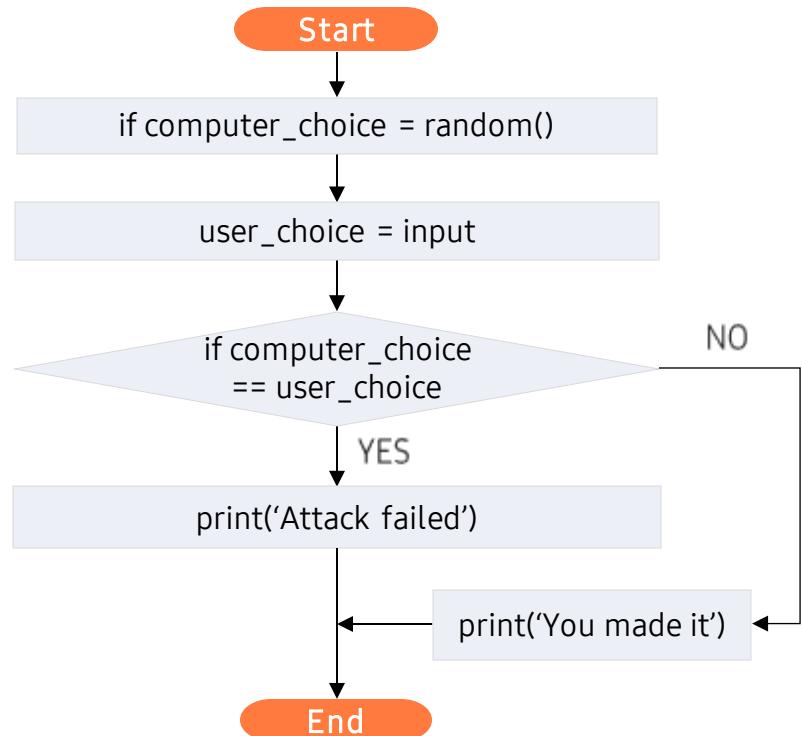
## 2. Mission

### 2.2. Programming plan

#### Pseudocode

- [1] Start
- [2] Receive random input.
- [3] User enters one of the three inputs.
- [4] Check if the direction of the keeper and the direction of the ball are identical.
- [5] if identical, print 'Attack failed'
- [6] if different, print 'Offense succeeded.'
- [7] End

#### Flowchart



## 2. Mission

### 2.3. Penalty shootout game final code

```
1 import random
2
3 n = random.randint(1, 3) # randomly generates numbers among 1,2,3
4 if n == 1:
5     computer_choice = "Left"
6 elif n == 2:
7     computer_choice = "Middle"
8 else:
9     computer_choice = "Right"
10
11 user_choice = input("Which side will you attack?(left, middle, right) : ")
12 if computer_choice == user_choice:
13     print("Offense failed.")
14 else :
15     print("Congrats!! Offense succeeded.")
16 print('computer defense position :, computer_choice)
```

# | Key concept

## 1. Two-way Decision-Making Using else, and elif Statement

### 1.1. else statement which is executed when the condition is false

| Python provides if-else statements for selection structures. Suppose that you get accepted and receives a scholarship with a grade equal to or higher than 90. On the other hand, not accepted if below 90.

- ▶ Code as follows in Python.

```
1 if score > 90:  
2     print('Accepted')  
3     print('Also receive a scholarship')  
4 else:  
5     print('Not accepted.')
```

- | The if-else sentence is saying "if the condition is true, execute this statement, and if the condition is not true, execute the other statement."
- | This unit will deal with more complex conditional statements as such.

## 1. Two-way Decision-Making Using `else`, and `elif` Statement

### 1.2. `if` statement without `else` statement

| This is a new situation. Consider a program that deals with the situation below which is dealt by the if statement from the previous chapter.

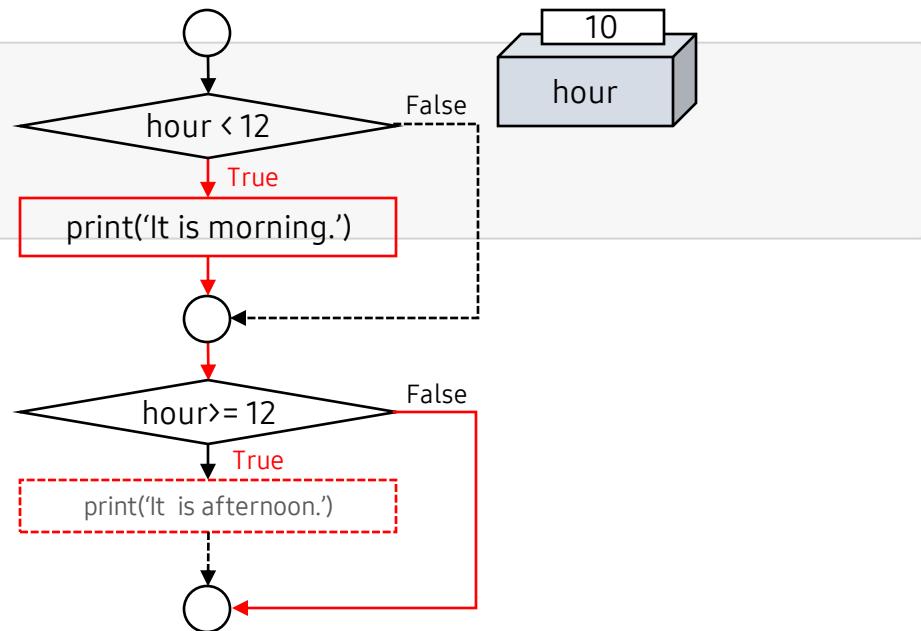
A function to print "It's morning" before 12pm, and "It's afternoon" after 12pm.

| If only the if statement is used, it will be possible to construct the code by using two ifs as follows.

| The figure on the right is a flowchart of this code.

```
1 hour = 10
2 if hour < 12:
3     print('It is morning.')
4 if hour >= 12:
5     print('It is afternoon.)
```

It is morning.



# 1. Two-way Decision-Making Using else, and elif Statement

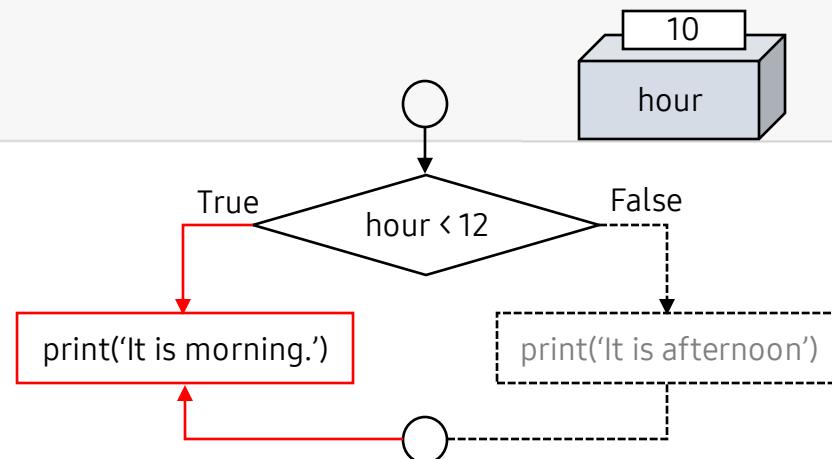
## 1.3. if - else statement

The previous code using if statements can be written as follows with an if-else statement.

- ▶ It can't be morning and afternoon at the same time. This relationship is called an exclusive relationship.
- ▶ The if-else statement is a command statement that can be used such exclusive relationships.
- ▶ Expressing an exclusive relationship with an if-else simplifies the flow chart as well.

```
1 hour = 10  
2  
3 if hour < 12:  
4     print('It is morning.')  
5 else:  
6     print('It is afternoon.')
```

It is morning.

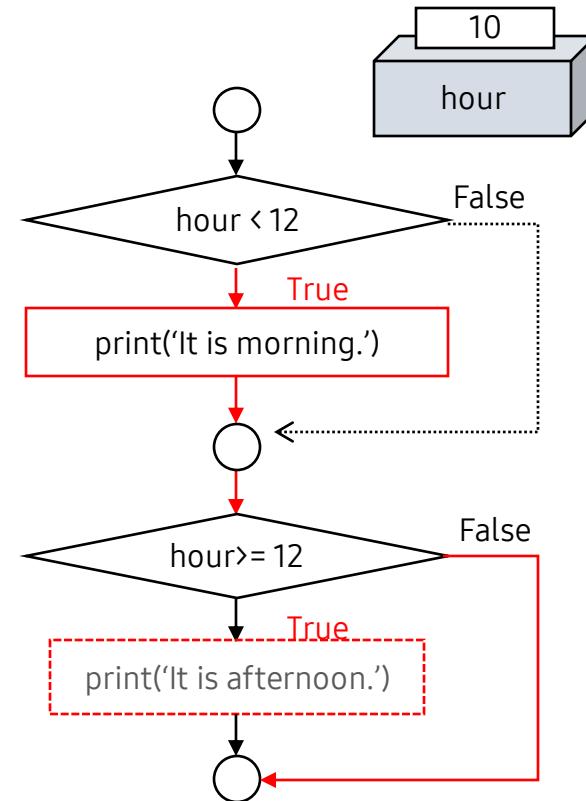
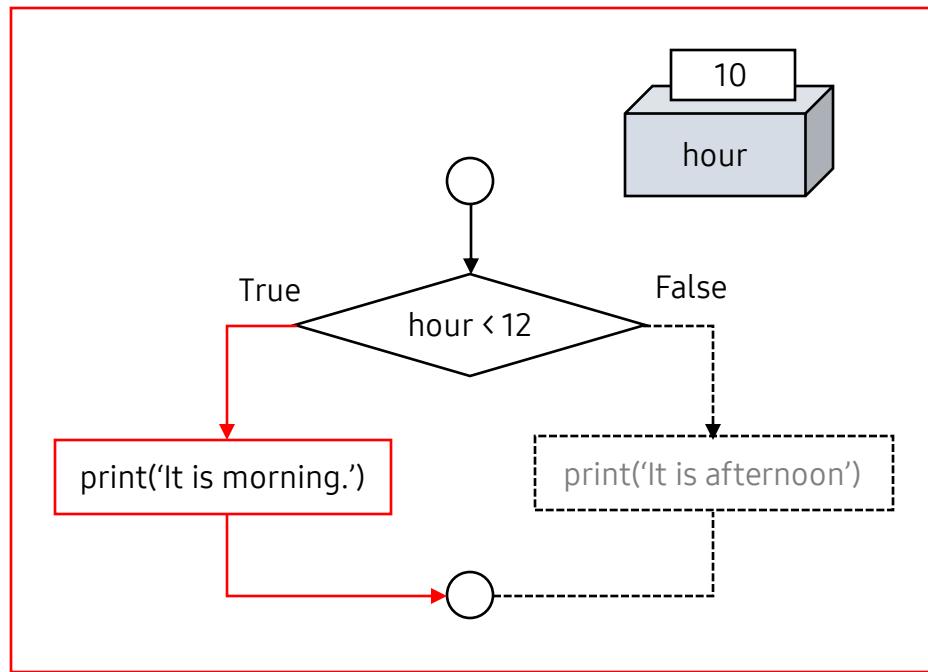


## 1. Two-way Decision-Making Using else, and elif Statement

### 1.4. Comparing flowchart of if-else and if

Compare flowcharts of if-else and if used twice.

- The left is much clearer.



## 1. Two-way Decision-Making Using else, and elif Statement

### 1.5. if-else statement to determine odd/even numbers

| Determine odd/even numbers with if-else statements. (In the case of an exclusive relationship, you can use the if-else statement as follows.)

```
1 num = 10
2 if num % 2 == 0:
3     print(num, 'is an even number.')
4 else:
5     print(num, 'is an odd number.')
```

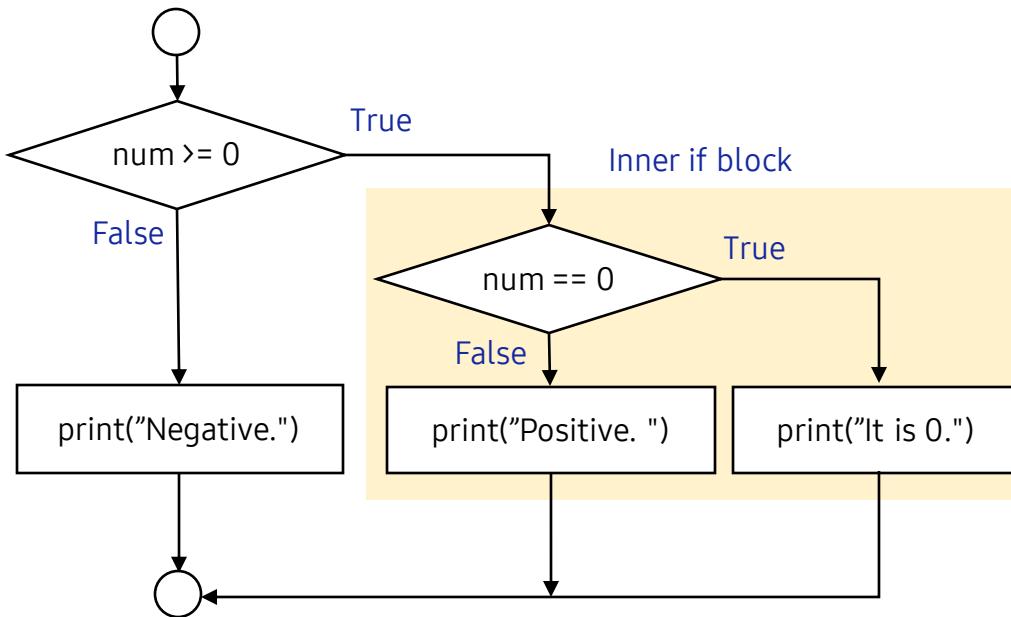
10 is an even number

 Line 2,4

- If num is divided by 2 and the remainder is 0, the lower Line 3 block is executed if the remainder is 0.
- Otherwise, Line 5 block under else is executed.

## 1. Two-way Decision-Making Using else, and elif Statement

### 1.6. Writing if-else statement inside if-else statement



- ▶ If necessary, another if statement may be included in the if statement. This is called a double if statement.
- ▶ A double if statement can occur in a situation where an additional specific condition needs verification after True is returned for the first condition.
- ▶ In this example, after receiving num from the user, if this value is `num >= 0`, another condition verification is performed to see if num is 0 or not, and prints "It is 0 " or "Positive".

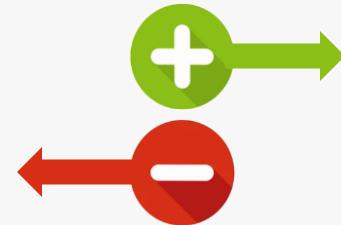
## 1. Two-way Decision-Making Using else, and elif Statement

### 1.6. Writing if-else statement inside if-else statement

| Consider a complicated case of inserting an if-else statement inside an if-else statement.

- ▶ We need to be able to resolve various situations with coding.
- ▶ Suppose a case in which a user receives an integer, determines whether it is positive, zero, or negative and prints the result on a screen.

```
1 num = int(input("Enter an integer: "))
2 if num >= 0:      # make sure to indent the block
3     if num == 0:    # check specific conditions inside the block
4         print("It is 0.")
5     else:
6         print("It is positive.")
7 else:
8     print("It is negative.)
```



Enter an integer: -20

It is negative.

- ▶ As such, you can solve problems by inserting an if-else statement inside an if-else statement.

## 1. Two-way Decision-Making Using else, and elif Statement

### 1.7. Using nested if - else statement to discriminate even/odd numbers

| Let's use another if-else within a block composed of if-else statements. In this code, even and odd numbers are determined and prints only in the case of positive numbers.

```
1 num = 100
2 if num < 0:
3     print(num, 'is negative.')
4 else:
5     print(num, 'is not negative.')
6     if num % 2 == 0:
7         print(num, 'is an even number.')
8     else:
9         print(num, 'is an odd number.)
```

'100 is not negative.  
'100 is even number.

 Line 2, 4

- Since num is greater than 0, the block right below is not executed, but the else statement of Line 4 is executed.

