Variables and Data Types Operators

Topic 03 Key concept

1. Data Types

1.1 String and Number Data Types

- Earlier, we printed a number and a string enclosed in quotation marks using the print statement.

```
1 print('The radius of a circle', 4.0)
```

```
The radius of a circle 4.0
```

  - In the code above, 'The radius of a circle' is a string, and 4.0 is a number.

1. Data Types

1.1 String and Number Data Types

- Number data types consist of numbers, including integers (0, 1, 2, etc.), and floating-point numbers with decimal places (0.1, 0.12, etc.). They are referred to as "numeric types" or "number data types".
- Numeric types can perform calculations such as addition, subtraction, multiplication, and division.

1. Data Types

1.1 String and Number Data Types

- Words or sentences are in the form of a sequence of characters.
- They must be enclosed in single or double quotation marks to indicate to the computer where to start and end reading.
- The data type enclosed in quotation marks is called a "string data type" and is commonly referred to as a "string".

1. Data Types

1.1 String and Number Data Types

- Words or sentences are in the form of a sequence of characters.
- They must be enclosed in single or double quotation marks to indicate to the comput
where
- The da marks is called a "string data type" and
comm ly referred to as a "string".

We will learn various data types such as numeric types, strings, lists, etc., in the future.

1. Data Types

1.2 Number Data Types and Operators

- It is common in programming to perform operations using numerical information and then utilize this data.

e.g. Information like a person's height and weight is represented by numerical values such as 177cm and 58kg.

1. Data Types

1.2 Number Data Types and Operators

- In Python, numerical data is represented using numbers.
- Various mathematical operators are available for numerical data.

e.g. We can perform the addition operation between the numbers 100 and 20 using 100 + 20.

1. Data Types

1.2 Number Data Types and Operators

- Let's learn the terminology for operations. The data value or variable that is the subject of the operation is called an operand.
- The entity performing the operation is called an operator. Let's consider the example of the operation "100 + 200".

Operand	Operator	Operand
100	+	20

1. Data Types

1.2 Number Data Types and Operators

- The following table describes Python's operators and their functions.

Operator	Meaning	Operation
+	Addition	Adds the left operand and the right operand.
-	Subtraction	Subtracts the right operand from the left operand.
*	Multiplication	Multiplies the left operand by the right operand.
/	Floating-point Division	Divides the left operand by the right operand. In Python, division returns a float value by default.

1. Data Types

1.2 Number Data Types and Operators

- The following table describes Python's operators and their functions.

Operator	Meaning	Operation
//	Integer Division	Returns only the integer part of the division result, discarding the decimal places, unlike the / operator.
%	Modulus	Returns the remainder of the division.
**	Exponentiation	Raises the left operand to the power of the right operand. In the case of ** 0.5, it performs a square root operation.
//	Integer Division	Returns only the integer part of the division result, discarding the decimal places, unlike the / operator.

1. Data Types

1.2 Number Data Types and Operators

- You can perform operations on numeric data using numeric data types and operators such as +, -, *, /, //, %, **.

```
1 100 + 20
```

```
120
```

```
1 100 - 20
```

```
80
```

```
1 100 * 20
```

```
2000
```

1. Data Types

1.3 Order of Operator Execution

- When addition and multiplication operators appear in the same statement, multiplication takes precedence over addition.

```
1 10 + 20 * 30
```

```
610
```

```
1 (10 + 20) * 30
```

```
900
```

1. Data Types

1.3 Order of Operator Execution

- If parentheses appear in a statement, the operators inside the parentheses are executed first.

```
1 10 + 20 * 30
```

```
610
```

```
1 (10 + 20) * 30
```

```
900
```

1. Data Types

1.3 Order of Operator Execution

- The order of operator precedence affects the results of the calculations within the statement.

```
1 10 + 20 * 30
```

```
610
```

```
1 (10 + 20) * 30
```

```
900
```

1. Data Types

1.4 Distinguishing Data Types

- Distinguishing between data types is crucial in Python.
- Although characters and numbers can have the same form, they can have very different characteristics.

> e.g. The number 10 can be used in arithmetic operations, but the string '10' cannot.

1. Data Types

1.4 Distinguishing Data Types

- Different data types have different available operators. In the following case, adding the number 2 to the string '10' causes a syntax error.

TypeError

```
1 10 + 2
```

```
12
```

```
1 '10' + 2
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[16], line 1
----> 1 '10' + 2

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

1. Data Types

1.4 Distinguishing Data Types

- The behavior of operators differs based on the data type in Python.

