SAMSUNG

Samsung Innovation Campus

Coding and Programming

Together for Tomorrow!
Enabling People
Education for Future Generations

Python Handbook

Coding and Programming

Handbook Description

Handbook objectives

- ✓ Students can understand the basic concept of Python and computer programming. Also, they can learn about different data types, operators, conditional statements, loops, functions, etc.

Handbook contents

Unit 1.

About Python

1. What is Computer Programming?

Computational thinking is a scientific way of thinking to analyze real life problems and find suitable solutions. **Computer programming** is to write these solutions into command lines for computer. It can be defined as "creating a plan in advance for computer."

Pro	+	Gram	=	ProGram
'In Advance'	+	'Created'	=	Created in advance

2. Why Python?

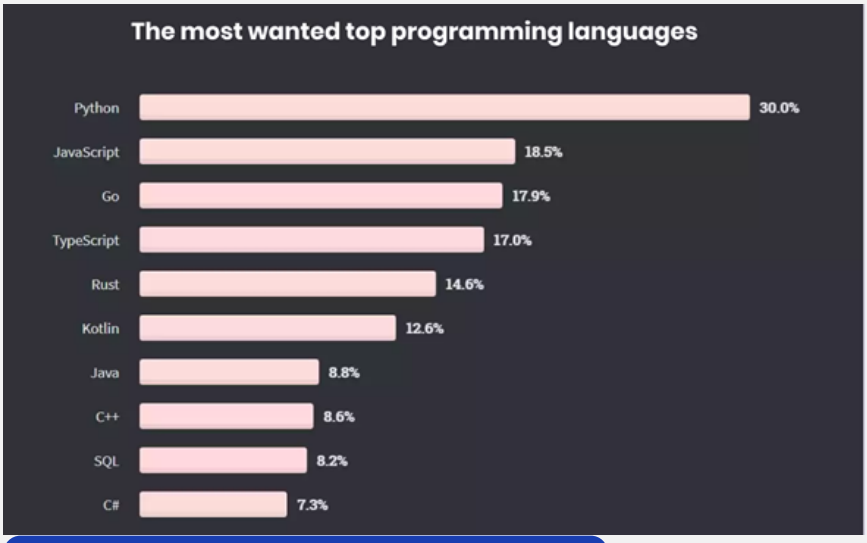
There are unimaginable numbers of languages in the world. Similarly, programming language also has different types of languages, and Python is one of them.

Why learn English? → Most spoken language

Universal

Why learn Python? → Most utilized language

Python is one of the most popular programming languages around the world, and it is ranked among the top programming languages. As shown in the below survey, Python is wanted not only in educational fields but also by many developers in professional workplaces.



https://www.codica.com/blog/top-programming-languages-trends

Advantages of Python

1) Easy to learn
   - Grammar is easy to learn and understand.
   - C language has complex technique that even the computer majors have trouble understanding.

2) Coding as the thought flows
   - Extremely easy to transcribe a plan and design into coding due to flexibility of Python

3) Flexible in a variety of fields
   - Used by web servers, hackers, IoT technology, and Artificial Intelligence

4) Utilized by most of the operating systems
   - Including Windows, Mac, and Linux

Disadvantage of Python

1) Around 10 to 350 times slower than C language
   - In general, C, C++, and C# are utilized to create high budget games.

However, is it really a disadvantage?

Exponential growth of computer quality made the speed difference between programming languages inconsequential, but we can feel some difference when we create a program that requires a lot of computation. Still, machine learning and deep learning use **Python for overall programming**, while they use C for operation dependent programs.

In conclusion, Python can be used for almost every programming except when creating operation dependent programs.

Unit 2.

Basic Concept

- 2.1 Expression and Statement
- 2.2 Reserved Keyword
- 2.3 Identifier
- 2.4 print()
- 2.5 Comment

1. Expression and Statement

Expression is a representation of values. Values can be something like a number or a string. Statement is a combination of expressions that the Python interpreter can execute. A line can become a statement.

```
In [1]: 273
        10 + 20 + 30 * 10
        'Python Programming'
```

Focus **Expressions** form a **statement**, and **statements** form a **program**.

2. Reserved Keyword

Python reserved keywords are special reserved words that have specific purpose and cannot be used for anything but those specific purposes.

False	None	True	and	as	assert
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			

Focus Reserved keywords cannot be used when assigning a name to a value.

3. Identifier

Identifiers are the names used to identify a variable, function, class, module or other objects.

‹ Rules when creating an identifier ›

1) Keywords cannot be an identifier.
2) Underscore (_) is the only special letter allowed.
3) You cannot start it with a numeric value.
4) It must not include spaces.

Naming convention:

Without using spaces, English words take time to understand. So the programmers came up with snake_case and CamelCase to make identifiers easier to interpret. Using underscore (_) in between words is called snake_case, and using capital letters for words is called CamelCase. However, we do not use CamelCase in Python. snake_case is considered as a more Pythonic style.

Identifiers with no spaces	Using _ between words (snake_case)	Using capital letters (CamelCase)
itemlist loginstatus characterhp rotateangle	item_list login_status character_hp rotate_angle	ItemList LoginStatus CharacterHp RotateAngle

TIP

You can refer to other style guide for Python code in the below website:
https://www.python.org/dev/peps/pep-0008/

4. print()

The print() function prints the given object to the standard output device screen or to the text stream file. You can print out any amount of object or value.

```
In [1]: # You can print out only one object.
        print('Hello')
        print(54)

        # Or you can print out more than one.
        print('Hello', 54, 1240)

        # You can also make a new line by typing nothing into the function.
        print()
```

```
Hello
54
Hello 54 1240
```


5. Comment

Comments are used to add notes to the code or to prevent Python from interpreting specific parts of the line. Hashtag (#) is used in front of a line to process it as a comment. Words after # are all interpreted as comments, not affecting the program.

```
In [1]: # This is an example of a comment.
        print('Hello, World!') # This will print the string.
```

```
Hello, World!
```

You can also write a multiline comment with either # or '''.

In [1]:
```
# This is a comment
# written in more than
# one line.
print('Hello, World!')
```

```
Hello, World!
```

In [2]:
```
'''
This is a comment
written in more than
one line.
'''
print('Hello, World!')
```

```
Hello, World!
```


Unit 3.