## 1. Two-way Decision-Making Using else, and elif Statement

### 1.7. Using double if - else statement to discriminate even/odd numbers

```
1 num = 100
2 if num < 0:
3     print(num, 'is negative.')
4 else:
5     print(num, 'is not negative.')
6     if num % 2 == 0:
7         print(num, 'is an even number.')
8     else:
9         print(num, 'is an odd number.)
```

100 is not negative.

100 is an even number.

#### Line 5~7

- After printing 'It is not negative.' using the print function, divide num by 2 once more to compare whether the remainder value is 0.
- Since  $100 \% 2$  is 0, the if statement is executed and the program is terminated.

## 1. Two-way Decision-Making Using else, and elif Statement

### 1.8. Flow control in various cases

| Below is a case where num=-100

```
1 num = -100
2 if num < 0:
3     print(num, 'is negative.')
4 else:
5     print(num, 'is not negative.')
6     if num % 2 == 0:
7         print(num, 'is an even number.')
8     else:
9         print(num, 'is an odd number.)
```

-100 is negative.

- ▶ In this code, if-else is used twice. The if-else statement on the outside is called the outer if-else statement, and the if-else statement on the inside is called the inner if-else statement. Closely examine the code.
- ▶ The if-else conditional statement of Line 2 and 4 is an outer if-else statement. The if statement of Line 2 is executed only when the value of num is less than 0, i.e., negative.

## 1. Two-way Decision-Making Using `else`, and `elif` Statement

### 1.8. Flow control in various cases

| Below is a case where num=-100

```
1 num = -100
2 if num < 0:
3     print(num, 'is negative.')
4 else:
5     print(num, 'is not negative.')
6     if num % 2 == 0:
7         print(num, 'is an even number.')
8     else:
9         print(num, 'is an odd number.)
```

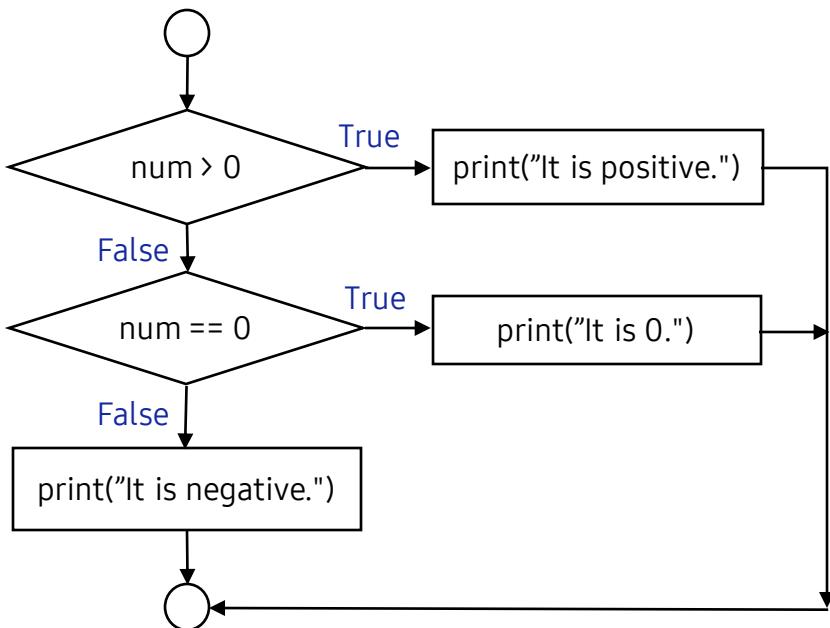
-100 is negative.

- ▶ The `else` statement of Line 4 is executed only when the value of `num` is not negative.
- ▶ Line 6 ~ 9 are an `if-else` statement inside the `else` statement and is called the inner `if-else` statement.
- ▶ The `if` statement of Line 6 is executed only when the variable `num` has no remainder after division by 2, that is, if it is an even number.
- ▶ Let's look at Line 8. This `else` block is executed when it is odd.
- ▶ Therefore, if the value of the variable `num` is changed to -100, it is executed as above.

## 2. Nested Conditional Statement

### 2.1. How to continuously examine other conditions when the conditions are false

| How to continuously examine other conditions when the conditions are false



- ▶ Consider a program that receives an integer num from the user and prints "It is positive," "It is 0," and "It is negative."
- ▶ If True is returned from the first if statement, the code line is executed and exits the conditional statement.
- ▶ If not (False), a decision may have to be made according to the second conditional expression.
- ▶ Programs with such a structure may be constructed using **if-elif-else**.
- ▶ elif is an abbreviation of "else if".

## 2. Nested Conditional Statement

### 2.2. Code example that prints positive, negative, and 0 after receiving integer as an input

| Below is a code that receives integers using an if-elif statement and prints positive, zero, and negative numbers.

```
1 num = int(input("Enter an integer: "))
2
3 if num > 0:
4     print("It is positive.")
5 elif num == 0:
6     print("It is 0.")
7 else:
8     print("It is negative.")
```

Enter an integer: 6

It is positive.

Enter an integer: -5

It is negative.

Enter an integer: 0

It is 0.

## 2. Nested Conditional Statement

### 2.3. Reason to use nested conditional statement

- | Assume a grade generator problem which classifies according to range of scores using multiple if statements. How can you print out grades according to score range as shown in the table below?
- | This problem can be solved by applying multiple if statements and conditions inside if statements.

Score	Grade
90 ~ 100	A
80 to Less than 90	B
70 to Less than 80	C
60 to Less than 70	D
Below 60	F

## 2. Nested Conditional Statement

### 2.4. Grade Generator example A: Code using multiple if statements

```
1 score = int(input('Enter score : '))
2 if score >= 90:           # If 90 or above, 'A'
3     grade = 'A'
4 if score < 90 and score >= 80: # If under 90, 80 or above, 'B'
5     grade = 'B'
6 if score < 80 and score >= 70: # If under 80, 70 or above, 'C'
7     grade = 'C'
8 if score < 70 and score >= 60: # If under 70, 60 or above, 'D'
9     grade = 'D'
10 if score < 60:            #If under 60, 'F'
11    grade = 'F'
12 print('Your grade is: ', grade)
```

Enter score : 88  
Your grade is: B

- ▶ Grade B for 88 points is properly printed.
- ▶ However, this code has become more complex and difficult to read than the previous if statement.
- ▶ Even if there is an incorrect conditional expression inside each conditional expression, the error is hard to identify.
- ▶ The possibility of errors increases because you need to identify the meaning of each if statement.
- ▶ To solve this problem, apply the if-else statement as shown below.

## 2. Nested Conditional Statement

### 2.5. Grade generator example B : Grade generator code using if-else statement

```
1 score = int(input('Enter score : '))
2 if score >= 90:                      # If 90 or above, 'A'
3     grade = 'A'
4 else:
5     if score >= 80:                  # If under 90, 80 or above, 'B'
6         grade = 'B'
7     else:
8         if score >= 70:              # If under 80, 70 or above, 'C'
9             grade = 'C'
10    else:
11        if score >= 60:              # If under 70, 60 or above, 'D'
12            grade = 'D'
13        else:                      #If under 60, 'F'
14            grade = 'F'
15 print('Your grade is: ', grade)
```

Enter score : 88

Your grade is: B

- ▶ Grade B for 88 points is properly printed.
- ▶ The likelihood of errors is reduced.
- ▶ Readability is still poor because if-else can only represent two conditions.

## 2. Nested Conditional Statement

### 2.5. Grade generator example B : Grade generator code using if-else statement

```
1 score = int(input('Enter score : '))
2 if score >= 90:           # If 90 or above, 'A'
3     grade = 'A'
4 else:
5     if score >= 80:       # If under 90, 80 or above, 'B'
6         grade = 'B'
7     else:
8         if score >= 70:   # If under 80, 70 or above, 'C'
9             grade = 'C'
10    else:
11        if score >= 60:   # If under 70, 60 or above, 'D'
12            grade = 'D'
13        else:             #If under 60, 'F'
14            grade = 'F'
15 print('Your grade is: ', grade)
```

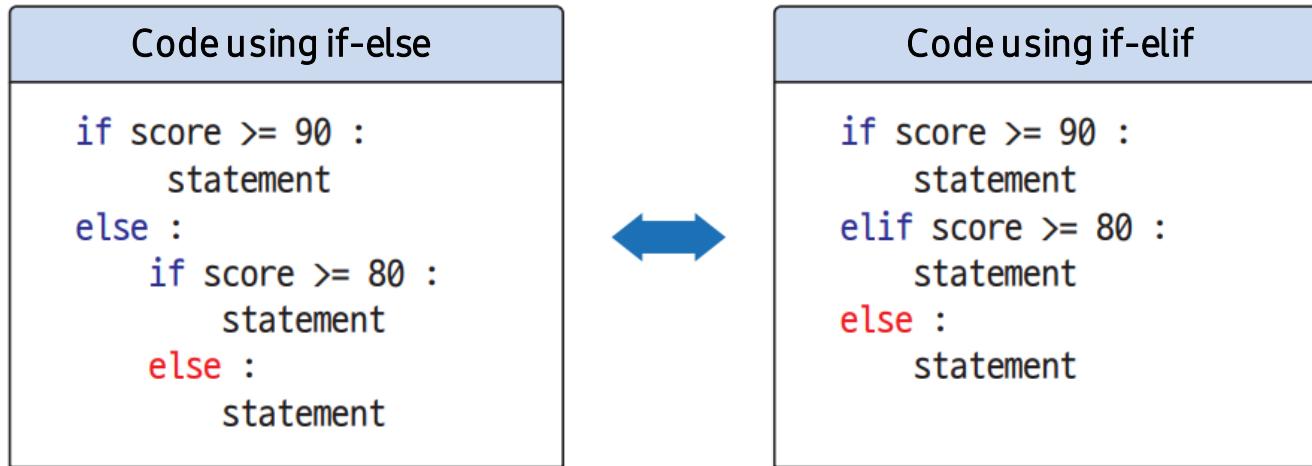
Enter score : 88

Your grade is: B

- ▶ There are too many indentations.
- ▶ For multiple conditions, what could be a simpler way?

## 2. Nested Conditional Statement

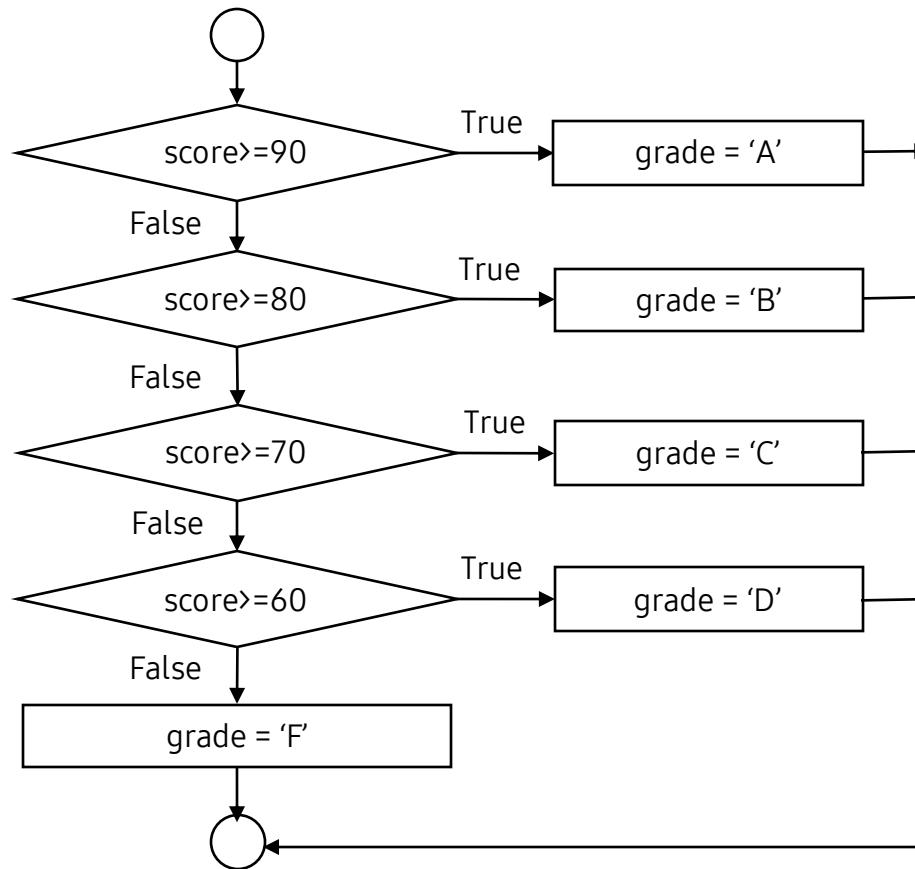
### 2.6. Comparing if-elif statement and elif statement



- ▶ Both codes perform the same task.
- ▶ The code on the right is indented less and the number of lines is reduced. Thus the code is easier to read and understand.

## 2. Nested Conditional Statement

### 2.7. Grade generator example C: if-elif-else statement execution flowchart



## 2. Nested Conditional Statement

### 2.8. Grade generator example C: grade generator code using if-elif-else statement

```
1 score = int(input('Enter score : '))
2 if score >= 90:      # If 90 or above, 'A'
3     grade = 'A'
4 elif score >= 80:    # If not 'A', 'B' if 80 or above
5     grade = 'B'
6 elif score >= 70:    # If not 'B', 'C' if 70 or above
7     grade = 'C'
8 elif score >= 60:    # If not 'C', 'D' if 60 or above
9     grade = 'D'
10 else:                # Otherwise, 'F'
11     grade = 'F'
12 print('Your grade is: ', grade)
```

Enter score : 88

Your grade is: B

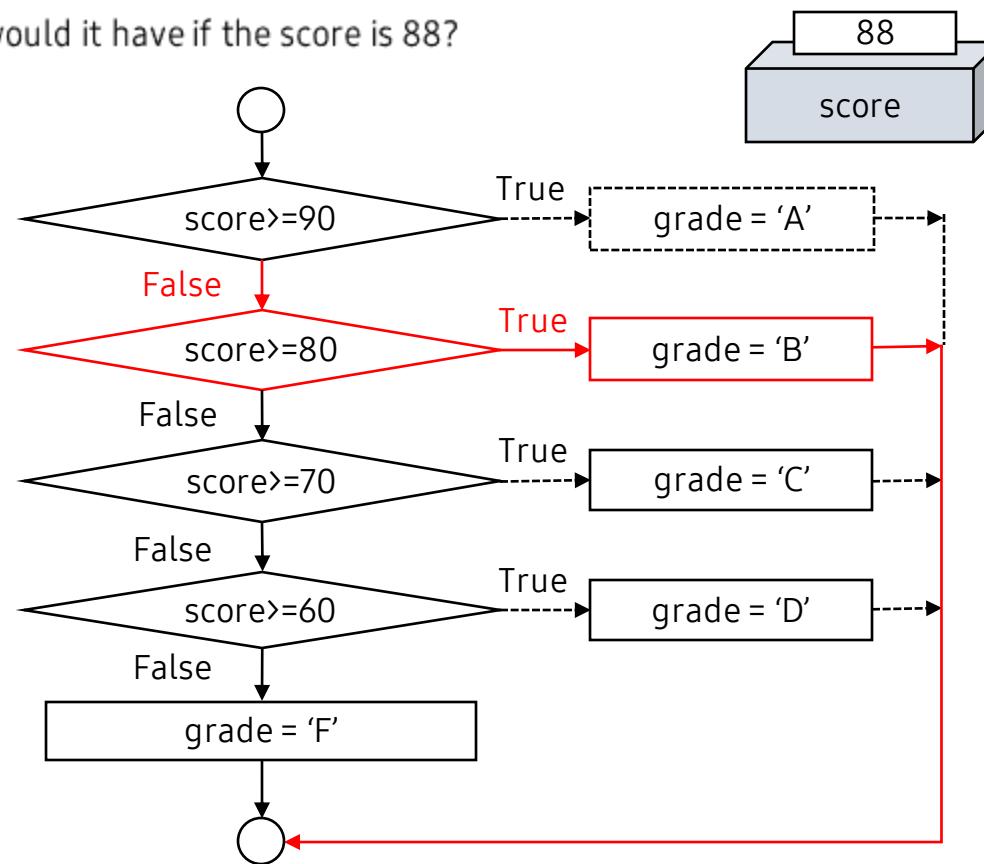
- ▶ Grade B for 88 points was properly printed.
- ▶ Compared with the examples A and B, the chance of an error is smaller because example C used the if-elif-else statement.
- ▶ Also the code is easy to understand because the condition is clear.