```
1 100 * 2
```

```
200
```

```
1 'Hello' * 2
```

```
'HelloHello'
```

```
1 '100' * 2
```

```
'100100'
```

1. Data Types

1.4 Distinguishing Data Types

- The result of the multiplication operation (*) between the number 100 and 2 is 200.

```
1 100 * 2
```

```
200
```

```
1 'Hello' * 2
```

```
'HelloHello'
```

```
1 '100' * 2
```

```
'100100'
```

1. Data Types

1.4 Distinguishing Data Types

- However, the result of the multiplication operation (*) between the string '100' and the number 2 is '100100', and the result of the multiplication operation (*) between the string 'Hello' and the number 2 is 'HelloHello'.

```
100 * 2
```
```
200
```
```
'Hello' * 2
```
```
'HelloHello'
```
```
'100' * 2
```
```
'100100'
```

1. Data Types

1.4 Distinguishing Data Types

- However, the result of the multiplication operation (*) between the string '100' and the number 2 is '100100', and the result of the multiplication operation (*) between the string 'Hello' and the number 2 is 'HelloHello'.

Thus, the multiplication operator behaves differently for strings and numeric types.

1. Data Types

1.4 Distinguishing Data Types

- The behavior of operators differs based on the data type in Python.

```
100 + 2
```
```
102
```
```
'100' + '2'
```
```
'1002'
```
```
'Hello' + ' World!'
```
```
'Hello World!'
```

1. Data Types

1.4 Distinguishing Data Types

- The result of the addition operation (+) between the number 100 and 2 is 102.

```
1 100 + 2
```

```
102
```

```
1 '100' + '2'
```

1. Data Types

1.4 Distinguishing Data Types

- However, the result of the addition operation (+) between the string '100' and '2' is '1002', and the result of the addition operation (+) between the string 'Hello' and ' World!' is 'Hello World!'. The operator concatenates strings.

```
1 100 + 2
```

```
102
```

```
1 '100' + '2'
```

```
'1002'
```

1. Data Types

1.4 Distinguishing Data Types

- However, the result of the addition operation (+) between the string '100' and '2' is '1002', and the result of the addition operation (+) between the string 'Hello' and ' World!' is 'Hello World!'. The operator concatenates strings.

Thus, the addition operator behaves differently for strings and numeric types.

1. Data Types

1.5 Checking Data Types

- Use the type function to check the data type.

```
1 type(100)
```

```
int
```

```
1 type(100.0)
```

```
float
```

```
1 type('100')
```

```
str
```

1. Data Types

1.5 Checking Data Types

- Numeric types include various types; those with decimal places are called float types, and those without decimal places are called int types.
- Strings are of type str.

```
1 type(100)
```

```
int
```

```
1 type(100.0)
```

Numeric types 100 and 100.0 are different data types.

```
float
```

```
1 type('100')
```

```
str
```

One Step Further

- In a Jupyter Notebook cell, type(100) outputs int, but print(type(100)) outputs <class 'int'>.
- Class refers to an independent object that performs a specific role in software. Therefore, 10 is an object of type int.
- Classes will be discussed in detail later.

```
1 type(10)
```

```
int
```

```
1 print(type(10))
```

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

1. Data Types

1.6 Data Type Conversion

- Python data can be converted to different data types for calculations.

```
1 '10' + str(2)
```

```
'102'
```

```
1 'I like number ' + str(10)
```

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

```
1 int('10') / 2
```

```
5.0
```

1. Data Types

1.6 Data Type Conversion

- To convert the number 10 to the string '10', use str(10). The converted string can be concatenated with other strings using the addition operation.

```
1 '10' + str(2)
```

```
'102'
```

```
1 'I like number ' + str(10)
```

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

1. Data Types

1.6 Data Type Conversion

- The string '10' can be converted to int('10') successfully. Therefore, int('10') / 2 can be executed without errors.

```
1 '10' + str(2)
```

```
'102'
```

```
1 'I like number ' + str(10)
```

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

```
1 int('10') / 2
```

```
5.0
```

1. Data Types

1.7 bool Data Type

- The commonly used basic data types in Python and their examples are as follows.

Data type	Example
int	⋯, -3, -2, -1, 0, 1, 2, 3, ...
float	3.14, 4.28, 0.01, 123.432
str	'Hello World', '123', "Hi"
bool	True, False

1. Data Types

1.7 bool Data Type

Data type	Example
int	⋯, -3, -2, -1, 0, 1, 2, 3, ...
float	3.14, 4.28, 0.01, 123.432
str	'Hello World', '123', "Hi"
bool	True, False

- In addition to commonly used data types such as int, float, and str, there is a bool data type in Python. It represents true and false values.

2. Variables

2.1 What is a Variable?

- When writing a program, if values need to be constantly changed, any mistakes can lead to errors in the program.