Data Type

1. Variable

Variable is a name given to a memory location, that is used to hold a single value. You can put any data type into a variable, including a string, integer, floating point number, Boolean, etc.

variable name = value

The direction of value allocation

3 steps of utilizing a variable:

1) **Declare** a variable: we must first decide a word that we will use for a variable (ex: a, x, dic).
2) **Allocate** a value into the variable: we choose a value (ex: 2.45, 'computer') for the variable.
3) **Reference** the variable: after the allocation is complete, we can use the variable (in this case, x) to reference the value we have allocated.

```
In [1]: x = 2.45
        print(x + 2)

        4.45
```

Allocating value 2.45 into variable 'x'
Referencing variable 'x'

Assignment operator:
The symbols may seem complicated, but it is simple after understanding the logic. For example, a += 10 is basically the same as a = a + 10. Thus, the equal sign is added to assign the value after calculation into the variable.

Operator	Description
+=	Assign after addition.
-=	Assign after subtraction.
*=	Assign after multiplication.
/=	Assign after division.
%=	Assign after modulus.
**=	Assign after exponential.

2. String

String in Python is a sequence of characters. Strings are created using quotation marks (also known as apostrophes), and you can use either a single or double quotes to create a string.

```
In [1]: print('Hello, World!')
        print("Hello, World!")
        print('P', 'y', 't', 'h', 'o', 'n')
        print("P", "y", "t", "h", "o", "n")
```

Strings

```
Hello, World!
Hello, World!
P y t h o n
P y t h o n
```

1) Creating quotation marks within the strings:
If you want to print a string with double quotes, you use a single quote to print it out. If you want to print a string with a single quote, you use double quotes to print it out.

```
In [1]: print('I say, "Hello, World!"')
        print("I think, 'Hello, World!'")

        I say, "Hello, World!"
        I think, 'Hello, World!'
```


2) Escape character:
Instead of the prior method, you can use an escape character backslash (\) to print out the apostrophes. If you write \" inside of a string, you can print out double quotation marks. You can also print out a single quotation mark with \'. In other words, the escape character lets the program recognize the quotation marks simply as they are, not as a symbol to make a string.

```
In [1]: print("\"Good day for fishing, aint it?\" the fisherman said.")
        print('\'Welcome to Honeywood!\' the villager greeted.')

        "Good day for fishing, aint it?" the fisherman said.
        'Welcome to Honeywood!' the villager greeted.
```

3) New line (\n) and Tab (\t):

\n splits strings into new lines. And \t creates an indentation in between the strings.

```
In [1]: print('Python\nProgramming')

Python
Programming

In [2]: print('Python\tProgramming')

Python  Programming
```


4) Repeating quotes 3 times (""" or '''):

In Python, you can just type out the strings in several lines and print it out using """ or '''. This is used to make the codes clean and easy to read, since it does not require the strings to be filled with \n.

```
In [1]: print("""These are some famous Shakespeare quotes
        But where there is true friendship, there needs none
        Keep thy friend under thy own life's key
        Most friendship is feigning, most loving is folly""")

        These are some famous Shakespeare quotes
        But where there is true friendship, there needs none
        Keep thy friend under thy own life's key
        Most friendship is feigning, most loving is folly
```

Focus **Do not be confused with multiline commenting using '''.** Since you put triple quotation marks inside the print() function, the above example prints out the part.

5) String concatenation operator (+):
If you want to connect multiple strings together, you can use a concatenation operator (+). It will connect the strings without a space. It works differently from the arithmetic operator (+) for numbers.

```
In [1]: print('Pyt' + 'hon')
        Python

In [2]: print('Python' + '!')
        Python!
```


6) String repetition operator (*):
If you want to repeat a string several times, you can use a repetition operator (*). It will repeat the string for the number of times indicated without a space.

```
In [1]: print('Python' * 3)
        PythonPythonPython

In [2]: print(3 * 'Python')
        PythonPythonPython
```

7) String indexing ([]):

We can choose a value within the string using [] operator, which is also known as indexing. Index refers to the position of each value within a string. Python utilizes the zero index, meaning that the value position always starts at 0.

H	E	L	L	O
[0]	[1]	[2]	[3]	[4]

```
In [1]: print('Hello'[0])
        print('Hello'[2])
        print('Hello'[4])

        H
        l
        o
```


8) String slicing ([:]):

We can select a range within a string using [:] operator, which is also known as slicing. When you insert the index into [:] operator, the last index will not be included.

H	E	L	L	O
[0]	[1]	[2]	[3]	[4]

```
In [1]: print('Hello'[1:4])

        ell

In [2]: print('Hello'[1:])

        ello
```

9) type() and len() functions:
type() function is used to find the type of the data. len() function is used to directly find the number of character(s) in a string, i.e. the length of the string.

```
In [1]: print(type('Hello'))
        <class 'str'>

In [2]: print(type(20))
        <class 'int'>

In [3]: print(len('Hello'))
        5
```

Focus 'str' refers to a string, and 'int' refers to an integer.

Learn more about...

Other string methods: Here are other string methods that are used practically in Python.

Method	Explanation
x.lstrip()	Trim left white spaces.
x.rstrip()	Trim right white spaces.
x.strip()	Trim white spaces from both sides.
x.replace(str1, str2)	Replace string from str1 to str2.
x.count(str)	Count the string pattern.

Learn more about...

Other string methods: Here are other string methods that are used practically in Python.

Method	Explanation
x.find(str)	Position of the first match. -1 if no match is found.
x.index(str)	Position of the first match. Error if no match is found.
x.join(str_list)	Join the string list items using x as connector.
x.split(y)	Split a string by y.
x.upper()	Convert to the upper case.
x.lower()	Convert to the lower case.

Learn more about...

Other string methods: Here are some examples of using the methods mentioned above.

```
In [1]: x = 'Python'

        # Convert the letters to uppercase
        print(x.upper())
```

```
PYTHON
```

```
In [2]: # Convert the letters to lowercase
        print(x.lower())
```

```
python
```

Learn more about...