## 2. Nested Conditional Statement

### 2.8. Grade generator example C: grade generator code using if-elif-else statement

| Grade generator example C: What flow would it have if the score is 88?

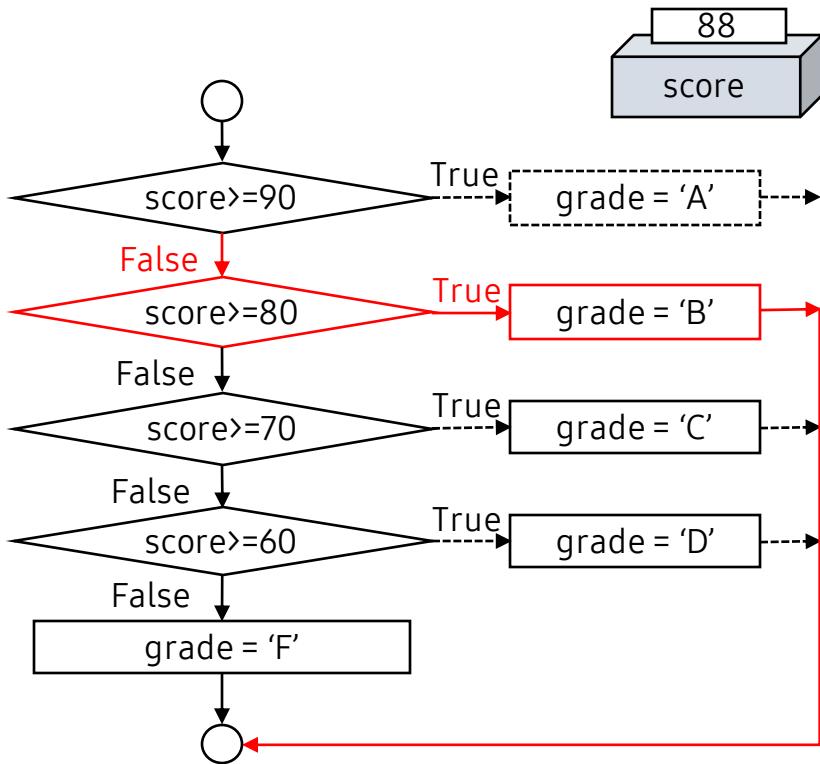
When the value of the score variable is 88, it has the following execution flow.



## 2. Nested Conditional Statement

### 2.8. Grade generator example C: grade generator code using if-elif-else statement

| Grade generator example C: What flow would it have if the score is 88? Follow the diagram.



- ▶ First, the program executes the statement `if score >=90:` to determine whether the score value is greater than 90.
- ▶ Since the result of the comparison is False, the statement `elif score>=80:` is performed to determine whether the score value is greater than 80.
- ▶ Now that this result is True, the `grade = 'B'` sentence is executed, and the execution of all `elif-else` statements below are skipped.
- ▶ Therefore, the program finally prints "your grade is : B".

## 2. Nested Conditional Statement

### 2.9. Diverse and complex real-life problems



Programmers must deal with more complex problems than this.

To impose more sophisticated conditions, you can combine comparison operators and logical operators.

Python has the following comparison operators and logical operators, all of which **return Boolean values (True, False)**.

Some are examples of multiple operators combined.

Comparison operator	Explanation
<code>==</code>	Returns True if two operands have identical value.
<code>!=</code>	Returns True if the values of the two operands are different.
<code>&gt;</code>	Returns True if the left operand is greater than the right operand.
<code>&lt;</code>	Returns True if the left operand is less than the right operand.
<code>&gt;=</code>	Returns True if the left operand is greater than or equal to the right operand.
<code>&lt;=</code>	Returns True if the left operand is less than or equal to the right operand.

Logical operator	Meaning
<code>x and y</code>	If one of x or y is False, it is False. It is only True if it all are True.
<code>x or y</code>	If either x or y is True, it is True, and False only if all are False.
<code>not x</code>	If x is true, it is False, and if x is false, it is True.

## 2. Nested Conditional Statement

### 2.10. Example of a complex conditional expression

```
1 a = 10
2 b = 14          # if b is changed to 13, it does not meet the first conditional statement
3 if (a % 2 == 0) and (b % 2 == 0):    # first conditional statement
4     print('Both are positive.')
5 if (a % 2 == 0) or (b % 2 == 0):    # second conditional statement
6     print('At least one is positive.)
```

Both are positive.

At least one is positive.

- ▶ The above code that generates results with two conditions when a = 10 and b = 14.
- ▶ The if statement in this code prints outputs by using logical operators and and or.
- ▶ In this code, (a% 2==0) is True and (b% 2==0) is True. Therefore, both print statements of lines 3 and 4 are printed.

## 2. Nested Conditional Statement

### 2.10. Example of a complex conditional expression

```
1 a = 10
2 b = 13          # if b is changed to 13, it does not meet the first conditional statement
3 if (a % 2 == 0) and (b % 2 == 0):      # first conditional statement
4     print('Both are positive.')
5 if (a % 2 == 0) or (b % 2 == 0):       # second conditional statement
6     print('More than one is positive.)
```

More than one is positive.

- ▶ The above code generates results with two conditions when  $a = 10$  and  $b = 14$ .
- ▶ The if statement in this code prints outputs by using logical operators and and or.
- ▶ Here, since  $b$  is 13,  $(b \% 2 == 0)$  is False.
- ▶ Therefore,  $(a \% 2 == 0)$  and  $(b \% 2 == 0)$  are False, so 'Both numbers are even.' is not printed.
- ▶ On the other hand, since  $(a \% 2 == 0)$  is True,  $(a \% 2 == 0)$  or  $(b \% 2 == 0)$  is True.

 One More Step

- ▶ Distinguish the leap year

The simpler the code, easier it is to code and the easier it is for the reader to understand. Such codes are called "codes with good readability." To review the previous mission, find a leap year by using complex conditional expressions and assign it to `is_leap_year`. `is_leap_year` is True when the year value is a leap year, otherwise False.

```
1 year = int(input('Enter the year: '))
2 is_leap_year = ((year % 4 == 0) and (year % 100 != 0) or (year % 400 == 0))
3 print(year, 'is a leap year?', is_leap_year)
```

Enter the year : 2021  
2021 is a leap year? False

- ▶ If complex conditional expressions are used well, the code can be more concise.

## 3. Application production practice

### 3.1. Application exercise

- | Suppose that there are four types of fruits in David's fruit shop as shown below, and the type and number of fruits are inputs.
- | Make the code print a message that informs the user of the total amount and requests a deposit.
- | Print out the goods and tell them the amount of change. If money is still insufficient, send out a message that says money is insufficient.

```
#####
This is David's fruit shop.  
1: Apple( price : 5000 won)  
2: Grape( price : 6000 won)  
3: Melon( price : 8000 won)  
4: Orange( price : 2000 won)  
#####

Enter item number (between 1~4) >> 2  
Enter number of item(between 1~10) >> 2  
Fruit selected : grape  
Price : 6000  
Quantity: 2  
Total price is 12000 won.  
Insert money please(ex: 15000) >>> 20000  
20000 won received. Your change is 8000 won.
```

## 3. Application production practice

### 3.1. Application exercise

| Display the menu and receive input as following.

```
1 print('#' * 50)
2 print('This is David's fruit shop.')
3 print('1: Apple( price : 5000 won)')
4 print('2: Grape( price : 6000 won)')
5 print('3: Melon( price : 8000 won)')
6 print('4: Orange( price : 2000 won)')
7 print('#' * 50)
8
9 total_price = 0
10 order = int(input(('Enter item number(between 1~4) >> ')))
11 count = int(input(('Enter number of items(between 1~10) >> ')))
```

### 3. Application production practice

#### 3.1. Application example

| Assign the selected fruit and its price and compute the total price. Use an if statement to print 'Not enough money.' if money is insufficient. If not, print the amount of change.

```
13 if order == 1:  
14     fruit = 'Apple'  
15     price = 5000  
16 elif order == 2:  
17     fruit = 'Grape'  
18     price = 6000  
19 elif order == 3:  
20     fruit = 'Melon'  
21     price = 8000  
22 else:  
23     fruit = 'Orange'  
24     price = 2000  
25  
26 print('Fruit selected :', fruit)  
27 print('Price :', price)  
28 print('Quantity :', count)  
29 print('Total price is', price * count, 'won.')  
30  
31 money = int(input('Insert money please(ex: 15000) >>>'))  
32 if money < price * count :  
33     print('Not enough money.')  
34 else :  
35     change = money - price * count  
36     print(money, 'won received. Your change is', change, 'won.')
```

## 3. Application production practice

### 3.2. Key code commentary of the application example

#### I Key code commentary

- ▶ The central code of this program is as follows.
- ▶ If the order number is 1, the fruit is 'apple' and the price is 5000. If not, the code processes another condition verification through the if-elif statement.
- ▶ In such way, the if-elif-else statement simplifies the code.

```
13 if order == 1:  
14     fruit = 'Apple'  
15     price = 5000  
16 elif order == 2:  
17     fruit = 'Grape'  
18     price = 6000  
19 elif order == 3:  
20     fruit = 'Melon'  
21     price = 8000  
22 else:  
23     fruit = 'Orange'  
24     price = 2000
```

# | Paper coding

**Try to fully understand the basic concept before moving on to the next step.**

**Lack of understanding basic concepts will increase your burden in learning this course, which may make you fail the course.**

**It may be difficult now, but for successful completion of this course we suggest you to fully understand the concept and move on to the next step.**

**Q1.**

Receive an alphabetical letter from the user and print 'It is a vowel' for a, e, i, o, u, and 'It is a consonant' for any other letters.

Example Output

```
Enter the alphabet : k  
K is a consonant
```

Time

5min



Write the entire code and the expected output results in the note.

**Q2.** Write the following program which receives two integers a, b as input, determines whether a is a multiple of b and prints the result.

Example Output

Write two integers: 30 3  
30 is a multiple of 3

Time

5min



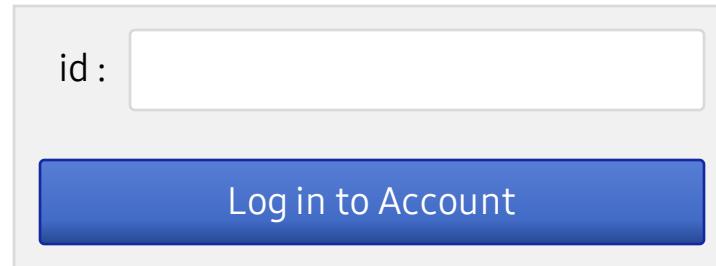
Write the entire code and the expected output results in the note.

| Let's code

## 1. if-elif Statement Application Example

### 1.1. Program for ID verification

| Let's write a program that receives an ID from the user and prints whether it matches the ID 'ilovepython' stored in the program.



The image shows a simple user interface for account login. It consists of two main components: a text input field and a button. The text input field is labeled "id :" and contains a blank input box. Below the input field is a blue button with the text "Log in to Account".

## 1. if-elif Statement Application Example

### 1.2. The program resolution code for ID verification.

| Problem solving code using if-else statement

```
1 d = "ilovepython"
2 s = input("Enter the ID: ")
3 if s == id:
4     print("Welcome.")
5 else:
6     print("ID not found.")
```

Enter the ID: ilovepython

Welcome.

- ▶ Take a string input using the input function. Next, the relational operator == checks whether the input string identical to 'ilovepython'. If the result of the relational operator is True, print 'Welcome'.



## One More Step

- ▶ Implementing login function using ID and password

Check the password after the ID verification is finished. That is, write a program with the following output, in which stored password is mypass1234. If both are correct, print "Welcome". Otherwise print "ID not found" or "The password is wrong."

Enter your ID:

Password

**Log in to Account**

Enter ID:

Enter Password:

Welcome.

## 1. if-elif Statement Application Example

### 1.3. Program to verify ID and password

```
1 my_id = "ilovepython"
2 password = "mypass1234"
3 string1 = input("Enter ID: ")
4 string2 = input("Enter Password: ")
5 if string1 == my_id and string2 == password:
6     print("Welcome.")
7 elif string1 != my_id:
8     print("ID not found.")
9 else:
10    print("The password is wrong.")
```

```
Enter ID: ilovepython
Enter password: mypass1234
Welcome.
```



Line 2~3

- Receive ID and password from the user.

## 1. if-elif Statement Application Example

### 1.3. Program to verify ID and password

```
1 my_id = "ilovepython"
2 password = "mypass1234"
3 string1 = input("Enter ID: ")
4 string2 = input("Enter Password: ")
5 if string1 == my_id and string2 == password:
6     print("Welcome.")
7 elif string1 != my_id:
8     print("ID not found.")
9 else:
10    print("The password is wrong.")
```

```
Enter ID: ilovepython
Enter password: mypass1234
Welcome.
```

#### Line 5~6

- Compare whether the string input by the user both match my\_id and password. Since we used the and operator, the block below is not executed if either one is wrong.
- If the fifth line is True, 'Welcome' is printed.

## 1. if-elif Statement Application Example

### 1.3. Program to verify ID and password

```
1 my_id = "ilovepython"
2 password = "mypass1234"
3 string1 = input("Enter ID: ")
4 string2 = input("Enter Password: ")
5 if string1 == my_id and string2 == password:
6     print("Welcome.")
7 elif string1 != my_id:
8     print("ID not found.")
9 else:
10    print("The password is wrong.")
```

```
Enter ID: ilovepython
Enter password: mypass1234
Welcome.
```



#### Line 7~8

- If the if statement on the fifth line is false, checks if my\_id is different from string1 (user input value) in the elif statement.
- If different, 'ID not found' is printed.

## 1. if-elif Statement Application Example

### 1.3. Program to verify ID and password

```
1 my_id = "ilovepython"
2 password = "mypass1234"
3 string1 = input("Enter ID: ")
4 string2 = input("Enter Password: ")
5 if string1 == my_id and string2 == password:
6     print("Welcome.")
7 elif string1 != my_id:
8     print("ID not found.")
9 else:
10    print("The password is wrong.")
```

```
Enter ID: ilovepython
Enter password: mypass1234
Welcome.
```

#### Line 9~10

- If both if and elif statements in Line 5 and Line 7 are false, the print statement of the last else statement is executed.

## 2. random Module Preview

### 2.1. Code example of random number generation

- I Study random number generation needed to solve this unit's mission. Random numbers can be generated in Python as follows. The code below generates a random number between 1 and 100.
- I When the random module's randint(1, 100) function is called, a random number between 1 and 100 is generated (the random number value includes 1 and 100).

```
1 import random      # call the random module  
2  
3 number = random.randint(1, 100)  
4 print(number)
```

93



#### Line 1

- In order to use the function randint which produces random numbers between 1 and 100, the random number module must be imported using an import statement.

## 2. random Module Preview

### 2.2. Coin tossing game using a random function

- | Write a coin tossing game code. Throwing coins can be done by generating random numbers.
- | In Python, after you import random and call the random.randrange(2) function, 0 or 1 may be randomly generated.



## 2. random Module Preview

### 2.2. Coin tossing game using a random function

```
1 import random  
2  
3 number = random.randrange(2)  
4 print(number)
```

1

- ▶ This is a test code for making a coin tossing game with a random function.
- ▶ Run several times. 0 or 1 will be printed.

## 2. random Module Preview

### 2.2. Coin tossing game using a random function

- I Randomly generate 0 or 1 using the randrange function.
- I If this value is 0, print 'front' as shown in the code below, otherwise print 'back'.

```
1 import random
2
3 print("Start coin tossing game. ")
4 coin = random.randrange(2)
5 if coin == 0:
6     print("Front.")
7 else:
8     print("Back.")
9 print("Game finished.")
```

Start coin tossing game.

Front.

Game finished.



Line 4

- The function called randrange(2) is a function that returns 0 or 1 through a random number generator.