```
1 print('The radius of a circle', 4.0)
2 print('The area of a circle', 3.14 * 4.0 * 4.0)
3 print('The circumference of a circle', 2.0 * 3.14 * 4.0)
```

```
The radius of a circle 4.0
The area of a circle 50.24
```

- In the code for calculating the area of a circle, if we need to change the radius value '4.0', making even a single mistake can lead to unintended results.

2. Variables

2.1 What is a Variable?

- In such cases, we can introduce variables to simplify the task.

```
1 radius = 4.0
2
3 print('The radius of a circle', radius)
4 print('The area of a circle', 3.14 * radius * radius)
5 print('The circumference of a circle', 2.0 * 3.14 * radius)
```

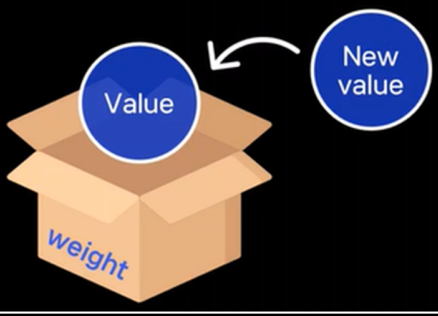
```
The radius of a circle 4.0
The area of a circle 50.24
The circumference of a circle 25.12
```

- In the code for calculating the area of a circle, we modified the code to receive the radius value '4.0' using the name 'radius'.

2. Variables

2.1 What is a Variable?

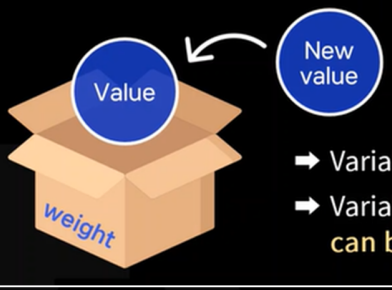
- Therefore, when writing a program, we need a name to store data values. This is where variables come in.
- "Variable" is like a name given to an empty box that can hold any value. Once the name is provided, we can use the value inside the box.



2. Variables

2.1 What is a Variable?

- As indicated by "Variable," the stored value can change over time.



➡ Variables are a fundamental concept in programming.

➡ Variables are spaces in the computer where values can be stored.

2. Variables

2.1 What is a Variable?

- Store values in variables and then use the variables instead of the values.

- In this case, it is more accurate to understand the value that goes into the box as data.
- Data 78.7 The value is stored in 'weight'.

2. Variables

2.2 Defining Variables

- Let's define variables as spaces for storing data.

e.g. Store your weight as a variable named 'weight'. To do this, use the following method.

```
1 weight = 78.7
2 weight
```

```
78.7
```

2. Variables

2.2 Defining Variables

- Different programming languages have different ways of defining variables.
- If you look carefully at the variable name, programming languages like C and Java attach a data type to the front of the name, but Python does not attach a data type!.

Python	C, JAVA
weight = 78.7	int weight = 78.7

2. Variables

2.2 Defining Variables

```
1 weight = 78.7
2 weight
```

```
78.7
```

Line 1

- When the code is executed, a variable named 'weight' is created in memory, and the value 78.7 is stored in it.
- The '=' symbol in the code does not mean 'equals,' but rather means 'assign the value on the right to the variable named weight.' This '=' operator is called an assignment operator.

One Step Further

- The method by which the data type is determined when a value is assigned to a variable is called dynamic typing.
- Python uses dynamic typing, so there is no need to declare the data type in advance.