Other string methods: Here are some examples of using the methods mentioned above.

```
In [3]: x = 'Python'

        # Look for the position of 'o'
        print(x.index('o'))
```

```
4
```

```
In [4]: # Find the position of 'z' (-1 means no match)
        print(x.find('z'))
```

```
-1
```


3. Number

There are three types of numbers in Python: integers, floating point numbers, and complex numbers. 'int' refers to an integer, 'float' refers to a floating point number, and 'complex' refers to a complex number.

```
In [1]: print(type(20))
```

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

```
In [2]: print(type(52.273))
```

```
<class 'float'>
```

```
In [3]: print(type(5+2j))
```

```
<class 'complex'>
```

4. input()

The input() function lets you ask a user for some data input. When the input function is executed, it tells the program to stop and wait for the user to key in the data. You can also allocate the value got from the user into a variable.

input('*a message you want to show to the user*')

x = input('*a message you want to show to the user*')


```
In [*]: input('Say hi to your friend. >> ')
        Say hi to your friend. >>
```

```
In [1]: input('Say hi to your friend. >> ')
        Say hi to your friend. >> Hello friend!
Out[1]: 'Hello friend!'
```

1) input() function's type:

It is important to understand that the result of input() function is always a string. Even if the value is a number, the result will always end of as a string type. Therefore, it is not possible for the user to use arithmetic operator (since it is not an integer or a floating point). Thus, we must understand the usage of int and float function to change the type of the result.

```
In [1]: x = input('Give me a number. >> ')
        print(type(x))

        Give me a number. >> 20
        <class 'str'>
```


2) Changing data type (str → int / str → float):

Casting is the act of changing a string into a number. There are two functions that you can use to cast. int() function changes strings into integers. float() function changes strings into floating point numbers. When you change strings into numbers, you can do arithmetic operations with them.

```
In [1]: int_num = int(input('Enter an integer. >> '))
        float_num = float(input('Enter a floating point number. >> '))

        Enter an integer. >> 3
        Enter a floating point number. >> 2.27

In [2]: print(int_num + float_num)

        5.27
```

3) Changing data type (int → str / float → str):
In vice versa, you can also change numbers into strings with the str() function.

```
In [1]: x = str(120)
        print(type(x))

        <class 'str'>

In [2]: y = str(30.258)
        print(type(y))

        <class 'str'>
```


Unit 4.

Operator

- 4.1 Arithmetic Operator
- 4.2 Comparison Operator
- 4.3 Logical Operator

1. Arithmetic Operator

You can do arithmetic operations with numbers in Python, which is similar to mathematics. First, + is an addition operator. For example in below, "x + y" will add y to x. Second, - is a subtraction operator. "x - y" will subtract y from x.

```
In [1]: x = 3
        y = 2
        print(x + y)

        5

In [2]: x = 3
        y = 2
        print(x - y)

        1
```


Third, * is a multiplication operator. "x * y" will multiply x by y. Fourth, ** is an exponential operator. "x ** y" will raise x to the power of y.

```
In [1]: x = 3
        y = 2
        print(x * y)

        6

In [2]: x = 3
        y = 2
        print(x ** y)

        9
```

If you divide two integers with /, the result will be in the form of real number. If you want to remove below the decimal point (for the result to be an integer), you should do floor division with //. Even when you divide two real numbers, you remove below the decimal point from the result which will always end in .0.

```
In [1]: x = 2
        y = 3
        print(x / y)

        0.6666666666666666

In [2]: x = 2
        y = 3
        print(x // y)

        0

In [3]: print(5.5 // 2.2)

        2.0
```


You can use the % operator to find the remainder of the division.

```
In [1]: print(5 % 2)

        1
```

Similar to mathematics, Python has an arithmetic operation priority. It first calculates multiplication and division and then addition and subtraction.

```
In [1]: print(2 + 2 - 2 * 2 / 2 * 2)
        print(2 - 2 + 2 / 2 * 2 + 2)

        0.0
        4.0

In [2]: print((5 + 3) * 2)
        print(5 + (3 * 2))

        16
        11
```


2. Comparison Operator

Comparison operators compare the relationship of two values.

Operator	Description
==	Equal to
!=	Not equal to
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to

True and False are Boolean data types. If the comparison result is correct, it is True. But if it is wrong, then it is False. You can compare both numbers and strings.

```
In [1]: print(3 > 1)
        True

In [2]: print(3 <= 1)
        False

In [3]: print(3 == 1)
        False

In [4]: print(3 != 1)
        True
```


When you compare strings, Python distinguishes between uppercase and lowercase letters.

```
In [5]: print('Python' == 'Python')
        True

In [6]: print('Python' == 'python')
        False
```

3. Logical Operator

Logical operators compare the logical value of two values. They are comprised of and, or, and not.

1) a and b:
It is True if the value of both a and b are the same. If at least one of the values is wrong, then it is False.

```
In [1]: print(True and True)
        True

In [2]: print(True and False)
        False

In [3]: print(False and True)
        False

In [4]: print(False and False)
        False
```


2) a or b:
If at least one value between a and b is True, then it is True. In other words, if only one is False, then it is True. If both values are False, then it is False.

```
In [1]: print(True or True)
        True

In [2]: print(True or False)
        True

In [3]: print(False or True)
        True

In [4]: print(False or False)
        False
```

3) not a:
not overturns the logical value. For example, not True becomes False.

```
In [1]: print(not True)
        False
```

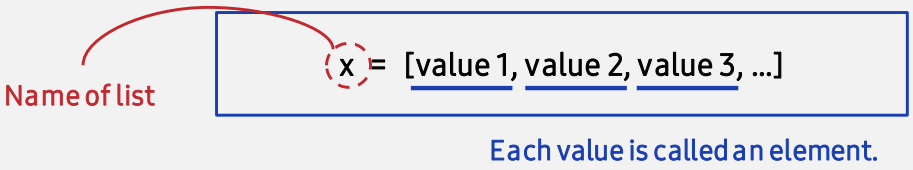
```
In [2]: print(not False)
        True
```


Unit 5.
Data Structure

- 5.1 List
- 5.2 Tuple
- 5.3 Index
- 5.4 Dictionary
- 5.5 Set

1. List

You can save multiple values in a single variable, which is called a list. You must put them in square brackets [], and separate them with commas. You can change, add, or delete elements that are already saved.