# | Pair programming



# Pair Programming Practice

## Guideline, mechanisms & contingency plan

Preparing pair programming involves establishing guidelines and mechanisms to help students pair properly and to keep them paired. For example, students should take turns “driving the mouse.” Effective preparation requires contingency plans in case one partner is absent or decides not to participate for one reason or another. In these cases, it is important to make it clear that the active student will not be punished because the pairing did not work well.

## Pairing similar, not necessarily equal, abilities as partners

Pair programming can be effective when students of similar, though not necessarily equal, abilities are paired as partners. Pairing mismatched students often can lead to unbalanced participation. Teachers must emphasize that pair programming is not a “divide-and-conquer” strategy, but rather a true collaborative effort in every endeavor for the entire project. Teachers should avoid pairing very weak students with very strong students.

## Motivate students by offering extra incentives

Offering extra incentives can help motivate students to pair, especially with advanced students. Some teachers have found it helpful to require students to pair for only one or two assignments.



# Pair Programming Practice



## | Prevent collaboration cheating

The challenge for the teacher is to find ways to assess individual outcomes, while leveraging the benefits of collaboration. How do you know whether a student learned or cheated? Experts recommend revisiting course design and assessment, as well as explicitly and concretely discussing with the students on behaviors that will be interpreted as cheating. Experts encourage teachers to make assignments meaningful to students and to explain the value of what students will learn by completing them.

## | Collaborative learning environment

A collaborative learning environment occurs anytime an instructor requires students to work together on learning activities. Collaborative learning environments can involve both formal and informal activities and may or may not include direct assessment. For example, pairs of students work on programming assignments; small groups of students discuss possible answers to a professor's question during lecture; and students work together outside of class to learn new concepts. Collaborative learning is distinct from projects where students "divide and conquer." When students divide the work, each is responsible for only part of the problem solving and there are very limited opportunities for working through problems with others. In collaborative environments, students are engaged in intellectual talk with each other.

### Q1.

Write a program that executes addition, subtraction, multiplication, and division. It prints the operation result of two positive integers, based on the desired operation number given as input. If a number else than 1, 2 ,3 and 4 is given as input, 'Entered an incorrect number' is printed. To enter two numbers, write one, press enter and write another one.

#### Output example

```
1) Addition    2) Subtraction   3) Multiplication   4) Division  
Enter the desired number of operation : 1  
Enter two numbers for operation.  
10  
20  
10 + 20 = 30
```

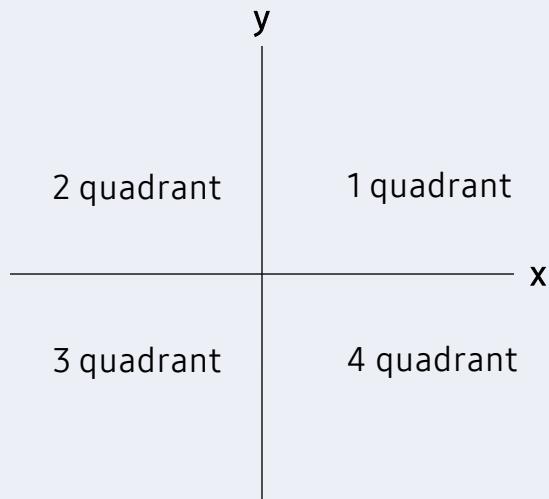
#### If inserted incorrectly

```
1) Addition    2) Subtraction   3) Multiplication   4) Division  
Enter the desired number of operation : 5  
Entered an incorrect number.
```

**Q2.** Write a program that receives a point with x and y coordinates as input, and determines in which quadrant among 1, 2, 3, 4 the point belongs. The position of the quadrant is shown in the following figure.

## Output example

```
Enter x,y coordinates: -5 6  
In the second quadrant
```



**Q3.**

Develop a menu ordering program for Yummy Restaurant. Show the following menu to the user and let the user select one. If the given input alphabet is not in the menu, print 'enter the menu again:' and receive another input.

## Output example

```
Welcome to yummy restaurant. Here is the menu.  
- Burger(enter b)  
- Chicken(enter c)  
- Pizza(enter p)
```

```
Choose a menu (enter b,c,p) : b  
You chose pizza.
```

- ▶ This needs complex conditional expressions. Combine the logical operators and the conditional statements carefully.

Unit 8.

# Loop-1

## Learning objectives

- ✓ Understand the purpose of for loop statement and write loop statements.
- ✓ Understand the concept of the range function and use it in iteration.
- ✓ Understand and utilize for in syntax and list.

## Lesson overview

- ✓ Understand the necessity of iteration and know how to use them.
- ✓ Repeat for a given number of times using for loops
- ✓ Learn how to write various iterations using for in statements and range.
- ✓ Learn how to generate sequences of various numbers using range functions and lists.

## Concepts You Will Need to Know From Previous Units

- ✓ Understand various conditional statements such as the if statement and the if-elif-else statement.
- ✓ Understand the structure of non-sequential executable statements in which the lower block is executed when the conditions are met.

# Keyword

for

in

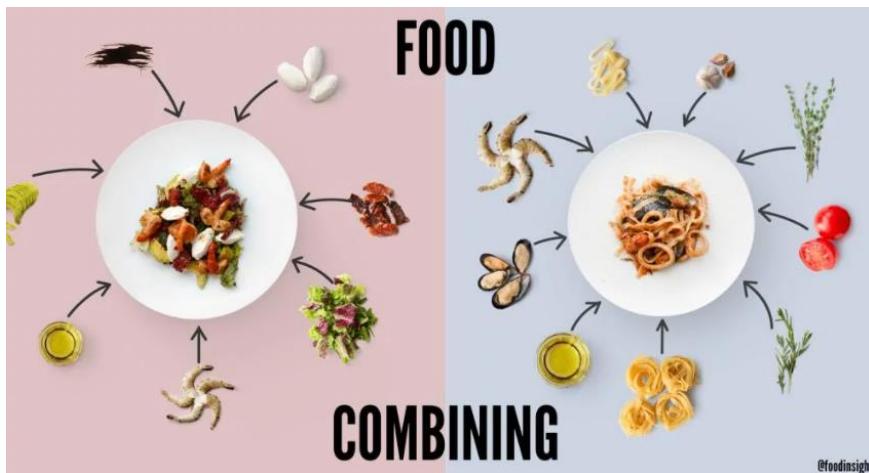
iteration

range

# Mission

### 3. Mission

#### 3.1. How to make Delicious Food Combinations



<https://foodinsight.org/what-is-food-combining/>

- ▶ David opened a new sandwich shop. David wants to print all kinds of sandwiches that can be made through combinations of bread, meat, vegetables, and sauces available at his store.
- ▶ David's bread types include "Rye bread," "Wheat" and "White" meat types include "Meatball", "Sausage" "Chicken breast", vegetable types include "Lettuce", "Tomato", "Cucumber", and sauces include "Mayonnaise", "Honey Mustard", and "Chili".
- ▶ Each ingredient type can be selected only once. What are the possible combinations? Let's print all the combinations as follows.

### 3. Mission

#### 3.2. Delicious Food Combinations operation process

```
1 breads = ["Rye bread", "Wheat", "White"]
2 meats = ["Meatball", "Sausage", "Chicken breast"]
3 vegis = ["Lettuce", "Tomato", "Cucumber"]
4 sauces = ["Mayonnaise", "Honey Mustard", "Chili"]
5
6 print('Possible combinations of David's sandwich shop')
7 for b in breads:
8     for m in meats:
9         for v in vegis:
10            for s in sauces:
11                print(b+ " + "+ m+ " + "+ v+ " + "+ s)
12
```

Possible combinations of David's sandwich shop  
Rye bread + Meatball + Lettuce + Mayonnaise  
Rye bread + Meatball + Lettuce + Honey Mustard  
Rye bread + Meatball + Lettuce + Chili  
Rye bread + Meatball + Tomato + Mayonnaise  
Rye bread + Meatball + Tomato + Honey Mustard  
Rye bread + Meatball + Tomato + Chili  
Rye bread + Meatball + Cucumber + Mayonnaise  
Rye bread + Meatball + Cucumber + Honey Mustard

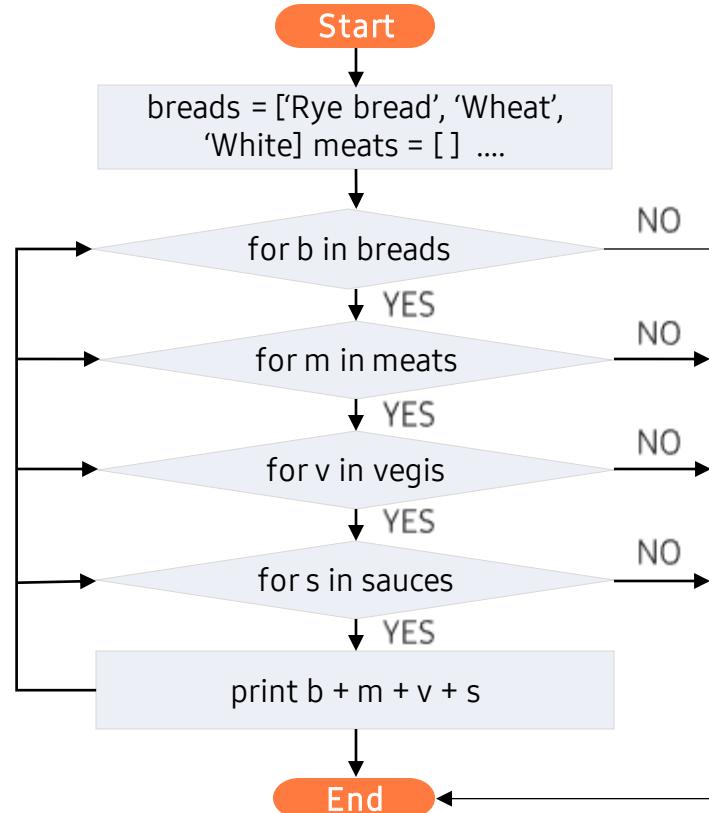
### 3. Mission

#### 3.3. Programming Plan

##### Pseudocode

- [1] Start
- [2] Define list-type data containing food ingredients.
- [3] the number of elements in for b in bread list :
- [4] the number of elements in for m in meats list :
- [5] the number of elements in for v in vegis list :
- [6] the number of elements in for s in sauces list :
- [7] print b + m + v + s.
- [8] end

##### Flowchart



### 3. Mission

#### 3.4. Delicious Food Combinations final code

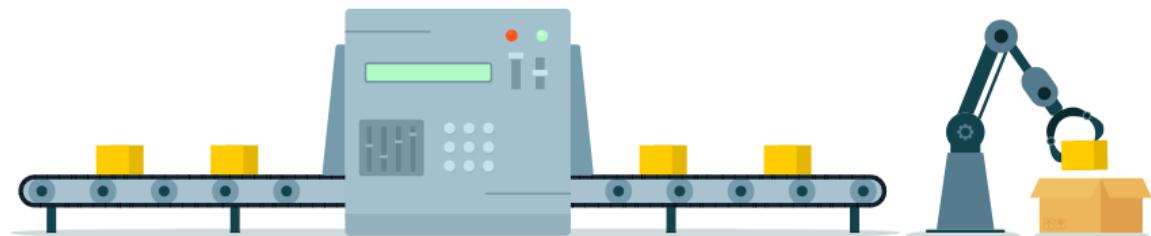
```
1 breads = ["Rye bread", "Wheat", "White"]
2 meats = ["Meatball", "Sausage", "Chicken breast"]
3 vegis = ["Lettuce", "Tomato", "Cucumber"]
4 sauces = ["Mayonnaise", "Honey Mustard", "Chili"]
5
6 print('Possible combinations of David's sandwich shop')
7 for b in breads:
8     for m in meats:
9         for v in vegis:
10            for s in sauces:
11                print(b+ " + "+ m+ " + "+ v+ " + "+ s)
12
```

# | Key concept

## 1. Basics of for statement

### 1.1. Necessity of iterated statements

- | Iterating a code according to its conditions helps solving complex problems.
- | For example, a program becomes much more concise by using an iterated structure instead of copy and pasting certain statements to repeat a same task.
- | In addition, loop statements can reduce the time required for programming.



## 1. Basics of for statement

### 1.1. Necessity of iterated statements

```
1 print('Welcome to everyone!!!')
2 print('Welcome to everyone!!!')
3 print('Welcome to everyone!!!')
4 print('Welcome to everyone!!!')
5 print('Welcome to everyone!!!')
```

Welcome to everyone!!  
Welcome to everyone!!  
Welcome to everyone!!  
Welcome to everyone!!  
Welcome to everyone!!

- ▶ Write print function 5 times to print 5 times repeatedly without using loop statements.
- ▶ Consider repeating this task 1000 times.
- ▶ This is inefficient.

## 1. Basics of for statement

### 1.1. Necessity of iterated statements

| Why is repetition important?

```
1 for i in range(5):
2     print('Welcome to everyone!!')
```

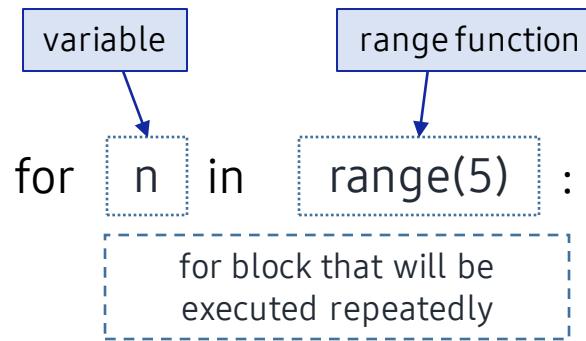
Welcome to everyone!!  
Welcome to everyone!!  
Welcome to everyone!!  
Welcome to everyone!!  
Welcome to everyone!!

- ▶ Of course, for only few repetitions, you may "copy and paste" as above.
- ▶ But can you repeat it 100 times?
- ▶ In this case, you can write a simple code by using the for statement.
- ▶ loop statements are important control statements that handle inconvenient repetitive tasks.

## 1. Basics of for Loop

### 1.2. Basics of range and loop statement syntax

- | The range function generates an integer sequence of a specific interval.
- | It can be used for circulation in the for loop.



## 1. Basics of for Loop

### 1.3. for statement example

```
1 for i in range(10):  
2     print('Welcome to everyone!!')
```

Welcome to everyone!!  
Welcome to everyone!!

- ▶ To repeat the above code 10 times, put 10 inside the parentheses range () .
- ▶ This simple code executes a huge amount of tasks, and code modification and maintenance becomes easier.
- ▶ To repeat 100 times : change to range(100).

## 1. Basics of for Loop

### 1.4. loop control variable

I Alphabet i, j, k, l, ... are used as control variable in loop.

```
1 for i in range(10):  
2     print(i, 'Welcome to everyone!!')
```

```
0 Welcome to everyone!!  
1 Welcome to everyone!!  
2 Welcome to everyone!!  
3 Welcome to everyone!!  
4 Welcome to everyone!!  
5 Welcome to everyone!!  
6 Welcome to everyone!!  
7 Welcome to everyone!!  
8 Welcome to everyone!!  
9 Welcome to everyone!!
```

- ▶ Variables used for loop statements are generally assigned with alphabets i, j, k, l, ...
- ▶ These variables are called **loop** control variables in C or Java.

 One More Step

## I Anonymization of loop control variables

- ▶ Since the newly assigned variable i in the above loop statement is a loop variable that is not used in the execution statement, you can anonymized the variable by substituting it with an underscore (\_) as follows.