```
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'>
```

One Step Further

- Dynamic: Refers to actions that occur during program execution.
- Static: In contrast, it refers to actions that are determined in advance before the program is executed.
- Python uses dynamic typing, so it can code without specifying a specific data type, making the program flexible.
- On the other hand, there are also languages that declare the data type in advance.
  In a static typing system like C, it is possible to filter out actions that try to put incorrect values into specified data types or perform operations with incompatible data types at the source code interpretation stage before program execution.

One Step Further

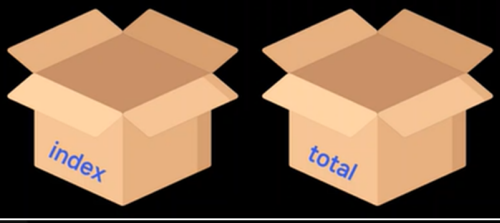
- Let's examine the difference between the two approaches using a table.

Static typing	Dynamic typing
• The data type is checked at compile time. • Each data type must be explicitly specified. • Used in: C, C++, C#, JAVA, Objective-C, PASCAL, etc.	• No need to specify each data type, making the code concise. • On the other hand, runtime type errors can occur. • Used in: Python, Basic, Ruby, PHP, JavaScript, etc.

2. Variables

2.3 Naming Conventions

- Identifier: We will learn how to handle a lot of data. To distinguish these data in the code, we need different names.
- Just as we distinguish people by names like 'Hong Gil-dong' and 'Kim Cheol-su,' identifiers distinguish variables. Identifiers must be created according to the following rules.



2. Variables

2.3 Naming Conventions

- Identifiers consist of letters, numbers, and underscores (_) and cannot use special characters other than these.
- The first letter of an identifier cannot be a number.
- Identifiers cannot have spaces in between. Instead of spaces between words, use underscores (_).
- Upper and lower case letters are distinguished in identifiers.
  Therefore, variables index and INDEX are different variables.
- Python's reserved words (keywords) cannot be used as identifiers.

2. Variables

2.3 Naming Conventions

- Although we can freely name variables, we cannot use reserved words that Python internally uses as variable names. Reserved words cannot be used as identifiers.
  - Words that are pre-assigned roles in Python to perform specific tasks are called keywords or reserved words.
  - Keywords can only perform predefined roles within Python,
    so they cannot be used as identifiers. If it is specified as an identifier,
    it cannot perform the role of a command because it overlaps with the existing role.

2. Variables

2.3 Naming Conventions

- To check Python's keywords

```
1 import keyword
2
3 print("Python keyword list...")
4 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', 'globa
l', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'r
eturn', 'try', 'while', 'with', 'yield']
```

2. Variables

2.3 Naming Conventions

- Be careful when creating variables by using keywords, as an error occurs if the following code is used.

SyntaxError

```
1 global = 500
```

```
Cell In[3], line 1
  global = 500
         ^
SyntaxError: invalid syntax
```

2. Variables

2.3 Naming Conventions

- Examples of unavailable identifiers and reasons.

Unavailable Identifier	Reason
1st_variable	Cannot be used because it starts with a number.
my list	Cannot be used because it contains a space.
global	Python keywords cannot be used as identifiers.
ver2.9	Uses special characters other than underscores.
num&co	Uses special characters other than underscores.

One Step Further

- How should we name variables?
- When naming variables, we should choose names that best describe the role of the variable. Good variable names make it easy to read the entire program.
- However, if you use names that are given without much thought, it will be very difficult to read the program later. This habit will undoubtedly be helpful later on.

One Step Further

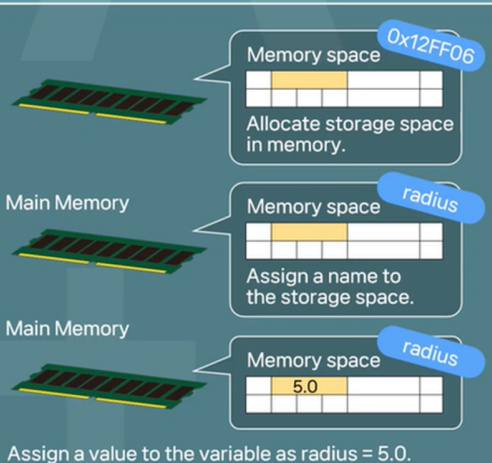
- Let's look at an example.

e.g. It would be easier to understand weight and height than x1 and x2, which represent weight and height, respectively.

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

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

One Step Further

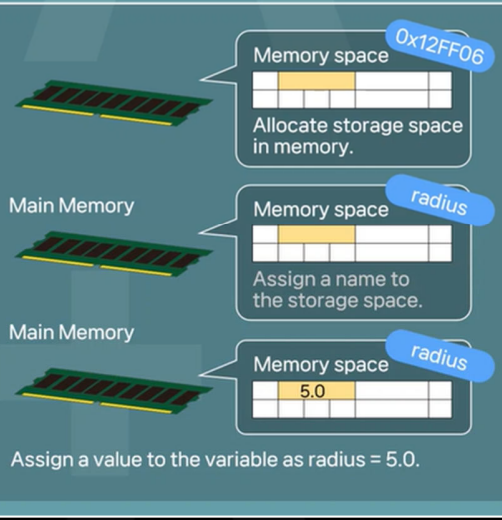


Assign a value to the variable as radius = 5.0.

- Why assign a name?
- Main Memory

  The place where a computer's data is stored for reading, writing, and overwriting is called main memory or memory.

One Step Further



- **Memory Address**
  The location where data is stored in memory. To read and write stored data, you need to know where the data is stored (space or location). Addresses are usually expressed in hexadecimal.
- It is convenient to use variable names like radius instead of addresses.

2. Variables

2.4 Multiple Assignment

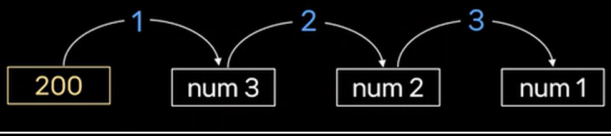
- **Multiple assignment** like num1 = num2 = num3 = 200 is also possible.

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

