Python Basics

Eunji Kim
eunjikim@cau.ac.kr

In this tutorial, we cover...

- Variable
- Comments
- Data Types
- Python Numbers: `int` and `float`
- Arithmetic operators
- Assignment operators

Variable

We can assign a value to a variable using `=`.

```
name = 'Eunji Kim'      # Your name
x = 34                  # number
y = "Hello"             # Another one.
z, w = 3.45, -1         # Assign two variables at once
```

> 📝 Variable Naming Rules:
>
> - Start with a letter or underscore `_`
> - Case-sensitive
> - No reserved keywords (e.g., `for`, `list`, `str`)
> - No spaces or special charactors

Comments

Python has commenting capability for the purpose of in-code documentation.

Comments starts with a `#`, and Python will ignore the rest of the line:

```
# This is a comment and will be ignored by Python
print("Hello, World!")

print("Hello, World!") # This is also a comment
```

A comment does not have to be text that explains the code, it can also be used to prevent Python from executing code:

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

Data Type

Every value in Python has a datatype.

```
# atomic data types
x1 = 42                          # integer
x2 = 3.14                        # float
x3 = True                        # boolean

# sequence data types
y1 = 'python'                    # string
y2 = [2, 3, 5, 7, 11]            # list
y3 = (2, 3, 5, 7, 11)            # tuple

# collection data types
z1 = {'cat', 'dog', 'duck'}                            # set
z2 = {'cat': 'meow', 'dog': 'arf', 'duck': 'honk'}     # dictionary
```

> 📝 Python uses **dynamic typing**, so you don't need to declare variable types.
> You can check the datatype of a variable using `type()`.

```
>>> type(x1)
<class 'int'>
```

If you want to specify the data type of a variable, this can be done with **casting**.

```
x = 3.14
y = str(x) # y will be "3.14"
z = int(x) # z will be 3
```

Numbers

Python has numeric data types: `int` and `float`.

```
x = 1    # int
y = 2.8  # float
```

Arithmetic Operators

We can perform mathematical operations on numbers using the arithmetic operators below:

Symbol	Task Performed	Meaning	Example
`+`	Addition	add two operands	`x + y`
`-`	Subtraction	substract right operand from the left	`x - y`
`*`	Multiplication	Multiply two operands	`x * y`
`/`	Division	Divide left operand by the right one (always results into float)	`x / y`
`//`	Integer/Floor division	division that results into whole number adjusted to the left in the number line	`x // y` (quotient of **x/y**)
`%`	Modulus (remainder)	remainder of the division of left operand by the right	`x % y` (remainder of **x/y**)
`**`	Exponentiation (power)	left operand raised to the power of right	`x ** y` (**x** to the power **y**)

Example: Arithmetic operations

```
x = 8
y = 2

print('x + y =', x + y)
print('x - y =', x - y)
print('x * y =', x * y)
print('x / y =', x / y)
print('x // y =', x // y)
print('x % y =', x % y)
print('x ** y =', x ** y)
```

Output:

```
x + y = 10
x - y = 6
x * y = 16
x / y = 4.0
x // y = 4
x % y = 0
x ** y = 64
```

Example: Comparison opeartions

Python compares `x` and `y` and returns a ***boolean***, either `True` or `False`.

```
x = 17
y = 42

print(f"x == y   : {x == y}")
print(f"x != y   : {x != y}")
print(f"x > y    : {x > y}")
print(f"x < y    : {x < y}")
print(f"x >= y   : {x >= y}")
print(f"x <= y   : {x <= y}")
```

Output:

```
x == y   : False
x != y   : True
x > y    : False
x < y    : True
x >= y   : False
x <= y   : True
```

String

In Python, a string is a sequence of characters.

```
mystr = "Python"
```

> 📝 Strings can be surrounded by either single quotes (`'Python'`) or double quotes (`"Python"`).

String Formatting

You can format a string using several methods: `f`-strings, the `.format()` method, and the modulo operator (`%`).

```
adj = "Red"
noun = "Alert"

cheese = f"{adj} {noun}"            # F-string is the most preferred way
print(cheese) # Red Alert

cheese = "{} {}".format(adj, noun) # Also possible since Python 3
print(cheese) # Red Alert

cheese = "%s %s" % (adj, noun)      # This modulo style was deprecated (PEP 3101)
print(cheese) # Red Alert
```

> 📝 The curly brakets `{}` in `f`-string is called **placeholders**.

List

List is used to store multiple items in a single variable.

```
mylist = ["apple", "banana", "cherry"]
```

- To create a list, enclose a set of items in square brakets `[]`.
- To check the length of a list, use `len()` :

```
print(len(mylist)) # 3
```

For Loops

A `for` loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).

Iterate over a list:

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

Output:

```
apple
banana
cherry
```

Iterate over a string:

```
mystr = 'Python'
for c in mystr:
    print(c)
```

Output:

```
P
y
t
h
o
n
```

`if-else` statements

In programming you often need to know if an expression is `True` or `False`.

```
if a > b:
    print('a is greater than b')
else:
    print('a is not greater than b')
```

Breakdown:

- The `if` statement executes `print('a is greater than b')` only if the specified condition `a > b` evaluates to `True`.
- The `else` staement executes `print('a is not greater than b')` when the condition is `False`.

Indentation

Indentation is whitespace at the beginning of a line.

Python relies on **indentation** (using **4 spaces** or **1 tab**) to define scope in the code.

If statement without indentation will raise an error:

```
a = 33
b = 200

if b > a:
print("b is greater than a")
```

Output:

```
IndentationError: expectred an indented block
```

> 📝 Other programming languages often use curly-brackets for this purpose.