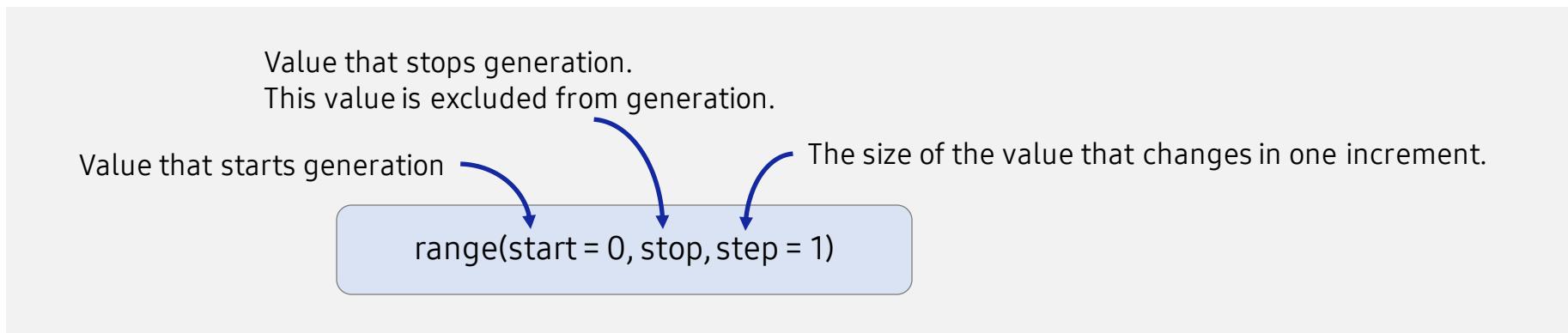
```
1 for _ in range(10):  
2     print('Welcome to everyone!!')
```

Welcome to everyone!!  
Welcome to everyone!!

## 2. range

### 2.1. Structure of range

- | You can generate successive integers between a given start value and the stop value, as in `range(0, 5)`, or you can add an incremental value at the end, as in `range(0, 5, 2)`.
- | In `range(0, 5, 1)`, the last 1 is a default step value that is added in each increment.



- | Calling `range(0, stop, 1)` generates integers from 0 to  $(\text{stop}-1)$  in increments of 1.

 One More Step

I Arguments of a range function cannot be real numbers.

- ▶ A range function cannot have a real number arguments. However, a function called `arrange()` in the `numpy` module (which will be dealt later) can take real numbers for start, step and stop values.

 TypeError

```
1 for i in range(0.9):  
2     print(i, 'Welcome to everyone!!')
```

```
TypeError                                 Traceback (most recent call last)  
<ipython-input-11-332ae3dce87d> in <module>  
----> 1 for i in range(0.9):  
      2     print(i, 'Welcome to everyone!!')
```

**TypeError:** 'float' object cannot be interpreted as an integer

## 2. range

### 2.2. About range

| A list can be easily created using the list function as follows.

```
1 print(list(range(0, 5, 1)))
```

```
[0, 1, 2, 3, 4]
```

```
1 print(list(range(0, 5, 2)))
```

```
[0, 2, 4]
```

```
1 print(list(range(2, 5)))
```

```
[2, 3, 4]
```

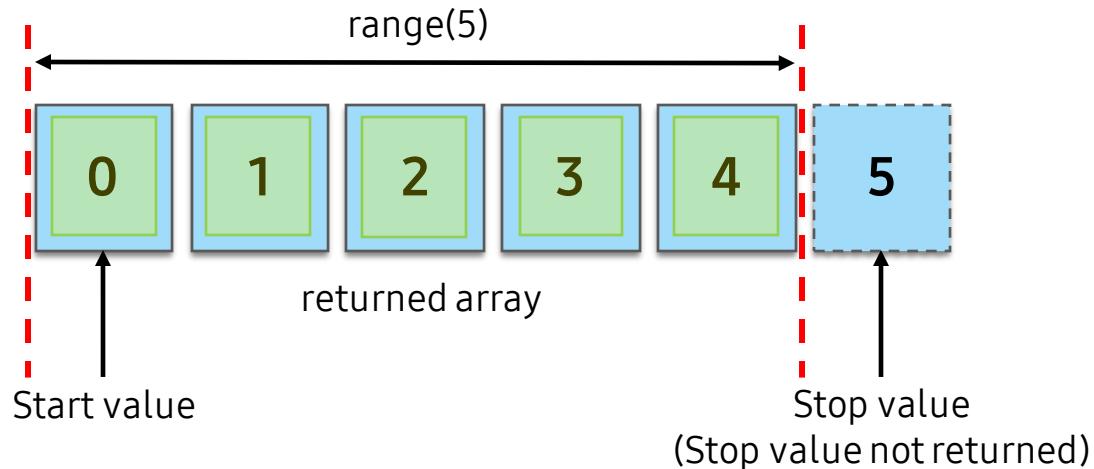
```
1 print(list(range(0, 10, 2)))
```

```
[0, 2, 4, 6, 8]
```

## 2. range

### 2.2. About range

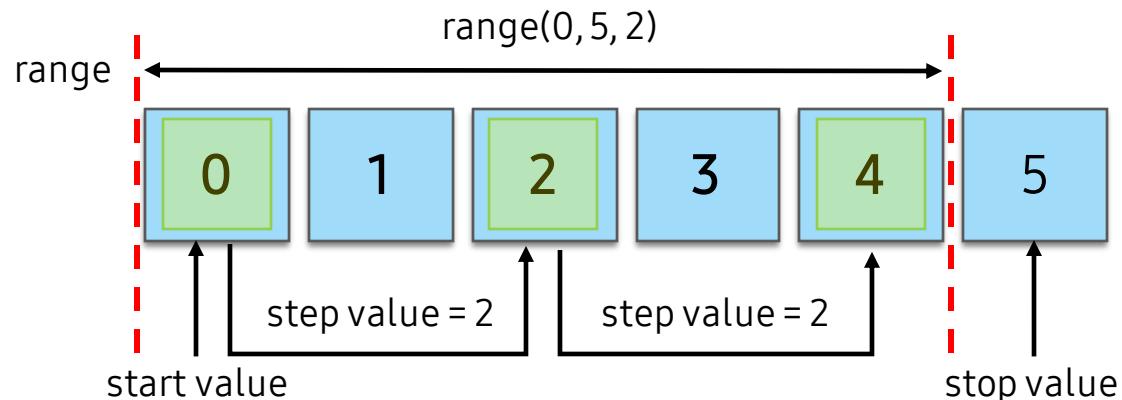
| Start and stop value of range(5) and returned array.



## 2. range

### 2.2. About range

- | range(2, 5) generates an integer array greater than or equal to 2 and less than 5, thus [2, 3, 4]. range(0, 5, 2) increases 2 when generating integers above or equal to 0 and less than 5. Thus, it generates integers [0, 2, 4].
- | For positive step values, the start value must be less than the stop value, as in range(0, 5, 1). For negative step values, the start value must be greater than the stop value, as in range(-2, -10, -2).



## 2. range

### 2.3. Problem solving using for in range

The following for loop can be used to find the sum of integers from 1 to 10.

```
1 s = 0
2 for i in range(1, 11):
3     s += i
4 print('Sum of numbers from 1 to 10: ', s)
```

Sum of numbers from 1 to 10: 55

 One More Step

## The importance of initialization in loop statements, cumulative operation

Iteration	i value	s value	Iterate or not	Computing process of s (Initial s is 0)
1st	1	1	Iterate	0+1
2nd	2	3	Iterate	0+1+2
3rd	3	6	Iterate	0+1+2+3
4th	4	10	Iterate	0+1+2+3+4
5th	5	15	Iterate	0+1+2+3+4+5
6th	6	21	Iterate	0+1+2+3+4+5+6
7th	7	28	Iterate	0+1+2+3+4+5+6+7
8th	8	36	Iterate	0+1+2+3+4+5+6+7+8
9th	9	45	Iterate	0+1+2+3+4+5+6+7+8+9
10th	10	55	Stop Iterating	0+1+2+3+4+5+6+7+8+9+10

- If the for statement is iterated in the program, i and s values change as follows.
- The value of i changes after each loop, and for this reason, the value of s also increases.
- "i" increases by 1, becoming 1, 2, 3... Since s = 0 is the initial state, 0+1 is assigned to s during the first iteration, but because the code executes s+i every time, it computes 0+1, 0+1+2, 0+1+2+3, ...
- After the tenth iteration, s has a value of 55 and the iteration is stopped.

## 2. range

### 2.4. Overlap of sum() function and variable name sum

- | The sum() function is a Python built-in function that calculates the sum of the elements in a list.
- | Occasionally, a sum variable inside a code can be confused with the Python built-in function sum(). That is, an error occurs if a variable sum = 0 and the function sum (numbers) are called within the same module.
- | The following code prints 55 by adding all values of 1, 2, 3, 4, 5, 6, 7, 8, 9, and 10 generated by range (1, 11) using the sum() function.

```
1 print(sum(range(1,11)))
```

55

- ▶ The sum() function calculates the sum of the elements in the list. Thus, a sum of numbers from 1 to 10, 55 is printed.



## Overlapping use of sum() function and variable name

```
1 sum = 0 ←  
2 for i in range(1, 11):  
3     sum += i  
4 print('The sum of numbers from 1 to 10: ', sum)  
5 print(sum(range(1,11)))
```

Since the sum function has already been used as the name of a variable, it cannot be called afterwards.

The sum of numbers from 1 to 10 : 55

```
-----  
TypeError                                     Traceback (most recent call last)  
<ipython-input-34-991f98d67989> in <module>  
      3     sum += i  
      4 print('The sum of numbers from 1 to 10: ', sum)  
----> 5 print(sum(range(1,11)))
```

TypeError: 'int' object is not callable

- ▶ If a variable sum is used inside the code as shown above, it may be confused with the Python built-in function sum().
- ▶ An error occurs when a variable called sum = 0 and a function called sum (numbers) are called within the same module.
- ▶ Therefore, it is recommended to use s or total as the name of the variable, so that it does not overlap with the sum() function.

## 2. range

### 2.5. Computing factorial

| n factorial is the value multiplied by all natural numbers from 1 to n when n is an arbitrary natural number.

$$n! = \prod_{k=1}^n k = n \cdot (n - 1) \cdot (n - 2) \cdots \cdots 3 \cdot 2 \cdot 1$$

```
1 n = int(input('Enter a number: '))
2 fact = 1
3 for i in range(1, n+1):
4     fact = fact * i
5 print(str(n) + '! = ', fact)
```

Enter a number :

5! = 120



## One More Step

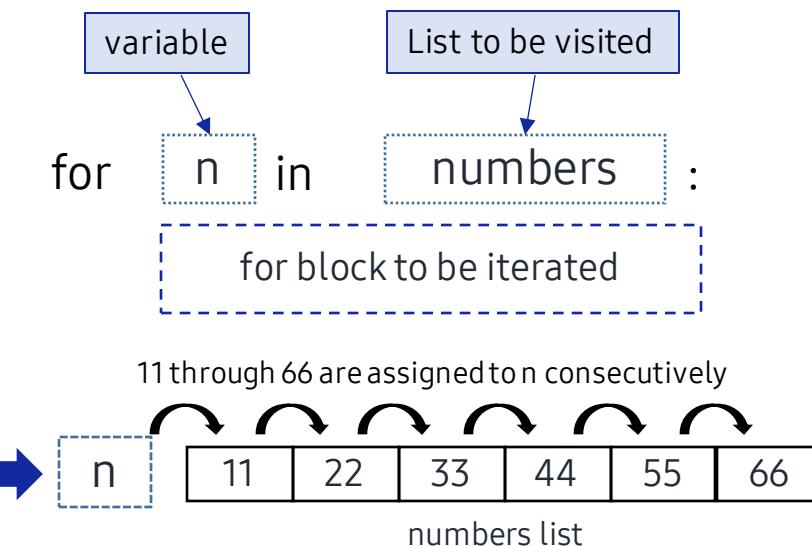
### I Python and limitations in integer expressions

- ▶ If we change the value of n in the code above to 50, it prints an enormous number of the following.
- ▶  $50! = 30414093201713378043612608166064768844377641568960512000000000000$ .
- ▶ In programming languages such as C and Java, integers larger than a certain size cannot be expressed because the integer size is set to 4 byte types. For example, Java's long-type integer ranges from -9223372036854775808 to 9223372036854775807. However in Python, there is no limitation in integer expression. This is another big advantage of Python.

### 3. Sequence object iteration

#### 3.1. Object with order

- Declare a new variable n to be assigned between the loop keyword for and in. Then put a numbers list with elements [11, 22, 33, 44, 55, 66] after 'in'.
- The code below performs an execution which sequentially visits the values inside the list after in. The n variable is sequentially assigned 11, 22, 33, 44, 55, and 66.



## 3. Sequence object iteration

### 3.1. Object with order

- Declare a new variable n to be assigned between the loop keyword for and in. Put the numbers list of elements [11, 22, 33, 44, 55, 66] after "in". Now, print n inside the for statement.

```
1 numbers = [11, 22, 33, 44, 55, 66]
2 for n in numbers :
3     print('n = ', n) # different values are printed each time
```

```
n = 11
n = 22
n = 33
n = 44
n = 55
n = 66
```

## 3. Sequence object iteration

### 3.2. Cumulative addition code example

- Add the integer items of the list elements. Store the result of the cumulative addition in s and print it.

```
1 numbers = [10, 20, 30, 40, 50]
2 s = 0
3 for n in numbers:
4     s = s + n
5 print('Sum of list items : ', s)
```

Sum of list items : 150

- ▶ This is a program to find the sum of integer item values in the list using the cumulative addition functionality.

## 3. Sequence object iteration

### 3.2. Cumulative addition code example

| Accumulated sum of integer values in the list item: It is possible to simply use the sum() function.

```
1 numbers = [10, 20, 30, 40, 50]
2 print('Sum of list items : ', sum(numbers))
```

Sum of list items : 150

- ▶ The sum of the list elements can be easily calculated using the built-in function sum() without using the for statement.

## 3. Sequence object iteration

### 3.3. Converting string data type to list data type

- You can convert string data types into lists by using the list function.

```
1 st = 'Hello'  
2 print(list(st))
```

```
['H', 'e', 'l', 'l', 'o']
```

- The string data type 'Hello' becomes a list of individual alphabet characters.

## 3. Sequence object iteration

### 3.4. How to use string data type in for loops

```
1 for ch in 'Hello':  
2     print(ch, end = ' ')
```

Hello

- ▶ You can convert the string 'Hello' into a list of character elements by applying the list function.
- ▶ Place the string after the for in expression. This will separate the string into individual characters, enabling the loop to visit each character.
- ▶ Insert the end = ' ' argument inside the print function. It prints another line without changing the line after printing ch.



## One More Step

- | Various options of the print function
- | Do the following to put characters that are not spaces between values. sep is derived from the word separator.

```
1 print('s','e','p', sep='-')
```

s-e-p

- | The keyword argument end specifies a string to be printed after the output value. In the code below, one space was printed using "end=" after "My name is". If you enter something else a string, that string is printed instead of a space.

```
1 print("My name is", end=" ")  
2 print("David")
```

My name is David

```
1 print("My name is", end=" : ")  
2 print("David")
```

My name is : David

# | Paper coding

**Try to fully understand the basic concept before moving on to the next step.**

**Lack of understanding basic concepts will increase your burden in learning this course, which may make you fail the course.**

**It may be difficult now, but for successful completion of this course we suggest you to fully understand the concept and move on to the next step.**

## Q1.

Declaring the list bts = ['V', 'J-Hope', 'RM', 'Jungkook', 'Jin', 'Jimin', 'Suga']. Then write a code that prints all items in this list using the for statement.

Example Output	V J-Hope RM Jungkook Jin Jimin Suga
Time	5min



Write the entire code and the expected output results in the note.

**Q2.** Use cumulative addition to compute and print the sum of integers from 1 to 100.  
(Hint: make the printed value of the range function range from 1 to 100.)

Example Output	Sum of integers from 1 to 100 : 5050
Time	5min

 Write the entire code and the expected output results in the note.

**Q3.** Use the step value of the range function to find the sum of even numbers from 1 to 100.  
(Hint: Set the start value of the range function to zero and the step value as two.)

Example Output	Sum of even numbers from 1 to 100 : 2550
Time	5min



Write the entire code and the expected output results in the note.

**Q4.** Use the step value of the range function to find the sum of odd numbers from 1 to 100.  
(Hint: Set the start value of the range function to one and the step value as two.)

Example Output	Sum of odd numbers from 1 to 100 : 2500
Time	5min



Write the entire code and the expected output results in the note.

| Let's code

## 1. Structure of Loop Statement

### 1.2. Use for loop for iteration of controlled number of times

- When using for loops, practice how to use lists as follows. You can iterate with the following code. Here, [...] is a list data type.
- This will be covered in more detail in a future unit. It can be understood as a shopping cart that carries multiple values. / Think that integers 1, 2, 3, 4 and 5 are contained in the list [1, 2, 3, 4, 5].

```
1 for i in [1, 2, 3, 4, 5]:    # There must be : at the end
2     print("Welcome.")        # Must indent
```

Welcome.  
Welcome.  
Welcome.  
Welcome.  
Welcome.