```
200 200 200
```

Line 1
- All num1, num2, and num3 are assigned the value 200.
  The assignment order is from the right equal sign.

The execution order is as follows.



2. Variables

2.5 Simultaneous Assignment

Focus
- You can declare multiple variables on one line and assign values to them simultaneously. This is called **simultaneous assignment**.

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

```
300
```

- The reason why simultaneous assignment is possible in Python is that there is a data type called 'tuple' in Python, but we will learn more about it later.

One Step Further

- In Python, you can use None if the data type is not determined.
- The data type of a variable 'a' with None as its value is NoneType.
- None seems unnecessary as it represents the absence of a value.
  However, it is often necessary as it is the only way to indicate the absence of a value.

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

```
NoneType
```

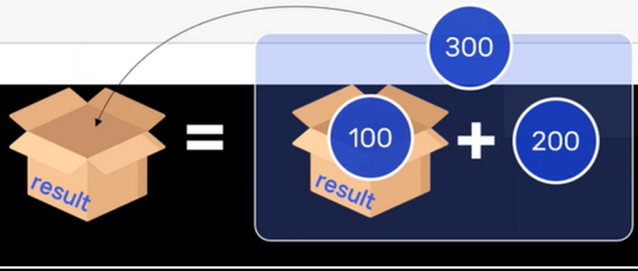
3. Operators

3.1 Shorthand Operators

- Let's examine the expression result = result + 200. This expression means to assign the value on the right to the variable named result on the left.

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

```
300
```



3. Operators

3.1 Shorthand Operators

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

```
300
```

- When assigning the variable to itself, the assignment operator (=) and addition operator (+) can be combined into shorthand notation.

3. Operators

3.1 Shorthand Operators

- Let's try using the shorthand operator += in the code that accumulates variable values.

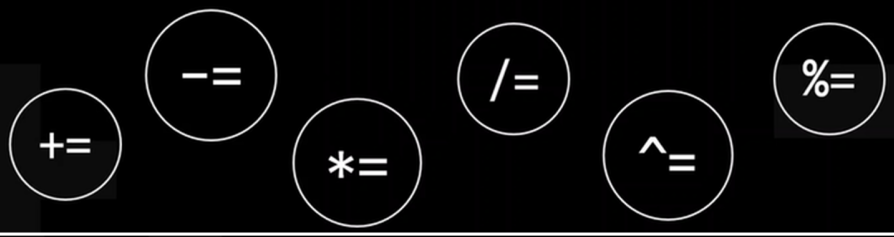
```
1 result = 100
2 result += 200
3 print(result)
```

```
300
```

3. Operators

3.1 Shorthand Operators

- Python supports various shorthand operators as shown below.
- Even if you are not familiar with these shorthand operators, you can code without any problems using the original arithmetic and assignment operators.



3. Operators

3.1 Shorthand Operators

- However, if you are not familiar with these commonly used shorthand operators, there may be problems when reading other people's code.

Operator	Description	Ex
+=	Combines the addition operator and simple assignment operator.	i+=10
-=	Combines the subtraction operator and simple assignment operator.	i-=10
*=	Combines the multiplication operator and simple assignment operator.	i*=10
/=	Combines the division operator and simple assignment operator.	i/=10
^=	Combines the bitwise XOR operator and simple assignment operator.	i^=10
%=	Combines the modulus operator and simple assignment operator.	i%=10

3. Operators

3.1 Shorthand Operators

```
num = 200
num += 100
```

Focus

It is important to understand shorthand operators because it may cause problems when reading other people's code if you are not familiar with them.

3. Operators

3.1 Shorthand Operators

```
num = 200
num += 100
print(num)
num -= 100
print(num)
num *= 20
print(num)
num /= 2
print(num)
```

```
300
200
4000
2000.0
```

3. Operators

3.2 Python's Swap

- Let's say 'a' is assigned 100, and 'b' is assigned 200.
  We want to exchange the values of these two variables.
- The following code shows a common method for value swapping.