You can create an empty list as below. Similar to declaring a data type in the earlier unit, we can first make an empty list before we fill it up with values or use it in Python.

```
In [1]: x = []
```


As we learned before, we can allocate different data type as a value into a variable. Similarly, you can put strings, integers, Booleans, and any Python data type into a list's elements. You can also mix and use them.

```
In [1]: x = [25, 46.1, 98.2, 117.5]
        print(x)

        [25, 46.1, 98.2, 117.5]

In [2]: y = ['Python', 'Computer', 'Programming']
        print(y)

        ['Python', 'Computer', 'Programming']

In [3]: z = ['python', 20, 13+8, True]
        print(z)

        ['python', 20, 21, True]
```

1) len() function:

Similar to the string function, len() function for list is used to directly find the number of element(s) in a list, i.e. the length of the list. It is a built-in Python function.

```
In [1]: x = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
        print(len(x))

        10
```


2) x.append(val) method:

This method adds a specific value into a list, appending it right after the end value of the list.

```
In [1]: x = [0, 1, 2, 3, 4]
        x.append(5)

        print(x)

        [0, 1, 2, 3, 4, 5]
```

3) x.append(list) method:
You can put a list as an element inside a list. This method adds another list into a list, appending it right after the end value of the list.

```
In [1]: x = [0, 1, 2, 3, 4]
        x.append([5, 6])

        print(x)
```

```
[0, 1, 2, 3, 4, [5, 6]]
```


Learn more about...

Other list methods: Here are other list methods that are used practically in Python.

Method	Explanation
x.insert(pos, val)	Insert a value at a specified position.
x.remove(val)	Remove a specific value from a list.
x.pop()	Remove and return the last value of a list.
x.count(val)	Count the appearances of a value in a list.

Learn more about...

Other list methods: Here are other list methods that are used practically in Python.

Method	Explanation
x.extend(y)	Extend the list x with another list y. Means x = x + y.
x.sort()	Sorting of a list.
x.reverse()	Reversal of the entry ordering.
x.index(val)	Looks for the position of a specific value. Error if no match is found.

Learn more about...

Other list methods: Here are some examples of using the methods mentioned above.

In [1]:
```
x = [3, 1, 5, 2, 4]

# Sort the list object (with lasting effect)
x.sort()
print(x)
```

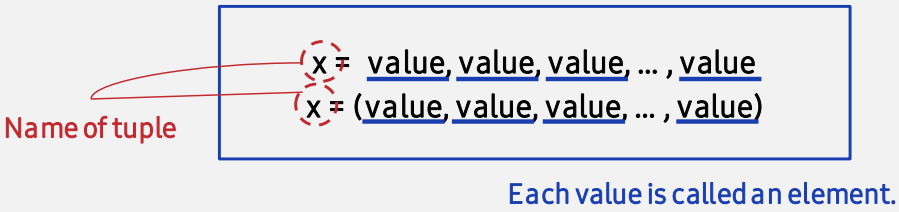
```
[1, 2, 3, 4, 5]
```

In [2]:
```
# Reverse the order of the list
x.reverse()
print(x)
```

```
[5, 4, 3, 2, 1]
```

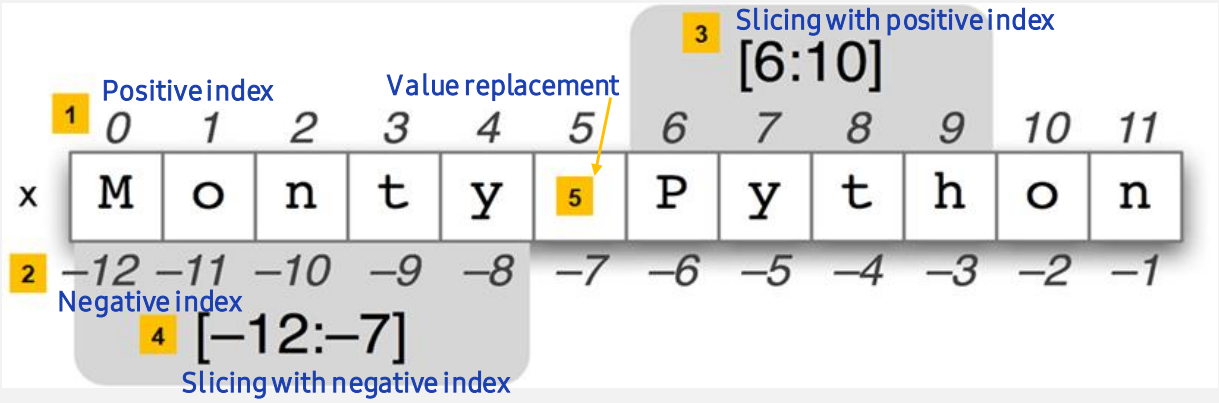
2. Tuple

The basic concept is the same as list, but you cannot change, add, or delete elements that are already saved. Therefore, it is easier to think of it as the reading mode of list. While list is in the square brackets, tuples are in parentheses. It is also okay to drop the parentheses. You can separate each value with commas.



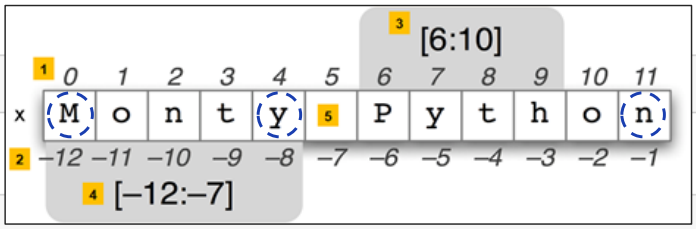
3. Index

Index is a method to approach a value in a specific place within a list, tuple, string, etc.



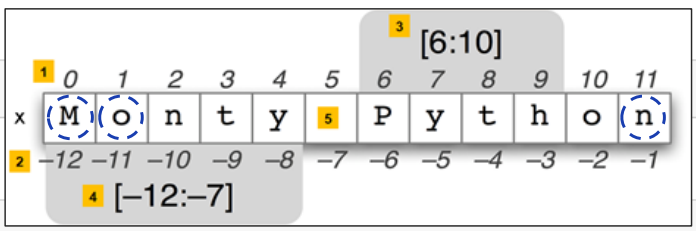
1) Positive index (sequence object[index number]):
You assign each index number with an integer from the far left starting from 0 and use it to access the element.

```
In [1]: x = ['M', 'o', 'n', 't', 'y', ' ', 'P', 'y', 't', 'h', 'o', 'n']
        print(x[0])

        M

In [2]: print(x[4])

        y

In [3]: print(x[11])

        n
```



2) Negative index (sequence object[negative index number]):
If the index number is negative, you can access the element from the back. You assign each index number with an integer from the far right starting from -1.