## 1. Structure of Loop Statement

### 1.3. for statement indentation

- Statements that will be iterated must be indented.
- The statement to be executed is a block inside the for statement. The code returns the following output.

```
1 for i in [1, 2, 3, 4, 5]:    # There must be : at the end
2     print("Welcome.")        # Must indent
```

Welcome.  
Welcome.  
Welcome.  
Welcome.  
Welcome.

- In the first iteration, the value of the variable i becomes 1, the first number of the list, and the print(...) sentence is executed.
- In the second iteration, the value of the variable i becomes 2, the second number in the list, and the print(...) sentence is executed.
- In the third iteration, the value of the variable i becomes 3, the third number of the list, and the print (...) sentence is executed.

## 1. Structure of Loop Statement

### 1.3. for statement indentation

- Statements that will be iterated must be indented.
- The statement to be executed is a block inside the for statement. The code returns the following output.

```
1 for i in [1, 2, 3, 4, 5]:    # There must be : at the end
2     print("Welcome.")        # Must indent
```

Welcome.  
Welcome.  
Welcome.  
Welcome.  
Welcome.

- In the fourth iteration, the value of the variable i becomes 4, the fourth number of the list, and the print (...) sentence is executed.
- In the fifth iteration, the value of the variable i becomes 5, the fifth number in the list, and the print (...) sentence is executed.

## 1. Structure of Loop Statement

### 1.4. Sequence object that follows for – in

A sequence object such as a list or a string may also follow for – in.

- ▶ An object that can have multiple items such as a list or a string is called a sequence object.
- ▶ In the previous slide, the value of the variable i was not used at all.
- ▶ This time, assign the items of the sequence object to the variable i inside the loop statement. Then, use the print function to print the value of i.

```
1 for i in [1, 2, 3, 4, 5]: # place the list after in and : after the list
2     print("i =", i) # print values of i
```

```
i = 1
i = 2
i = 3
i = 4
i = 5
```

## 1. Structure of Loop Statement

### 1.4. Sequence object that follows for - in

This statement can also be applied to the following strings.

```
1 for i in "Hello":      # : is necessary at the end
2     print("i =", i)    # print i values
```

```
i = H
i = e
i = l
i = l
i = o
```

## 1. Structure of Loop Statement

### 1.5. Range function that comes with the for statement

You can write a code that prints "Welcome to" and "Python Corporation!" five times with the for statement as follows. (2) is the iterated block.

```
1 for i in range(5):          # (1)
2     print('Welcome to')      # (2)
3     print("Python corporation!") # (2)
```

```
Welcome to
Python corporation!
```

## 1. Structure of Loop Statement

### 1.6. Characteristics of the range function

- range(5) function returns a **sequence** of 0, 1, 2, 3, and 4. In each iteration, it assigns these values to the variable i. That is, i becomes 0 in the first iteration and 1 in the second iteration. In the last iteration, i becomes 4.
- You can see integers generated by the range function if you apply a list function to the range function. In this way, the range function produces consecutive values.

```
1 print(list(range(5)))
```

```
[0, 1, 2, 3, 4]
```



## One More Step

- | Values returned by a range function are of range type.
  - ▶ Python's range function returns range-type data type.
  - ▶ range-type data type generates and returns consecutive values for each call. / Therefore, it is mainly used with for statements.

## 1. Structure of Loop Statement

### 1.6. Characteristics of the range function

The range function can assign the step value.

- ▶ The range function can generate more diverse integer sequences utilizing multiple arguments, and its general format is as follows.
- ▶ If you call with the format range (start, stop, step), integers from start to (stop-1) are generated with step intervals.
- ▶ The for loop is iterated as many times as the number of this sequence.
- ▶ Here, start and step may be omitted, in which case the start is 0 and the step is 1.
- ▶ However, the loop is executed **only when the stop value is specified**.

value that ends generation, but this  
stop is excluded from generation

Start value of generation

The size of the value that  
increases each time

```
range(start = 0, stop, step = 1)
```

If start and step are not specified,  
they are treated as 0 and 1,  
respectively.  
The value that must be specified is  
stop.

## 1. Structure of Loop Statement

### 1.7. range function examples

- | range(0, 5, 1) returns integers 0, 1, 2, 3, and 4. range(5) is identical to range(0, 5, 1) since start and step are omitted.
- | If you want to iterate from 1 (not 0) to 5, use range(1, 6, 1). Print the values of variable i as it is being iterated.

```
1 for i in range(1, 6, 1):
2     print(i, end=" ") # if you assign end = " ", it enumerates with spaces in between instead of changing lines.
```

1 2 3 4 5

- | To print odd numbers between 1 and 10, assign 2 to the step value.

```
1 for i in range(1, 10, 2):
2     print(i, end=" ")
```

1 3 5 7 9

## 1. Structure of Loop Statement

### 1.7. range function examples

- If you wish to print numbers from 1 to 10 in descending order, use range(10, 0, -10). Print the values of variable i as it is being iterated.

```
1 for i in range(10, 0, -1):
2     print(i, end=" ")
```

10 9 8 7 6 5 4 3 2 1



## One More Step

I range(n) is identical to range(0, n, 1).

- ▶ One is prone to think that range(10) function generates integers from 1 to 10. The number of iterations are 10. However, the integers produced are from 0 to 9. This has been a long-standing debate in computing.
- ▶ Starting from 0 has now become a trend. Anyhow, range(10) iterates 10 times and generates integers from 0 to 9. To print integers from 1 to 10, use range(1, 11).

## 2. format function

### 2.1. format function code example

| {} format function is used to format strings and numbers in a specified style.

```
1 # formatting integer value
2 print('{:5d}'.format(100))
```

100

```
1 # formatting integer value
2 print('{:5d}'.format(10))
```

10

```
1 # formatting real number value
2 print('{:6.3f}'.format(3.14))
```

3.140

```
1 # formatting real number value
2 print('{:6.2f}'.format(3.14))
```

3.14

- ▶ '{}.format(100) executes formatting function which prints 100.
- ▶ '{:5d}'.format(100), '{:5d}'.format(10) creates 5 spaces to print 100 and 10.
- ▶ '{:6.2f}'.format(3.14), creates 6 spaces and prints 2 digits below the decimal point in order to print the number 3.14.

# | Pair programming



# Pair Programming Practice

## Guideline, mechanisms & contingency plan

Preparing pair programming involves establishing guidelines and mechanisms to help students pair properly and to keep them paired. For example, students should take turns “driving the mouse.” Effective preparation requires contingency plans in case one partner is absent or decides not to participate for one reason or another. In these cases, it is important to make it clear that the active student will not be punished because the pairing did not work well.

## Pairing similar, not necessarily equal, abilities as partners

Pair programming can be effective when students of similar, though not necessarily equal, abilities are paired as partners. Pairing mismatched students often can lead to unbalanced participation. Teachers must emphasize that pair programming is not a “divide-and-conquer” strategy, but rather a true collaborative effort in every endeavor for the entire project. Teachers should avoid pairing very weak students with very strong students.

## Motivate students by offering extra incentives

Offering extra incentives can help motivate students to pair, especially with advanced students. Some teachers have found it helpful to require students to pair for only one or two assignments.



# Pair Programming Practice

## | Prevent collaboration cheating

The challenge for the teacher is to find ways to assess individual outcomes, while leveraging the benefits of collaboration. How do you know whether a student learned or cheated? Experts recommend revisiting course design and assessment, as well as explicitly and concretely discussing with the students on behaviors that will be interpreted as cheating. Experts encourage teachers to make assignments meaningful to students and to explain the value of what students will learn by completing them.

## | Collaborative learning environment

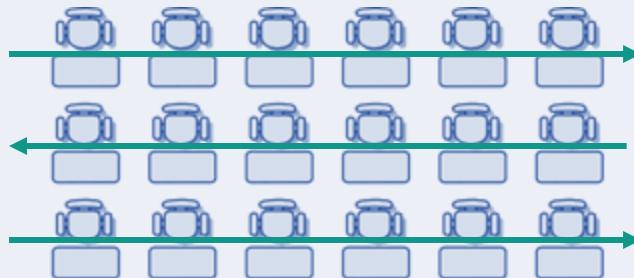
A collaborative learning environment occurs anytime an instructor requires students to work together on learning activities. Collaborative learning environments can involve both formal and informal activities and may or may not include direct assessment. For example, pairs of students work on programming assignments; small groups of students discuss possible answers to a professor's question during lecture; and students work together outside of class to learn new concepts. Collaborative learning is distinct from projects where students "divide and conquer." When students divide the work, each is responsible for only part of the problem solving and there are very limited opportunities for working through problems with others. In collaborative environments, students are engaged in intellectual talk with each other.

**Q1.**

Agency A is planning to issue tickets of a concert hall for idol singers' concert. Here, the number  $n$  is the input and the seat number is arranged as follows.  $n * n$  seats are placed when  $n$  is given as input. The below arrangement of the seat numbers is called a snake matrix because the array increases in ones in the shape of a snake's trunk. Write a program that produces arrays of these numbers.

Output Example

```
Enter n : 5
 1  2  3  4  5
10  9  8  7  6
11  12 13 14 15
20  19 18 17 16
21  22 23 24 25
```



Unit 9.

# Loop-2

## Learning objectives

- ✓ Be able to define while repetition and use the statement to write repeated sentences according to conditions.
- ✓ Be able to import random number module to use various random number functions inside.
- ✓ Be able to stop or control loop statements using break and continue.
- ✓ Be able to solve more diverse and complex problems than previous problems using a double loop.

## Lesson overview

- ✓ Learn the necessity while statement and how to use it.
- ✓ Learn how to generate random numbers and apply the values to the program.
- ✓ Learn how to give various conditions to the while loop.
- ✓ Learn how to write a program that performs repetitive multiplication using double statement.

## Concepts You Will Need to Know From Previous Units

- ✓ Concept of loop statement and be able to use it for repetitive command.
- ✓ Able to write a program using for in expression and list.
- ✓ Differences between sequential statement, conditional statement, and loop statement.

# Keywords

while

random

import  
command

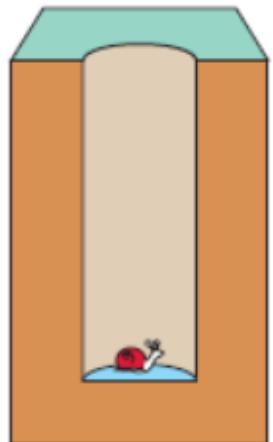
break, continue

double loop

# Mission

## 2. Solution

### 2.1. Snail in a Well



- ▶ Suppose there is a well with a depth of 30 meters deep. And a snail is at the bottom of this well. Snails living in this well can climb 7 meters if they crawl up all day. However, while taking a rest at night, it slides down 5 meters. How many days will it take to get out of this 30-meter-deep well?
- ▶ After going up 7 meters a day and going down 5 meters, it will end up moving 2 meters, but it is a problem that you should not simply think that it will take 15 days to move 30 meters. The answer is 13 days, but it will be confirmed through coding.
- ▶ Therefore, to solve this problem, we will use a new statement called the while statement, not for statement. In this mission, we will use the while statement to find the location of the snail every evening and calculate the date of exit from the 30-meter well.

<https://puzzlefry.com/puzzles/snail-in-well-riddle/>

### 3. Mission

#### 3.1. How Snail in a Well works

```
1 current_pos = 0 # Current position of snail
2 days = 1 # date
3 move_up = 7 # distance it goes up a day : in meters
4 move_down = 5 # distance it goes down while sleep : in meters
5 destination = 30 # Distance goal : in meters
6 current_pos += move_up # position of the snail on the first day
7
8 print('day :',days, 'Snail's position :', current_pos, 'meters')
9 while( current_pos <= destination): # repeat if snail hasn't reached the distance goal
10    current_pos -= move_down # goes down again
11    days += 1 # next day
12    current_pos += move_up # distance it went up
13    print('day :',days, 'snail's position :', current_pos, 'meters')
14 print()
15 print('It took {} days to get out of the well.'.format(days))
```

```
day : 1  Snail's position : 7 meters
day : 2  Snail's position : 9 meters
day : 3  Snail's position : 11 meters
day : 4  Snail's position : 13 meters
day : 5  Snail's position : 15 meters
day : 6  Snail's position : 17 meters
day : 7  Snail's position : 19 meters
day : 8  Snail's position : 21 meters
day : 9  Snail's position : 23 meters
day : 10  Snail's position : 25 meters
day : 11  Snail's position : 27 meters
day : 12  Snail's position : 29 meters
day : 13  Snail's position : 31 meters
```

It took 13 days to get out of the well.

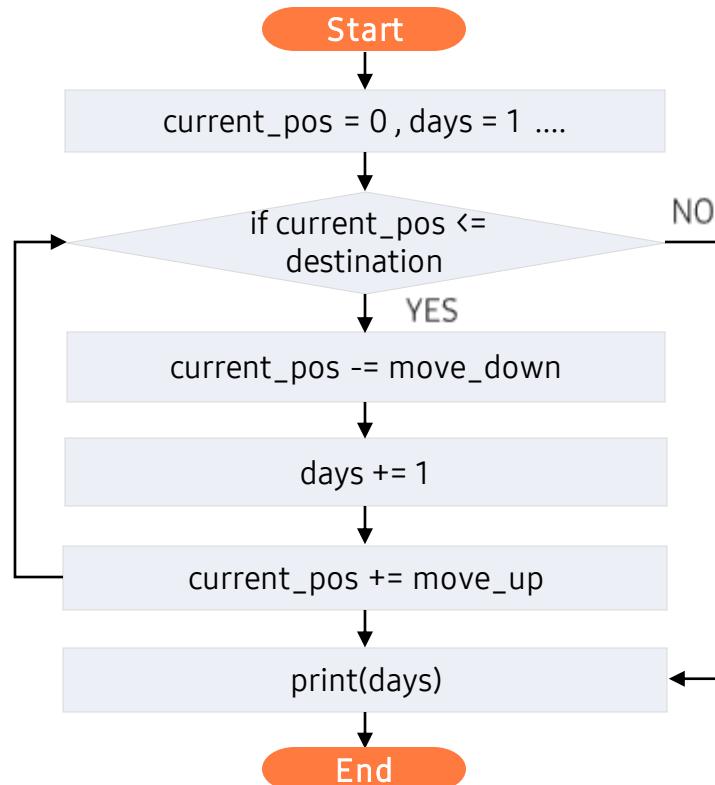
### 3. Mission

#### 3.2. Programming Plan

##### Pseudocode

- [1] Start
- [2] Initialize the location, date, target location, daily travel distance, etc. of snail.
- [3] while If the snail's position is smaller than the target position
  - [4] goes down while sleep
  - [5] Next day. Increase the day every time
  - [6] Goes up again..  
end while
- [7] Print the day it took to escape.
- [8] End

##### Flowchart



### 3. Mission

#### 3.3. Snail in a Well final code

```
1 current_pos = 0 # Current position of snail
2 days = 1          # date
3 move_up = 7       # distance it goes up a day : in meters
4 move_down = 5     # distance it goes down while sleep : in meters
5 destination = 30 # Distance goal : in meters
6 current_pos += move_up # position of the snail on the first day
7
8 print('day :',days, 'Snail's position :', current_pos, 'meters')
9 while( current_pos <= destination): # repeat if snail hasn't reached the distance goal
10    current_pos -= move_down        # goes down again
11    days += 1                      # next day
12    current_pos += move_up         # distance it went up
13    print('day :',days, 'snail's position :', current_pos, 'meters')
14 print()
15 print('It took {} days to get out of the well.'.format(days))
```

# | Key concept

## 1. Basic Structure of while Statement

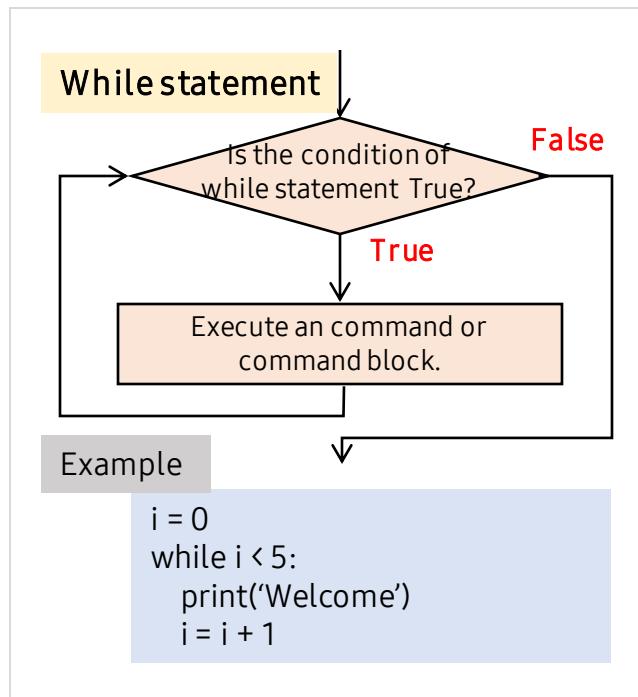
### 1.1. What is while statement?