Logical Operators: `and`, `or`, and `not`

Logical operators are used to combine multiple conditions:

```
# Define variables
temperature = 25  # in Celsius
is_weekend = True
is_holiday = False

if is_weekend or is_holiday:
    if temperature > 20 and is_weekend:
        print("Let's go to the park!")
    else:
        print("It's not a great day for the park.")
else:
    print("Let's go to work...")
```

Output:

```
Let's go to the park!
```

The `range()` Function

A built-in function used to generate a sequence of numbers.

- It is commonly used in `for` loops to iterate a specific number of times or over a sequence of integers.
- `range(stop)`: Generates a sequence of numbers starting from `0` (inclusive) and ending before the `stop` value (exclusive), incrementing by `1`.

```
# Example: range(5) generates 0, 1, 2, 3, 4
for i in range(5):
    print(i)
```

- `range(start, stop)`: Generates a sequence of numbers starting from `start` (inclusive) and ending before the `stop` value (exclusive), incrementing by `1`.

```
# Example: range(1, 6) generates 1, 2, 3, 4, 5
for i in range(1, 6):
    print(i)
```

- `range(start, stop, step)`: Generates a sequence of numbers starting from `start` (inclusive) and ending before the `stop` value (exclusive), incrementing or decrementing by the `step` value. The step can be positive or negative.

```
# Example: range(0, 10, 2) generates 0, 2, 4, 6, 8
for i in range(0, 10, 2):
    print(i)

# Example: range(10, 0, -1) generates 10, 9, 8, 7, 6, 5, 4, 3, 2, 1
for i in range(10, 0, -1):
    print(i)
```

Functions

A function is a block of code which only runs when it is called.

You can pass data, known as parameters or arguments, into a function.

A function can return data as a result.

Creating a Function

A function is defined using the `def` keyword:

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


def add(a, b):      # two input arguments
    return a + b    # return the addition result
```

Calling a Function

To call a function, use the function name followed by parenthesis:

```
print_hello()
print_hello()

x = add(9, 7)
print(x)
```

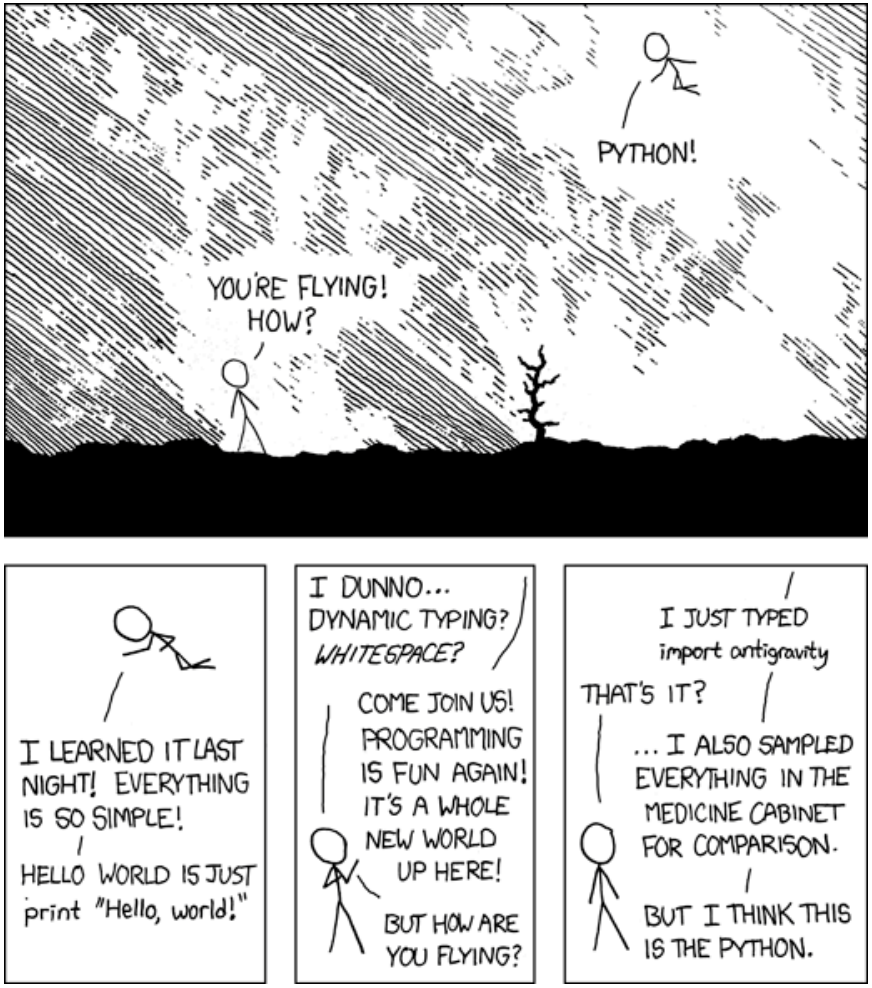
Output:

```
Hello from a function
Hello from a function
16
```

Modules

Consider a module to be the same as a code library.

You can use a module using `import`.



Built-in Modules

There are several built-in modules in Python, which you can import whenever you like, such as `os`, `math`, and `datetime`.

```
import math
x = math.sqrt(4)
print(x) # 2
```

```
from datetime import datetime
x = datetime.now()

print(x) # 2025-08-29 12:49:41.718642
print(f"This year is {x.year}!") # This year is 2025!
```

Third-party Modules

These are modules created by the Python community.

- Popular examples for Data Science: `pandas`, `numpy`, `matplotlib`
- They extend Python's functionality far beyond what is included by default, allowing you to perform specialized tasks.

You can install them from the Python Package Index (PyPI) using `pip`.

```
pip install pandas
```

Thank You.