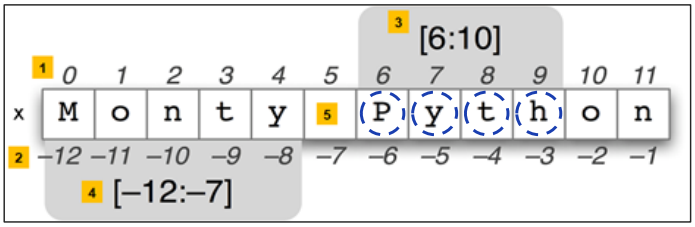
```
In [1]: x = ['M', 'o', 'n', 't', 'y', ' ', 'P', 'y', 't', 'h', 'o', 'n']
        print(x[-1])

        n

In [2]: print(x[-11])

        o

In [3]: print(x[-12])

        M
```



3) Slicing with positive index (sequence object[first index:last index]):

You can cut out a portion of a sequence object and use it, which is called "slice." Note that the last index's value is not included in the range of values actually imported. In the example below, you can see that the index value of 10 is 10. But in an actual code, you can check that it is only up to 9.

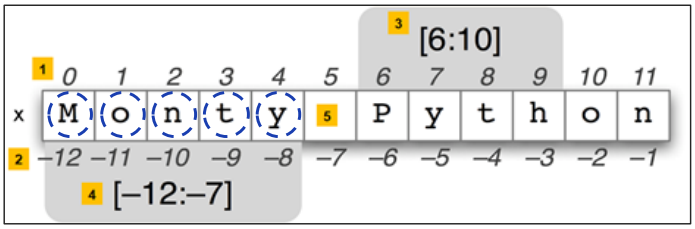
```
In [1]: x = ['M', 'o', 'n', 't', 'y', ' ', 'P', 'y', 't', 'h', 'o', 'n']
        print(x[6:10])

        ['P', 'y', 't', 'h']
```



4) Slicing with negative index (sequence object[negative first index:negative last index]):

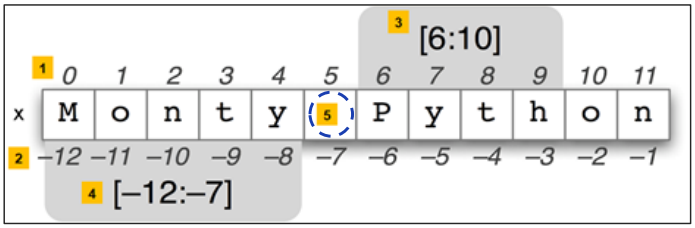
You can also bring the range with negative indexes.

```
In [1]: x = ['M', 'o', 'n', 't', 'y', ' ', 'P', 'y', 't', 'h', 'o', 'n']
        print(x[-12:-7])

        ['M', 'o', 'n', 't', 'y']
```



5) Value replacement (sequence object[index number] = value):
It replaces element values of corresponding index numbers with new values or assigns new values.

```
In [1]: x = ['M', 'o', 'n', 't', 'y', ' ', 'P', 'y', 't', 'h', 'o', 'n']
        x[5] = 's'
        print(x)

        ['M', 'o', 'n', 't', 'y', 's', 'P', 'y', 't', 'h', 'o', 'n']
```



4. Dictionary

We use an object called dictionary, which is a series data, to save related values in groups. Dictionary and list are similar because both of them are collections that contain references to other objects. While we access an item by its *location* (using an index) with a list, we access it by its *name* (called a key) with a dictionary.

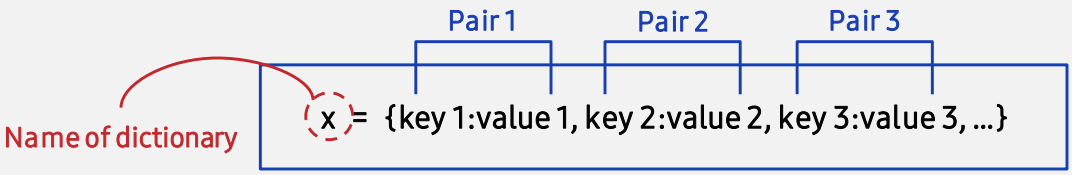
python_list

Index	Value
0	'Eggs'
1	'Milk'
2	'Cheese'
3	'Yogurt'
4	'Butter'

python_dictionary

Key	Value
'Eggs'	2.59
'Milk'	3.19
'Cheese'	4.80
'Yogurt'	1.35
'Butter	2.59

Here is the way to make a dictionary. A key and a value should be 1:1, and we call it as a key-value-pair. The name of keys in a dictionary should not be duplicated. The name of keys can be an integer, a real number, or Booleans, either independently or in a combination.




```
In [1]: dic = {'Red':'FF0000', 'Green':'00FF00', 'Blue':'0000FF'}
        print(dic)

        {'Red': 'FF0000', 'Green': '00FF00', 'Blue': '0000FF'}
```


1) Dictionary[key]:
In this way, you can obtain the value of a specific key.

```
In [1]: dic = {1:'a', 2:'b'}
        print(dic[1])

        a

In [2]: print(dic[2])

        b
```

2) Dictionary[key] = value:
In this way, you can assign a value to a specific key.

```
In [1]: dic = {1:'a', 2:'b'}
        dic[1] = 'Hello'
        print(dic)

        {1: 'Hello', 2: 'b'}
```


5. Set

The elements inside set are not repeated and duplicated. Each element is different. There is no concept of ordering (sorting) nor indexing. The reason for using set is to understand if a value belongs to it or not.

```
In [1]: a = set() # an empty set
        b = set([1, 2, 3, 3, 3, 4, 5]) # a set is created from a list.
        print(b)

        {1, 2, 3, 4, 5} ← No repeated elements

In [2]: print(type(b))

        <class 'set'>
```

Here are a couple of examples of set methods.