```
a = 100
b = 200
print('Before swap: a =', a, 'b =', b)
temp = a
a = b
b = temp
print('After swap: a =', a, 'b =', b)
```

```
Before swap: a = 100 b = 200
After swap: a = 200 b = 100
```

3. Operators

3.2 Python's Swap

- To perform value swapping, a temporary variable 'temp' is used with three assignment statements.

```
a = 100
b = 200
print('Before swap: a =', a, 'b =', b)
temp = a
a = b
b = temp
print('After swap: a =', a, 'b =', b)
```

```
Before swap: a = 100 b = 200
After swap: a = 200 b = 100
```

- However, in Python, it can be written more simply.

3. Operators

3.2 Python's Swap

- Let's modify the code from the previous page to make it simpler.

```
1 a = 100
2 b = 200
3 print('Before swap: a = ', a, 'b = ', b)
4 a, b = b, a
5 print('Swap result using tuple: a = ', a, 'b = ', b)
```

```
Before swap: a =  100 b =  200
Swap result using tuple: a =  200 b =  100
```

3. Operators

3.2 Python's Swap

- In Python, we can use a single line like a, b = b, a to exchange values.
- The code is very simple and intuitive.

```
1 a = 100
2 b = 200
3 print('Before swap: a = ', a, 'b = ', b)
4 a, b = b, a
5 print('Swap result using tuple: a = ', a, 'b = ', b)
```

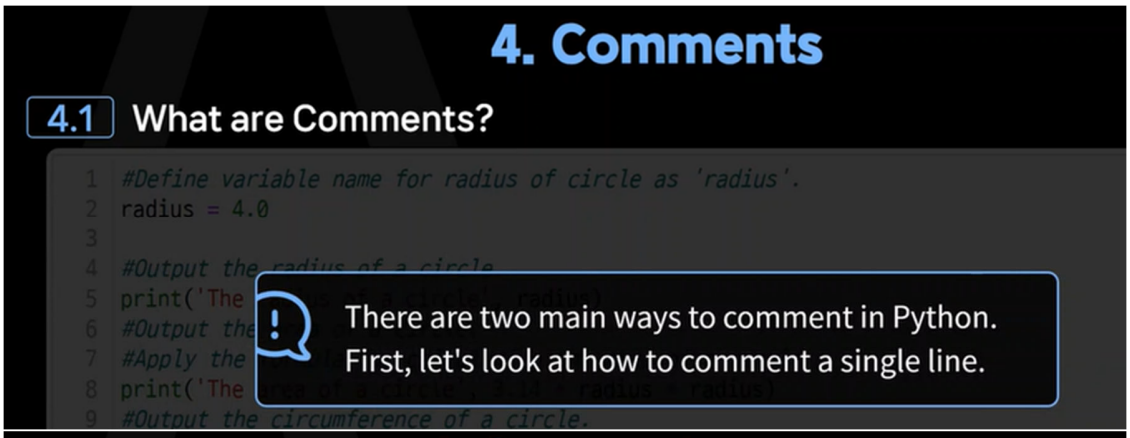
```
Before swap: a =  100 b =  200
Swap result using tuple: a =  200 b =  100
```

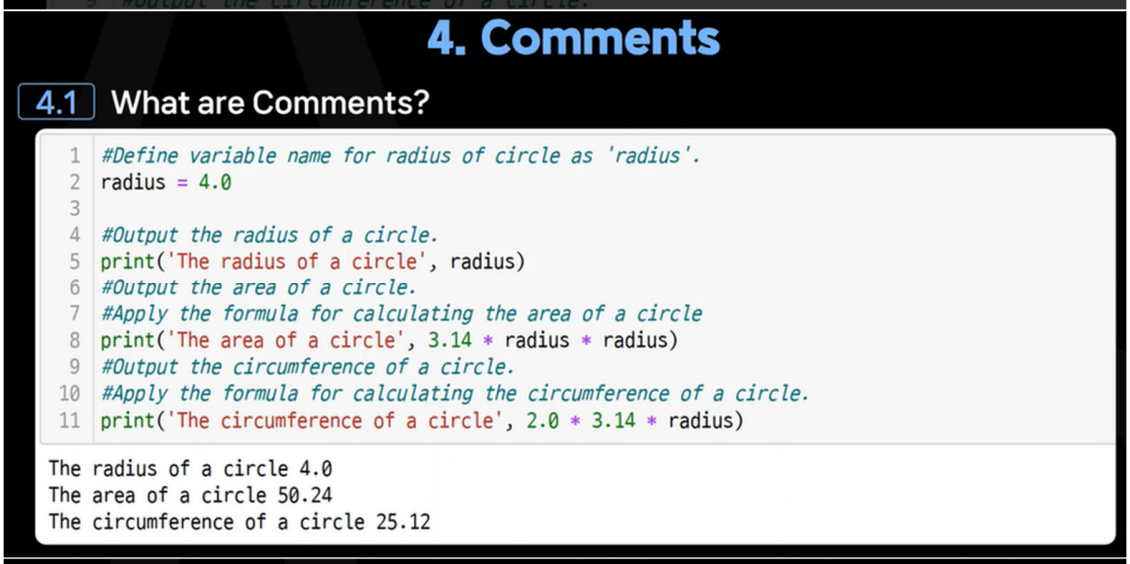
  - The reason why Python can implement swap code easily and simply, unlike other programming languages, is that Python has a data type called 'tuple.' We will learn more about this later.

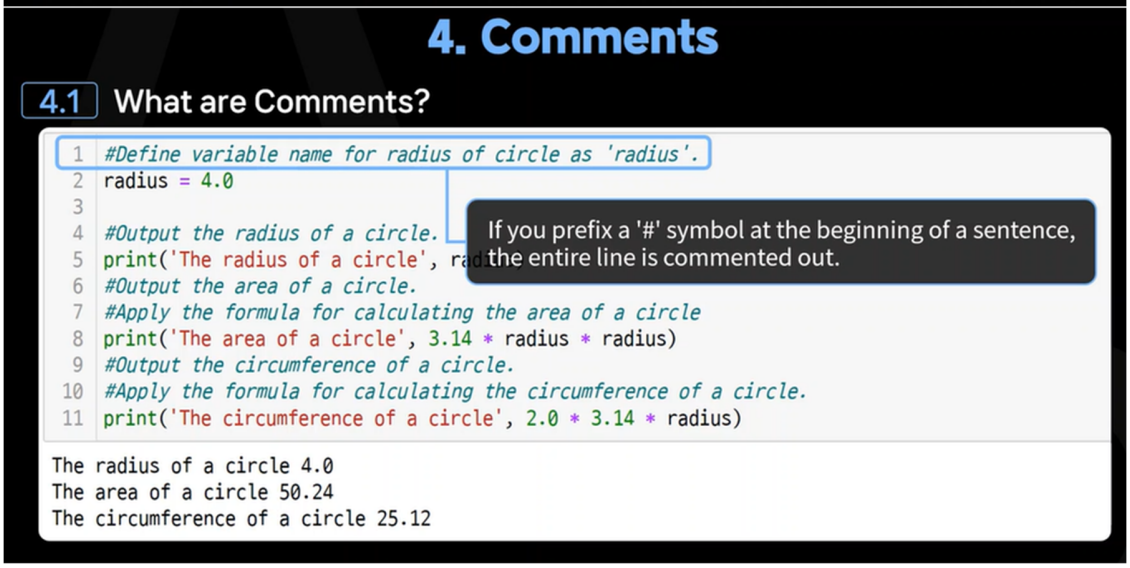
4. Comments

4.1 What are Comments?

- Comments are sentences used in a program to explain the functionality of the code. They are not executed by the computer and are skipped.