- | The while statement repeatedly performs the sentence like the for statement.
- | The for statement ends after taking out all the sequences following the in keyword.
- | The while statement repeats the instruction only if a particular condition is true.
  - ▶ It is in form of while *[Conditional expression]*: .
  - ▶ If write as while True : , it goes into infinite loop.
- | If the condition continues to be true, this loop may not end, so careful when writing the code.

## 1. Basic Structure of while Statement

### 1.1. What is while statement?

- In the previous unit, we printed 'Welcome' five times using the for statement. Now let's find out how to use the while statement.
- The meaning of this flowchart and code is as shown below.



- Initialize i to zero.
- The while statement prints 'Welcome' and increases the i value by 1 if the condition of  $i < 5$  is true.
- Since the initial i value is 0, this iteration will also be performed five times.

## 1. Basic Structure of while Statement

### 1.2. Syntax of while statement

Form	Example
set initial value while conditional statement : code block to be excuted	i = 0            # Set initial value while i<5 :    # conditional expression print('Welcome to everyone!!') i += 1

- ▶ Despite it is loop statement, it is very similar to the if statement.
- ▶ If the conditional expression is true, it repeats and executes the code.

## 1. Basic Structure of while Statement

### 1.2. Syntax of while statement

```
1 i = 0 # initial value
2 while i < 5:    # block is executed when loop's condition is true
3     print('Welcome to everyone!!')
4     i += 1
```

Welcome to everyone!!  
Welcome to everyone!!  
Welcome to everyone!!  
Welcome to everyone!!  
Welcome to everyone!!

- ▶ Increase i by 1 from 0 and repeat the code inside while only if it is less than 5.

## 1. Basic Structure of while Statement

### 1.3. A program to find the sum of values up to the input value

| A program that uses a while statement to find the sum of values up to the input value.

```
1 n = int(input('Enter a number to sum up to : '))
2 s = 0
3 i = 1
4 while i <= n:
5     s = s + i
6     i += 1
7 print('Sum of 1 to {} is {}'.format(n, s))
```

Enter a number to sum up to : 5

Sum of 1 to 5 is 15

- ▶ The above code receives 5 as an input and calculates and prints the sum of integers from 1 to 5.
- ▶ In this way, if the **number of iterations is clear**, it is better to use the for statement because the code of the while statement becomes longer.
- ▶ So when should we use the while statement?

## 1. Basic Structure of while Statement

### 1.4. Comparing while statement and for statement

| Compare while statement and for statement using example code.

for

```
1 s = 0
2 for i in range(1, 11):
3     s += i
4 print('Sum of 1 to 10:' .s)
```

while

```
1 n = 10
2 s = 0
3 i = 1
4 while i <= n:
5     s = s + i
6     i += 1
7 print('Sum of 1 to {} is {}'.format(n, s))
```

- ▶ The while statement is more suitable when the performance condition is clear, although the number of iteration is not known exactly.
- ▶ If the number of repetitions is clear, the for statement is appropriate.

## 1. Basic Structure of while Statement

### 1.5. while loop and input condition

I Receive only 'scissors', 'rock', and 'paper' as inputs from the user and print the value. If inputs other than 'scissors', 'rock', and 'paper' come in, ask for the input again.

```
1 selected = None
2 while selected not in ['scissors', 'rock', 'paper']:
3     selected = input('Choose among scissors, rock, paper> ')
4 print('Chosen value:', selected)
```

Choose among scissors, rock, paper> no  
Choose among scissors, rock, paper> why  
Choose among scissors, rock, paper> paper  
Chosen value: paper

- ▶ In the case of such a code, it is difficult to solve the problem with for statement.
- ▶ While is suitable for repeated statements that are executed only when the desired value is entered.



## One More Step

- ▶ In the code discussed earlier, there is expression selected = None.
- ▶ This code is being input from Line 3 after inspecting the selected from the while statement in Line 2.
- ▶ If the value None is not assigned to selected by selected = None, this code causes an error.
- ▶ In other words, None means 'no value', but in this way, it does important things in the flow of the program.

```
1 selected = None
2 while selected not in ['scissors', 'rock', 'paper']:
3     selected = input('Choose among scissors, rock, paper> ')
4 print('Chosen value:', selected)
```

Choose among scissors, rock, paper>

## 1. Basic Structure of while Statement

### 1.5. while loop and input condition

| Receive a positive integer n from the user and calculate the sum  $1 + 2 + \dots + n$ . However, if the user's input is -10, the program falls into a logical error. Therefore, the input value must be a positive value.

```
1 n = int(input('Enter a number to sum up to :'))
2 s = 0
3 for i in range(1, n+1) :
4     s = s + i
5 print('Sum of 1 to {} is {}'.format(n, s))
```

Enter a number to sum up to : -10  
Sum of 1 to -10 is 0

- ▶ This code has a logical error.
- ▶ It would be better to limit the range of input values to positive integers. In this case, it is more appropriate to use a while statement that is repeatedly executed according to the conditions than a for statement.

## 1. Basic Structure of while Statement

### 1.5. while loop and input condition

I Receive a positive integer n from the user and calculate the sum  $1 + 2 + \dots + n$ . The while statement is appropriate when you want to receive positive n.

```
1 n = -1
2 while n <= 0: # input() repeated until positive value is entered
3     n = int(input('Enter a positive number to sum up to :'))
4 s = 0
5 for i in range(1, n+1):
6     s = s + i
7 print('Sum of 1 to {} is {}'.format(n, s))
8
```

```
Enter a positive number to sum up to : -1
Enter a positive number to sum up to : 0
Enter a positive number to sum up to : 7
Sum of 1 to 7 is 28
```



Line2

- The key is to receive re-input if the condition  $n \leq 0$  is satisfied. If -1 or 0 is entered, the input is repeatedly requested.

## 2. random

### 2.1. random module

- | Let's take a closer look at the random module we saw in the previous unit.
- | First, look at the concept and syntax of modules.
- | A module refers to a script file in which Python functions, variables, or classes are collected.
  - ▶ Python has many modules made by several organizations and developers, and we can effectively develop software using them.
  - ▶ The module developed in this way is provided with a Python installation file under the name of a standard library.
- | When bringing in a pre-made module, write the module name with "import", and when using it, mark a dot (.) in the module name and call the components in the module.

```
import moduleName1[, moduleName2, ...]
```

## 2. random

### 2.2. Details of the features of the random module.

- | The random module includes a function of generating an arbitrary number or randomly mixing or selecting elements in the list.
- | For convenience, random modules are used as abbreviations rd. Use the as command as following.

```
import [module name]as [abbreviation]
```

- | The functions and tasks of the random module are as following.

function	tasks
random()	It generates a real number between 0 and 1. (not including 1)
randrange()	Returns an integer within the specified range.
randint(a, b)	Returns a random integer N between a <=N <=b.
shuffle(seq)	Randomly mix elements from a given seq list.
choice(seq)	Select any element in the seq sequence.
sample()	Randomly select a specified number of elements.

## 2. random

### 2.3. Functions included in random module

```
1 import random as rd      # import command that calls inner module and abbreviation of random module, rd
2
3 print(rd.random())       # Returns a random real
4 print(rd.random())       # Returns a random real
5 print(rd.randrange(1, 7)) # Returns an integer greater than or equal to 1 and less than 7.
6 print(rd.randrange(0, 10, 2))# Returns a multiple of 2 of integers greater than or equal to 0 and less than 10.
7 print(rd.randint(1, 10))  # Returns an arbitrary integer greater than or equal to 1 and less than
                           # or equal to 10 (including 10).
```

0.4753236338712161

0.4515339413828623

5

2

1

- ▶ random(): Returns a real number greater than or equal to 0 and less than 1, and returns a different real number each time it is used.
- ▶ range 1 and 7 : returns an integer greater than or equal to 1 and less than 7.
- ▶ range (0, 10, 2) : Returns a multiple of 2 of integers greater than or equal to 0 and less than 10.
- ▶ randint (1, 10) : returns an arbitrary integer greater than or equal to 1 and less than or equal to 10 (including 1 and 10).

## 2. random

### 2.3. Functions included in random module

```
1 numlist = [10, 20, 30, 40, 50]
2 rd.shuffle(numlist)
3 print(numlist)
```

[30, 10, 20, 40, 50]

```
1 print(rd.choice(numlist))
```

40

```
1 print(rd.sample(numlist, 3))
```

[20, 30, 10]

- ▶ shuffle() : Randomly mixes elements of the sequence and returns them.
- ▶ choice(): Extract any element from the sequence that enters the factor.
- ▶ sample() : Randomly returns the element. At this time, the number of elements to be returned can be added as a factor.

## 2. random

### 2.3. Functions included in random module

```
1 a = list(range(1, 11)) # returns integers from 1 to 10
2 rd.shuffle(a)          # randomly mix elements of list
3 print(a)
```

```
[9, 5, 3, 2, 8, 10, 7, 4, 1, 6]
```

- ▶ shuffle() returns the sequence in a different order each time.

## 2. random

### 2.4. sample function

| Print three of the integers between 1 and 10 using a sample function.

```
1 import random as rd
2
3 a = list(range(1, 11))
4 b = rd.sample(a, 3)
5 print('Arbitrary integer between 1 to 10 :', b)
```

Arbitrary integer between 1 to 10 : [2, 5, 10]

#### Line 3~4

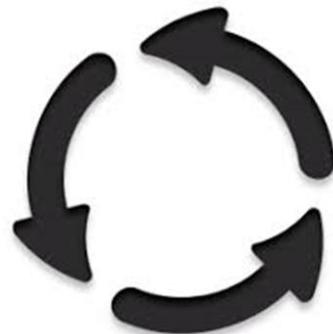
- 3:a is assigned with list with elements 1 to 10.
- 4: Samples three arbitrary integers using the sample function and put them in b.

## 3. break, continue

### 3.1. Keyword that controls loop statement

I break and continue are keywords that control loop statement

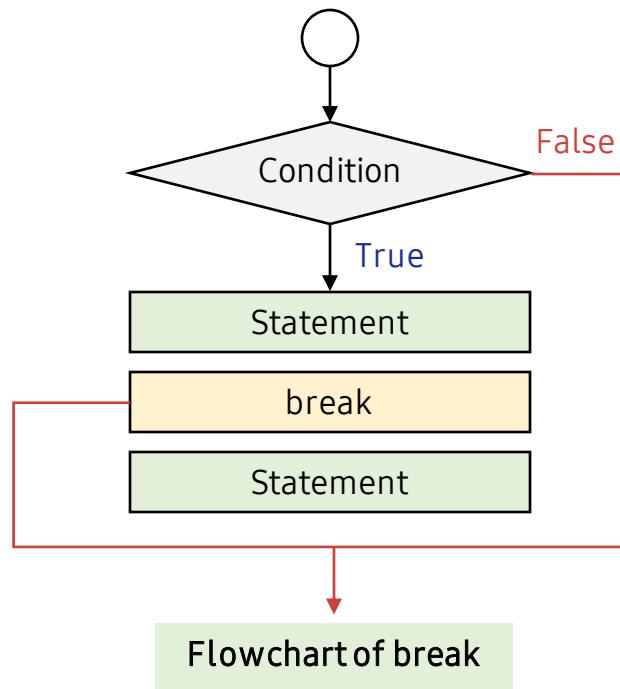
- ▶ To end the loop: break
- ▶ To skip the remaining execution parts within the iteration loop and continue to execute the loop : continue
- ▶ continue does not end the loop.



## 3. break, continue

### 3.2. Flowchart of break

- I while and loop statements perform statements within a block if the condition is true.
- I If it comes to break in the middle, it immediately end and exit the loop.



### 3. break, continue

#### 3.3. break statement example

```
1 st = 'Programming'  
2 # function of executing only when it is consonant  
3 for ch in st:  
4     if ch in ['a','e','i','o','u']:  
5         break    # end the loop if it is vowel  
6     print(ch)  
7 print('The end')
```

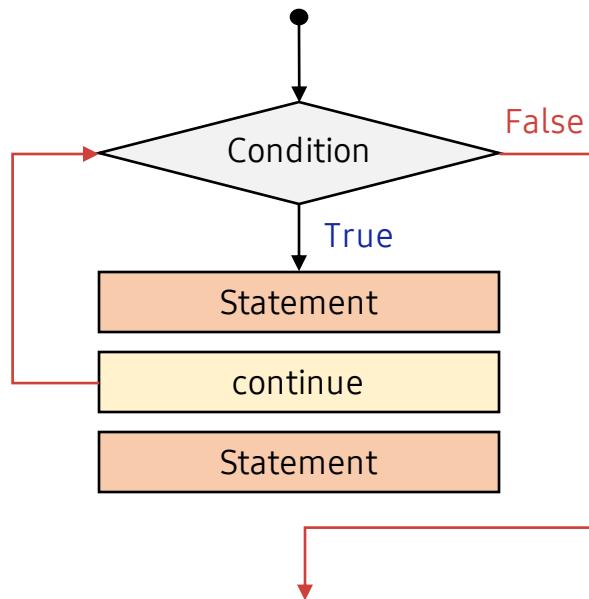
P  
r  
The end

- ▶ break if it is a vowel, otherwise print using print.
- ▶ When the break is activated, the loop is stopped without executing the rest of the loop.

## 3. break, continue

### 3.4. Flowchart of continue

- | It plays a role of skipping only the sentences below continue without exiting the loop.
- | The termination of the loop is only applicable when the condition is false.



### 3. break, continue

#### 3.5. continue statement example

```
1 st = 'Programming'  
2 # function of executing only when it is consonant  
3 for ch in st:  
4     if ch in ['a','e','i','o','u']:  
5         continue    # skip the execution below if it is vowel  
6     print(ch)  
7 print('The end')
```

P  
r  
g  
r  
m  
m  
n  
g  
The end

- ▶ When you write continue, it does not execute the rest of the code below.
- ▶ In other words, it returns to the beginning of the loop without executing the sixth line's print statement.

## 3. break, continue

### 3.6. Risk of break and continue flow control

- | Break and continue can be conveniently used to efficiently control programs.
- | However, if break and continue statements are used too much, it becomes difficult to understand the program due to inconsistent flow of control. Therefore, it is recommended to use continue and break only when necessary.