In [1]:
```
a = set([1, 2, 3, 4, 5])

# Add a value to the set
a.add(6)
print(a)
```

```
{1, 2, 3, 4, 5, 6}
```


In [2]:
```
# Add several values to the set
a.update([7, 8, 9])
print(a)
```

```
{1, 2, 3, 4, 5, 6, 7, 8, 9}
```

In [3]:
```
# Remove a specific value from the set
a.remove(9)
print(a)
```

```
{1, 2, 3, 4, 5, 6, 7, 8}
```

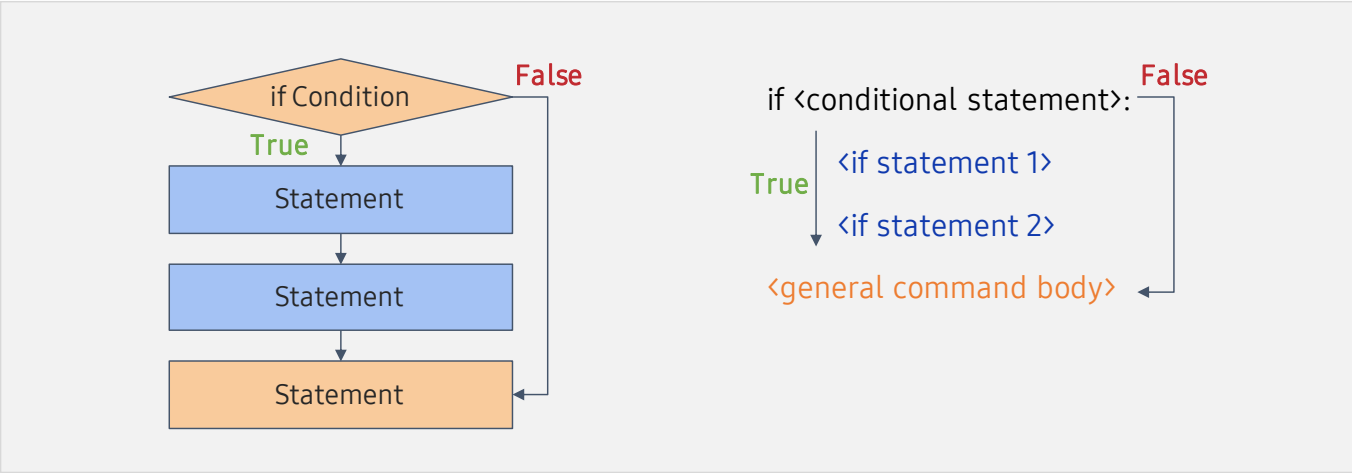
Unit 6.

Conditional Statement

- 6.1 if Statement
- 6.2 else
- 6.3 elif

1. if Statement

Conditional statement is a grammar that executes the code whenever it fulfills a specific condition. First, you need to create a conditional statement between if and a colon sign. Then, you write the code (statement) in the next lines with indentation. The code will be executed if the condition is true.



When you write a conditional statement, you must be careful not to use = sign, which is used for variable assignment. You need to you == sign, which is a comparison operator.

```
In [1]: x = 10

        if x == 10:
            print('True.')

        print('End.')
```

```
True.
End.
```

```
In [2]: x = 20

        if x == 10:
            print('True.')

        print('End.')
```

```
End.
```

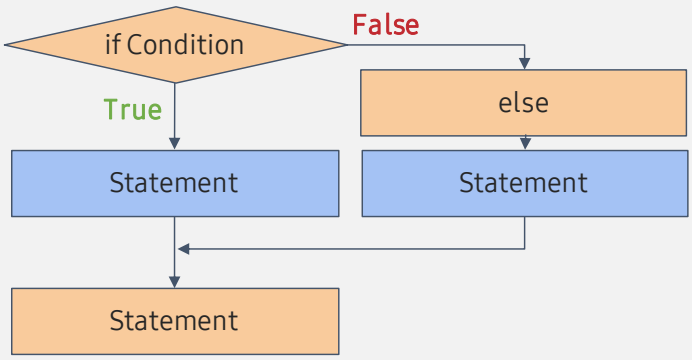

You can make another conditional statement inside a conditional statement. This is called a nested if statement. It is also possible to have two or more nested if statements. You need to be careful of indentation.

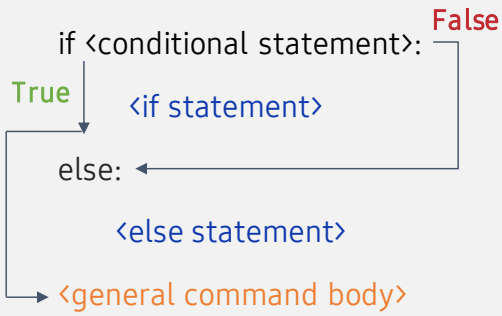
```
In [1]: x = int(input('Enter your favorite number. '))

        if x > 10:
            print('It is above 10.')
            if x == 20:
                print('It is 20.')
            if x == 30:
                print('It is 30.')

        print('End.')
```

```
Enter your favorite number. 20
It is above 10.
It is 20.
End.
```

2. else

else executes the code when the conditional statement for if is not satisfied, i.e. when if conditional statement is false.





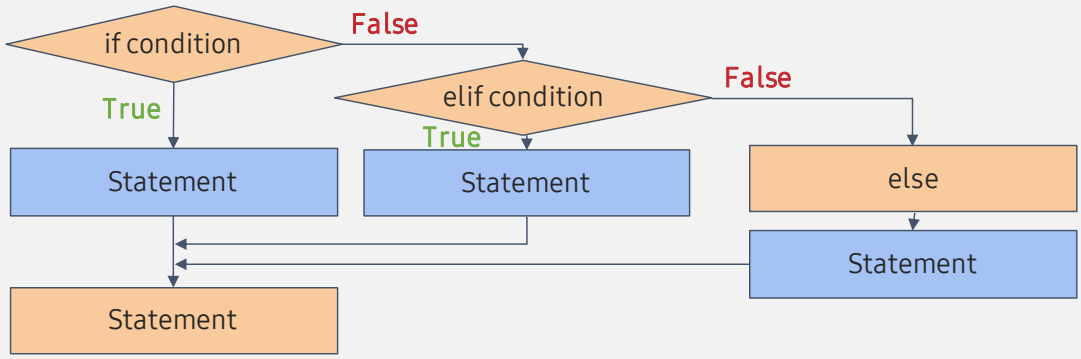
The code under the else statement must also be indented like if.

In [1]:
```
x = int(input('Enter your favorite number. '))

if x == 10:
    print('It is 10.')
else:
    print('It is not 10.')
```

```
Enter your favorite number. 20
It is not 10.
```

3. elif

elif is a short for "else if," and it enables us to make multiple conditional statements. Each elif condition can have different codes beneath it. Note that else and elif cannot be used on their own, and they can be used only in the form of if~else or if~elif~else.



You can have multiple elif statements between if and else statements.