- Comments are crucial in programming. When someone else modifies your code, comment statements help them understand the meaning of the code.
- Also, if you write explanations next to difficult code, it will be very helpful when you read the code again.


4. Comments
4.1 What are Comments?
#Define variable name for radius of circle as 'radius'.
radius = 4.0
There are two main ways to comment in Python.
First, let's look at how to comment a single line.


4. Comments
4.1 What are Comments?
#Define variable name for radius of circle as 'radius'.
radius = 4.0
#Output the radius of a circle.
print('The radius of a circle', radius)
#Output the area of a circle.
#Apply the formula for calculating the area of a circle
print('The area of a circle', 3.14 * radius * radius)
#Output the circumference of a circle.
#Apply the formula for calculating the circumference of a circle.
print('The circumference of a circle', 2.0 * 3.14 * radius)
The radius of a circle 4.0
The area of a circle 50.24
The circumference of a circle 25.12


4. Comments
4.1 What are Comments?
#Define variable name for radius of circle as 'radius'.
radius = 4.0
#Output the radius of a circle.
print('The radius of a circle', radius)
#Output the area of a circle.
#Apply the formula for calculating the area of a circle
print('The area of a circle', 3.14 * radius * radius)
#Output the circumference of a circle.
#Apply the formula for calculating the circumference of a circle.
print('The circumference of a circle', 2.0 * 3.14 * radius)
If you prefix a '#' symbol at the beginning of a sentence, the entire line is commented out.
The radius of a circle 4.0
The area of a circle 50.24
The circumference of a circle 25.12

4. Comments

4.2 Comments Spanning Multiple Lines

- To write multiple lines of comments, you can use three single quotes (') or three double quotes (") at the beginning and end of the sentences.