## 4. Double loop

### 4.1. Double loop

- | Double loop refers to a structure in which other loops are contained inside a loop.
- | The nested for loop below contains a for statement in the for statement.
- | This code is a multiplication of 2x1 to 9x9.
- | Looking at the structure of the multiplication table, there are 2 to 9 stages, and 1 to 9 are multiplied by each stage and printed on the screen.
- | In order to implement this, a double for statement is needed to put the for statement back in the for statement.

```
1 for i in range(2, 10):          # outer for loop
2     for j in range(1, 10):      #inner for loop
3         print('{0}{1} = {2:2d}, '.format(i, j, i*j), end = ' ')
4     print()    # executes inner loop and change line
```

```
2x1 = 2, 2x2 = 4, 2x3 = 6, 2x4 = 8, 2x5 = 10, 2x6 = 12, 2x7 = 14, 2x8 = 16, 2x9 = 18,
3x1 = 3, 3x2 = 6, 3x3 = 9, 3x4 = 12, 3x5 = 15, 3x6 = 18, 3x7 = 21, 3x8 = 24, 3x9 = 27,
4x1 = 4, 4x2 = 8, 4x3 = 12, 4x4 = 16, 4x5 = 20, 4x6 = 24, 4x7 = 28, 4x8 = 32, 4x9 = 36,
5x1 = 5, 5x2 = 10, 5x3 = 15, 5x4 = 20, 5x5 = 25, 5x6 = 30, 5x7 = 35, 5x8 = 40, 5x9 = 45,
6x1 = 6, 6x2 = 12, 6x3 = 18, 6x4 = 24, 6x5 = 30, 6x6 = 36, 6x7 = 42, 6x8 = 48, 6x9 = 54,
7x1 = 7, 7x2 = 14, 7x3 = 21, 7x4 = 28, 7x5 = 35, 7x6 = 42, 7x7 = 49, 7x8 = 56, 7x9 = 63,
8x1 = 8, 8x2 = 16, 8x3 = 24, 8x4 = 32, 8x5 = 40, 8x6 = 48, 8x7 = 56, 8x8 = 64, 8x9 = 72,
9x1 = 9, 9x2 = 18, 9x3 = 27, 9x4 = 36, 9x5 = 45, 9x6 = 54, 9x7 = 63, 9x8 = 72, 9x9 = 81,
```

## 4. Double loop

### 4.2. Structure of double loop

| Double loop : When the loops overlap, the two loops are divided into terms such as outer loop and inner loop.

```
Outer loop   for i in range(2, 10):
              Inner loop   for j in range(1, 10):
                           print('{}*{}={:2d}'.format(i, j, i*j), end=' ')
                           print()
```

- ▶ The double for loop has an inner loop and an outer loop.
- ▶ {} in the print statement is a placeholder that specifies a place for output, the content covered by Unit 8.

## 4. Double loop

### 4.3. Execution of double loop

When i = 2	When i = 3	When i = 4	...	When i = 9
j = 1 : 2 * 1	j = 1 : 3 * 1	j = 1 : 4 * 1	...	j = 1 : 9 * 1
j = 2 : 2 * 2	j = 2 : 3 * 2	j = 2 : 4 * 2	...	j = 2 : 9 * 2
j = 3 : 2 * 3	j = 3 : 3 * 3	j = 3 : 4 * 3	...	j = 3 : 9 * 3
j = 4 : 2 * 4	j = 4 : 3 * 4	j = 4 : 4 * 4	...	j = 4 : 9 * 4
j = 5 : 2 * 5	j = 5 : 3 * 5	j = 5 : 4 * 5	...	j = 5 : 9 * 5
j = 6 : 2 * 6	j = 6 : 3 * 6	j = 6 : 4 * 6	...	j = 6 : 9 * 6
j = 7 : 2 * 7	j = 7 : 3 * 7	j = 7 : 4 * 7	...	j = 7 : 9 * 7
j = 8 : 2 * 8	j = 8 : 3 * 8	j = 8 : 4 * 8	...	j = 8 : 9 * 8
j = 9 : 2 * 9	j = 9 : 3 * 9	j = 9 : 4 * 9	...	j = 9 : 9 * 9

- ▶ i has a value from 2 to 9.
- ▶ The inner j has a value from 1 to 9.
- ▶ This structure is repeated, resulting in 72 calculations and outputs in total.
- ▶ In the case of a double for loop or triple for loop, it becomes difficult to understand the code.
- ▶ For this reason, nested loops do not use structures more than triple loops.

# | Paper coding

**Try to fully understand the basic concept before moving on to the next step.**

**Lack of understanding basic concepts will increase your burden in learning this course, which may make you fail the course.**

**It may be difficult now, but for successful completion of this course we suggest you to fully understand the concept and move on to the next step.**

## Q1.

Write a program that prints multiplication of 2 using the while statement as following.

### Example Output

```
2 * 1 = 2  
2 * 2 = 4  
2 * 3 = 6  
2 * 4 = 8  
2 * 5 = 10  
2 * 6 = 12  
2 * 7 = 14  
2 * 8 = 16  
2 * 9 = 18
```

### Time

5min



Write the entire code and the expected output results in the note.

**Q2.** Let's modify the above program to print all the stages 1 to 9 of the multiplication table. Use only the while statement.

## Example Output

```
1*1=1  
1*2=2  
1*3=3  
1*4=4  
1*5=5  
1*6=6  
1*7=7  
1*8=8  
1*9=9  
2*1=2  
2*2=4  
2*3=6  
2*4=8
```

## Time

5min



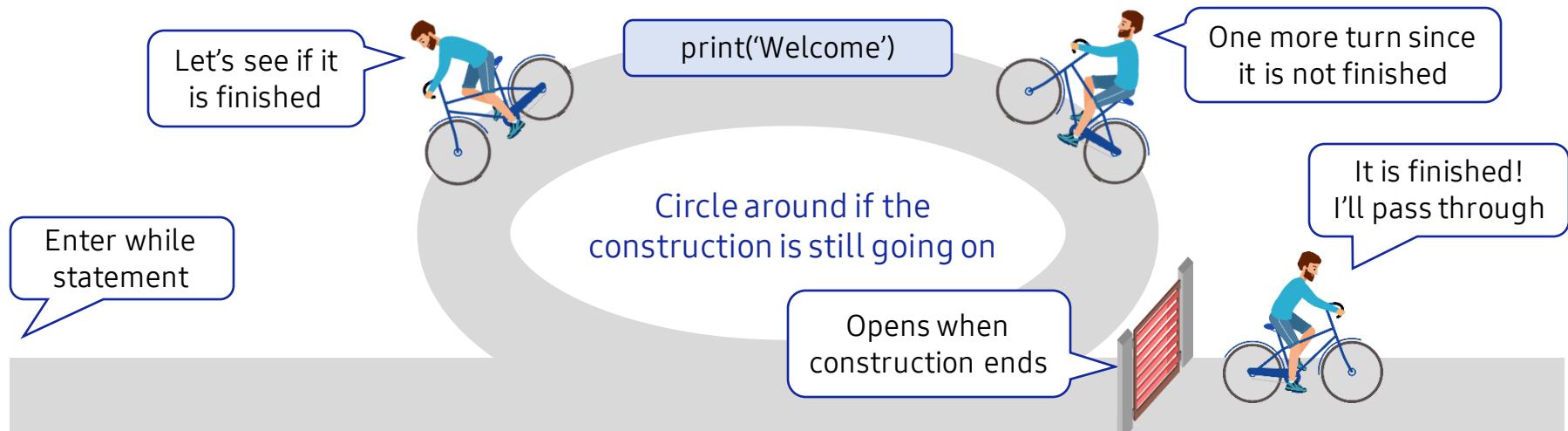
Write the entire code and the expected output results in the note.

| Let's code

## 1. while Statement

### 1.1. while statement that is repeatedly executed according to the conditions

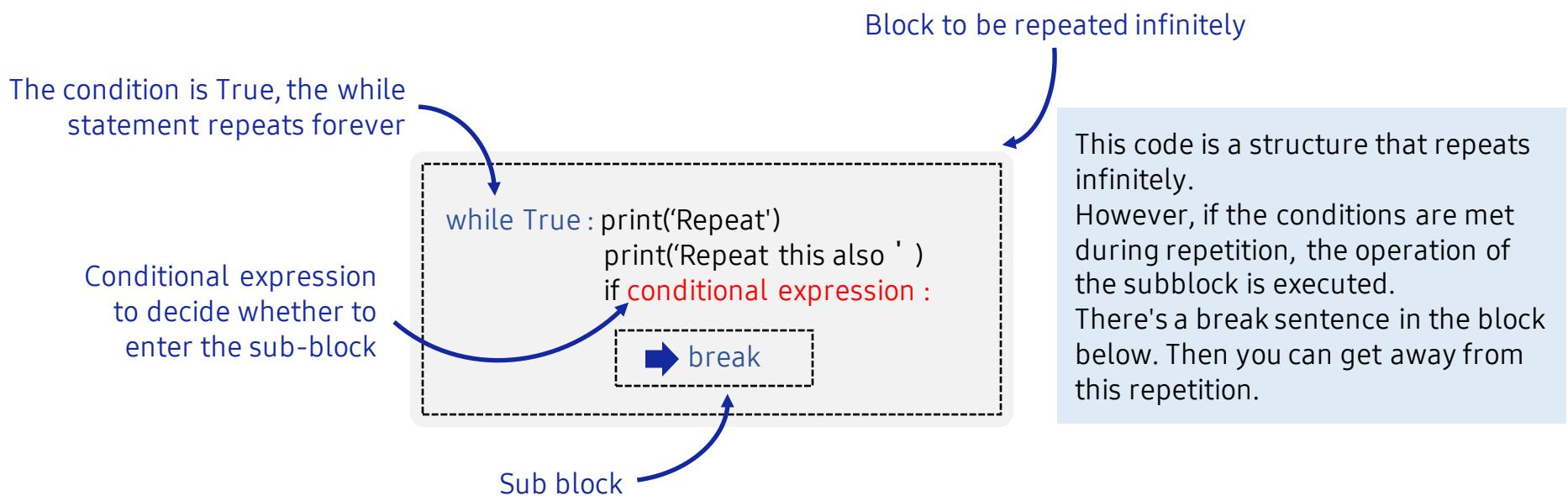
- Condition control repetition is the name given because it repeats while a condition is satisfied.
- For example, you were riding a bicycle, but construction took place in front of you, so you couldn't proceed. While construction is underway, you repeat the same path over and over again.
- Each round of this bicycle checks if construction is complete. If it is under construction, it continues to repeat and escapes after construction is completed.
- In this case, conditional control repetition is used. It is convenient to use a while loop for conditional control repetition.



## 1. while Statement

### 1.1. while statement that is repeatedly executed according to the conditions

The infinite loop has the following structure. Exit infinite loop using break.



## 1. while Statement

### 1.1. while statement that is repeatedly executed according to the conditions

- | It is a sentence that asks, "Is the construction completed?" and receives "Yes" or "No" and repeats until "Yes" is entered.
- | The conditions of repetition are clear, but the exact number of repetitions is not known. In this case, the while statement is better than the for statement.

```
1 under_construction = True
2 while under_construction:
3     response = input("Is the construction completed?");
4     if response == "Yes" :
5         under_construction = False
6
7 print("Escaped successfully.")
```

Is the construction completed? No  
Is the construction completed? Yes  
Escaped successfully.

## 1. while Statement

### 1.2. while statement that is repeatedly executed according to the conditions example

- | In which case can we not know the number of repetitions? The most representative case is when a value is received from a user and processed. This is because it is difficult to predict the value input by the user.
- | For example, let's say the user enters a password and checks the program for the password. The code we write continues with the question until the user enters the correct password.
- | Suppose the password is "pythonisfun". Enter the wrong password aaa as following. Check whether this password is correct and repeat the question if not.
- | When the password "pythonisfun" is entered properly, it exits the while loop and displays "\*\* Login success \*\*".

```
1 password = ""  
2 while password != "pythonisfun":  
3     password = input("Enter the password: ")  
4 print("** Login success **")
```

```
Enter the password: aaaa  
Enter the password: pythonisfun  
** Login success **
```

## 1. while Statement

### 1.3. Use while for a certain number of repetitions

- | The while loop is not only available if the number of times is unknown. Even if you know the number of times, you can use a while loop.
- | For example, write a code that calculates the sum of 1 to 10 in a while loop. To this end, initialize the variable count increasing from 1 to 10.
- | Declare the variable s to store these values and initialize them to zero. The code for adding s to the count could be written as following.

```
1 count = 1
2 s = 0 # s is variable that contains values accumulated and added, reset it as 0
3 while count <= 10 :
4     s = s + count # add count value to s each time
5     count = count + 1 # increase count value by 1 every time
6 print("Sum is", s)
```

Sum is 55

## 1. while Statement

### 1.4. Calculate the sum of the numbers input by the user

```
1 total = 0
2 answer = 'yes'
3 while answer == 'yes':
4     number = int(input('Enter the number: '))
5     total = total + number
6     answer = input('Continue?(yes/no): ')
7 print('Sum is : ', total)
```

```
Enter a number: 5
Continue?(yes/no): yes
Enter a number: 6
Continue?(yes/no): no
Sum is : 11
```

#### Line 2~3

- Let answer be initialized to the string yes.
- Line 3: while condition is true, so the block right below is executed.

## 1. while Statement

### 1.4. Calculate the sum of the numbers input by the user

```
1 total = 0
2 answer = 'yes'
3 while answer == 'yes':
4     number = int(input('Enter the number: '))
5     total = total + number
6     answer = input('Continue?(yes/no): ')
7 print('Sum is : ', total)
```

```
Enter a number: 5
Continue?(yes/no): yes
Enter a number: 6
Continue?(yes/no): no
Sum is : 11
```

#### Line 4~6

- A number is assigned in the number variable.
- Add the value to the total by accumulating it.
- When the value 'yes' is added to the answer variable, the loop repeats, and when 'no' is entered, the loop exits.

# | Pair programming



# Pair Programming Practice

## Guideline, mechanisms & contingency plan

Preparing pair programming involves establishing guidelines and mechanisms to help students pair properly and to keep them paired. For example, students should take turns “driving the mouse.” Effective preparation requires contingency plans in case one partner is absent or decides not to participate for one reason or another. In these cases, it is important to make it clear that the active student will not be punished because the pairing did not work well.

## Pairing similar, not necessarily equal, abilities as partners

Pair programming can be effective when students of similar, though not necessarily equal, abilities are paired as partners. Pairing mismatched students often can lead to unbalanced participation. Teachers must emphasize that pair programming is not a “divide-and-conquer” strategy, but rather a true collaborative effort in every endeavor for the entire project. Teachers should avoid pairing very weak students with very strong students.

## Motivate students by offering extra incentives

Offering extra incentives can help motivate students to pair, especially with advanced students. Some teachers have found it helpful to require students to pair for only one or two assignments.



# Pair Programming Practice



## | Prevent collaboration cheating

The challenge for the teacher is to find ways to assess individual outcomes, while leveraging the benefits of collaboration. How do you know whether a student learned or cheated? Experts recommend revisiting course design and assessment, as well as explicitly and concretely discussing with the students on behaviors that will be interpreted as cheating. Experts encourage teachers to make assignments meaningful to students and to explain the value of what students will learn by completing them.

## | Collaborative learning environment

A collaborative learning environment occurs anytime an instructor requires students to work together on learning activities. Collaborative learning environments can involve both formal and informal activities and may or may not include direct assessment. For example, pairs of students work on programming assignments; small groups of students discuss possible answers to a professor's question during lecture; and students work together outside of class to learn new concepts. Collaborative learning is distinct from projects where students "divide and conquer." When students divide the work, each is responsible for only part of the problem solving and there are very limited opportunities for working through problems with others. In collaborative environments, students are engaged in intellectual talk with each other.

## Q1.

A palindrome number refers to an integer whose value is the same as its original value, even if listed upside down, such as 121 or 3443. Write the following program to determine whether the number is a palindrome number or not by receiving the number n from the user.

### Output Example

```
Enter an integer: 135
```

```
135 is not a palindrome number
```

```
Enter an integer: 3443
```

```
135 is a palindrome number
```

## Q2.

The computer has a random integer between 1 and 100 as the correct answer value as following. When the user presents the correct answer, the program only informs whether the presented integer is higher or lower compared to the correct answer he or she stored. This game is repeated until the user answers correctly.

### Output Example

```
Guess a number between 1 to 100
Enter a number: 50
Lower!
Enter a number: 40
Higher!
Enter a number: 51
Higher!
Enter a number: 45
Lower!
Enter a number: 4
Congratulations. Total try = 5
```

A photograph of a person working at a desk. They are wearing an orange long-sleeved shirt and are holding a brown paper coffee cup with a black lid in their right hand. Their left hand is on a black computer keyboard. In front of them is a stack of papers or books. On the far left, a portion of a laptop screen is visible. The background shows a window with vertical blinds and a dark wall.

# End of Document



# Together for Tomorrow! Enabling People

Education for Future Generations

©2022 SAMSUNG. All rights reserved.

Samsung Electronics Corporate Citizenship Office holds the copyright of book.

This book is a literary property protected by copyright law so reprint and reproduction without permission are prohibited.

To use this book other than the curriculum of Samsung Innovation Campus or to use the entire or part of this book, you must receive written consent from copyright holder.