In [1]:
```
x = int(input('Enter your favorite number. '))

if x == 10:
    print('It is 10.')
elif x == 20:
    print('It is 20.')
elif x == 30:
    print('It is 30.')
else:
    print('It is not 10, 20, or 30.')
```

```
Enter your favorite number. 20
It is 20.
```

Unit 7.
Loop

- 7.1 range()
- 7.2 for Loop
- 7.3 while Loop
- 7.4 break, continue

1. range()

range() is a function that creates consecutive numbers. It can be used to create numbers as much as the number of times assigned, assign a start value and an end value, or set an increasing value.

1) range(number of times):
Number creation starts from 0 and increases by 1 up to the number of times, which is not included.
For example, range(10) does not include 10 and creates 0 through 9. If you combine it to a list, you can use it as below.

```
In [1]: x = list(range(10))
        print(x)

        [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

2) range(first value, end value):
Number creation starts from the first value and increases by 1 up to the end value, which is not included. For example, range(0, 10) does not include 10 and creates 0 through 9.

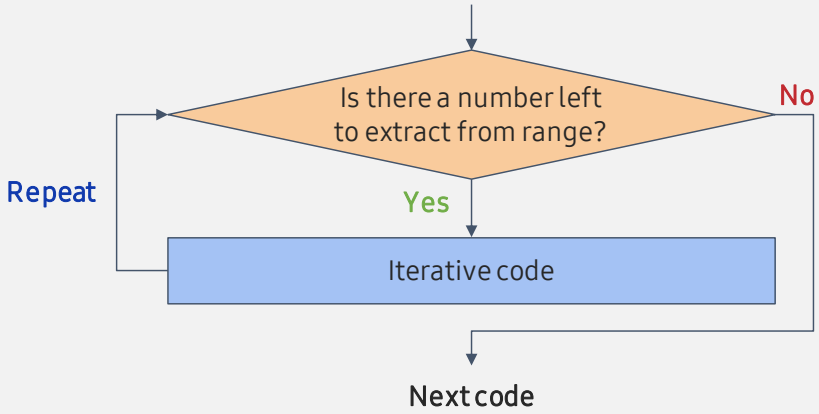
```
In [1]: x = list(range(0, 10))
        print(x)

        [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```


3) range(first value, end value, margin of increase or decrease):
Number creation starts from the first value and increases or decreases by the margin to the end value, which is not included.

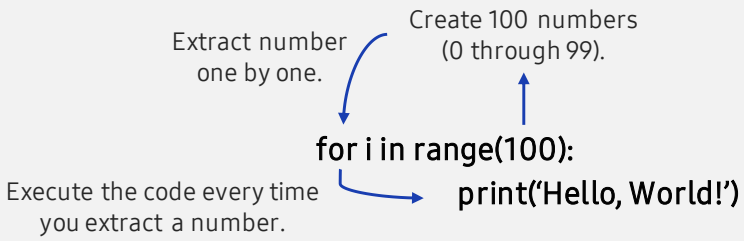
```
In [1]: x = list(range(0, 20, 1))
        print(x)

        [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]

In [2]: y = list(range(2, 20, 6))
        print(y)

        [2, 8, 14]

In [3]: z = list(range(30, 2, -2))
        print(z)

        [30, 28, 26, 24, 22, 20, 18, 16, 14, 12, 10, 8, 6, 4]
```

2. for Loop

Loops are used to execute repetitive codes in a more efficient way. For example, for loop sets the number of times to repeat in range, then it will repeat the code for that number of times.



As the example below, the for loop sets the number of times to repeat in range, then write *in* and *a variable* (ex. i) in front of it. Put the colon at the end, then on the next line place the code to repeat. The repetitive statement after for must be indented like an if statement.



Here are the multiple ways of using range() in for loops.

In [1]:
```
for i in range(3): # repeats from 0 to 2
    print('Hello, World!', i)
```

```
Hello, World! 0
Hello, World! 1
Hello, World! 2
```

In [2]:
```
for i in range(5, 9): # repeats from 5 to 8
    print('Hello, World!', i)
```

```
Hello, World! 5
Hello, World! 6
Hello, World! 7
Hello, World! 8
```


In [3]:
```
for i in range(0, 10, 2): # increases by 2 from 0 to 8
    print('Hello, World!', i)
```

```
Hello, World! 0
Hello, World! 2
Hello, World! 4
Hello, World! 6
Hello, World! 8
```

In [4]:
```
# Repeat the loop for the number of times entered
count = int(input('Enter the number of times to repeat. >> '))
for i in range(count):
    print('Hello, World!', i)
```

```
Enter the number of times to repeat. >> 3
Hello, World! 0
Hello, World! 1
Hello, World! 2
```

In [5]:
```
# Declare a new list
# Use len() method to decide the number of repetition
x = [0, 1, 2, 3, 4]

for i in range(len(x)):
    print('Hello, World!', i)
```

```
Hello, World! 0
Hello, World! 1
Hello, World! 2
Hello, World! 3
Hello, World! 4
```


We can set up the number of times to repeat the for loop not only with range() but also with other methods. One another method is using elements in a list to repeat the for loop.

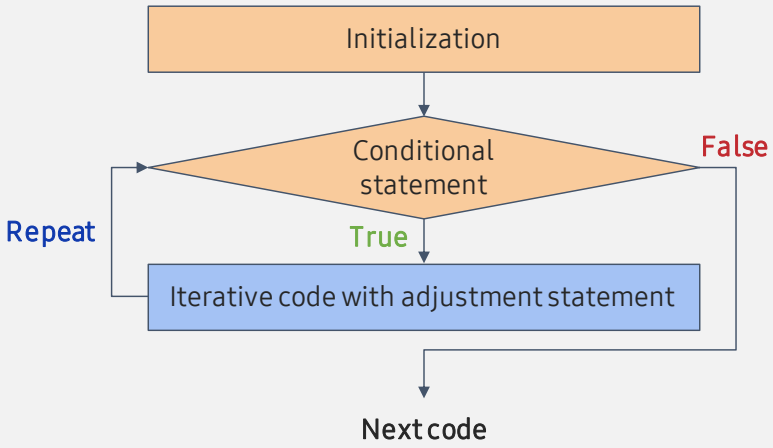
```
x = [value 1, value 2, value 3, ...]

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

Here is a way of using list in for loop. It will print out each element of the list one by one as it goes through the for loop.

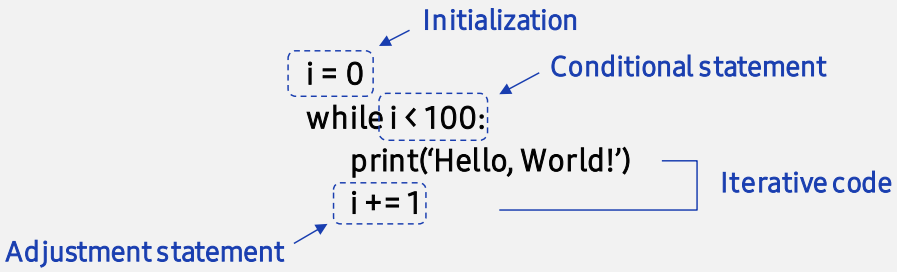
```
In [1]: x = ['Say', 'Hello', 'to', 'Python']

        for i in x:
            print(i)