```
radius = 4.0
print('The radius of a circle', radius)
print('The area of a circle', 3.14 * radius * radius)
print('The circumference of a circle', 2.0 * 3.14 * radius)
```

```
'''radius = 4.0
print('The radius of a circle', radius)
print('The area of a circle', 3.14 * radius * radius)
print('The circumference of a circle', 2.0 * 3.14 * radius)'''
```

4. Comments

4.2 Comments Spanning Multiple Lines

- Using this method, you can even comment out the entire code by selecting the entire code and typing three single quotes or three double quotes.

```
radius = 4.0
print('The radius of a circle', radius)
print('The area of a circle', 3.14 * radius * radius)
print('The circumference of a circle', 2.0 * 3.14 * radius)
```

```
'''radius = 4.0
print('The radius of a circle', radius)
print('The area of a circle', 3.14 * radius * radius)
print('The circumference of a circle', 2.0 * 3.14 * radius)'''
```

5. Pythonic way

5.1 Why is it necessary to code in a Pythonic way?

- It is important to develop good code style and habits, especially when you are learning Python for the first time.
- The main purpose is to write code that is clear and concise using the unique features that Python has compared to other programming languages.
- Following the rules used by other developers in the Python community is advantageous when collaborating with others.
- There are various Python code style guides, but we will only learn the essential ones that beginners should follow.

5. Pythonic way

5.1 Why is it necessary to code in a Pythonic way?

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

- The representative Python code style guides are two:

  PEP8 from the Python official community and Google Python Style Guide.
- This lesson will follow the guidelines of PEP8.

5. Pythonic way

5.2 Code layout

- In Python, you should be able to have spaces when writing code to distinguish separators and symbols. A comma (,) is a separator.
- In Python, square brackets ([]), curly brackets({}), and parentheses(()) are not separators but symbols, which we will learn more about later.

5. Pythonic way

5.2 Code layout

- The following coding style is not readable.

```
1 a=10;
2 b=[1,2,3]
```

- Therefore, put spaces around operators and separators (comma, colon, equal sign).

```
1 a = 10;
2 b = [1, 2, 3]
```

5. Pythonic way

5.2 Code layout

- When coding in Python, you can use a semicolon to write multiple statements on one line. However, this coding style is not readable.

```
1 a = 10; print(a)
```

- Therefore, use one statement per line.

```
1 a = 10;
2 print(a);
```

5. Pythonic way

5.2 Code layout

- There may be very long lines of code like the following example.
  These long statements are inconvenient to read at a glance.

```
1 print('Define variable name for radius of circle. Output the radius, area, circumference of a
```

```
Define variable name for radius of circle. Output the radius, area, circumference of a circle. App
f a circle. Apply the formula for calculating the circumference of a circle.
```

5. Pythonic way

5.2 Code layout

- When coding in Python, when statements become long, use line breaks as shown below.
  Use a backslash (\) for line breaks.

```
1 print('Define variable name for radius of circle. \
2 Output the radius, area, circumference of a circle. \
3 Apply the formula for calculating the area of a circle. \
4 Apply the formula for calculating the circumference of a circle.')
```

```
Define variable name for radius of circle. Output the radius, area, circumference of a circle. App
f a circle. Apply the formula for calculating the circumference of a circle.
```

5. Pythonic way

5.2 Code layout

- Let's change the code by using a backslash (\) for line breaks.

```
1 radius = 4.0
2 print('The radius of a circle',\
3       radius)
4 print('The area of a circle',\
5       3.14 * radius * radius)
6 print('The circumference of a circle',\
7       2.0 * 3.14 * radius)
```

```
The radius of a circle 4.0
The area of a circle 50.24
The circumference of a circle 25.12
```