        Say
        Hello
        to
        Python
```


3. while Loop

while loop repeats the code until a certain conditional statement is satisfied. Note that it is mandatory to have a conditional statement and adjustment statement, and you also need to initialize the variable that you will use in the conditional statement.



In the code below, i starts from 1 and repeats only when i < 100, and the repetition ends when it becomes 100. i increases from 0 to 99 through += 1. When it becomes 100, the repetition stops, and the program goes to the next code out of the loop.



Here are the examples of while loop with different initial values and adjustment statements.

In [1]:
```
# Start the initial value from 1, and increase it by 2
# Repeat the loop for 5 times

i = 1

while i < 10:
    print('Hello, World!', i)
    i += 2
```

```
Hello, World! 1
Hello, World! 3
Hello, World! 5
Hello, World! 7
Hello, World! 9
```

In [2]:
```
# Start the initial value from 5, and decrease it by 1
# Repeat the loop for 5 times

i = 5

while i > 0:
    print('Hello, World!', i)
    i -= 1
```

```
Hello, World! 5
Hello, World! 4
Hello, World! 3
Hello, World! 2
Hello, World! 1
```


4. break, continue

There are special keywords that can be only used inside the loops: break and continue. First, the break statement terminates the loop. It is usually used in a infinite loop and to stop the internal repetition. The below example shows how to make an infinite loop with Boolean condition and how the break statement stops it.

In [1]:
```
# Declare a variable
i = 0

# Start an infinite loop
while True:
    print(i)
    i += 1
    if i == 3:
        break
```

```
0
1
2
```

Next, the continue statement skips the remaining code in the current iteration and moves onto the next iteration. It does not terminate the loop itself. In the below example, you can see how it skips the print() line if the value (i) is an even number, which has a remainder of 0 when divided by 2.

```
In [1]: # Print out only odd numbers
        for i in range(10):
            if i % 2 == 0:
                continue
            print(i)
```

```
1
3
5
7
9
```


Unit 8.

Function

- 8.1 Function
- 8.2 Parameter
- 8.3 Return

1. Function

In Python, you can use a function to lessen repetitive codes. First, you need to define a function as below. Then you can use it by calling the function later in the code. Make sure to have indentation for the function definition part.

```
def function_name():
    statement 1
    statement 2
```

Indentation

2. Parameter

When you define a function, you might want to use several data in the function. These data are defined as parameters and indicated in parentheses next to the function name. It is optional to put parameters when defining a function.

```
def function_name(parameter 1, parameter 2, ... ):
    statement 1
    statement 2
```

Indentation

```
print(*value, sep=' ', end='\n', file=sys.stdout, flush=False)
```

These are all considered parameters.

1) Mutable parameter:
print() function is an example of mutable parameter where there are no limit in the amount of parameters you can use. You need to put an asterisk in front of the mutable parameter. There are *two rules* using mutable parameter. First, normal parameters must come before mutable parameter. Second, you can only use 1 mutable parameter.

Indentation

```
def function_name(parameter 1, parameter 2, ..., *mutable parameter):
    → statement 1
    → statement 2
```


2) Default parameter:
The default parameter is a way to set default value for function parameters with undefined value. There is a rule using default parameter. Normal parameters must come before default parameters. In the below example, n is set as 2 by utilizing a default parameter, which means that the function will print the result 2 times.

In [1]:
```
def print_n_times(value, n = 2): # n = 2 is a default parameter
    for i in range(n):
        print(value)

print_n_times('Hello!')
```

```
Hello!
Hello!
```

3) When a default parameter comes before a mutable parameter:
Before going into keyword parameter, we must understand if we can use both mutable and default parameter in one function. The problem in the below example is that 'Hello' went into n, and TypeError occurs because range() cannot execute strings (only numbers). Thus, we can conclude that default parameter cannot come before mutable parameter.

In [1]:
```
# If a default parameter comes before a mutable parameter
def print_n_times(n = 2, *values):
    for i in range(n):
        for value in values:
            print(value)

print_n_times('Hello', 'Fun', 'Python')
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-1-edeb32a3d0e2> in <module>
      5             print(value)
      6
----> 7 print_n_times('Hello', 'Fun', 'Python')
```


4) When a mutable parameter comes before a default parameter:
In the below example, the result has no error because strings can go into *values (mutable parameter).

In [1]:
```
# If a mutable parameter comes before a default parameter
def print_n_times(*values, n = 2):
    for i in range(n):
        for value in values:
            print(value)

print_n_times('Hello', 'Fun', 'Python')
```

```
Hello
Fun
Python
Hello
Fun
Python
```

5) Keyword parameter:
So what if you want to be able to change the default parameter? This is where keyword parameter comes in. Giving a parameter another value directly is the act of utilizing a keyword parameter.

```
In [1]: # The default parameter sets the range as 2
        def print_n_times(*values, n = 2):
            for i in range(n):
                for value in values:
                    print(value)

        # This n = 3 changes the range to 3
        print_n_times('Hello', 'Fun', 'Python', n = 3)
```

```
Hello
Fun
Python
Hello
Fun
Python
Hello
Fun
Python
```


3. Return

The ability to return the result back to the user is called the return statement. Return is also used as a keyword in functions. This keyword allow the function to go back to the previous location (previous line), changing the ending point of a function.

```
In [1]: def return_test():
            print("Location A")
            return
            print("Location B")

        return_test()
```

```
Location A
```

Focus The return function ignores the statement below it and returns back to the top.

Unit 9.

Error

1. SyntaxError

SyntaxError is an error due to the problem in the code, so the program itself does not execute at all. You need to look into the code you wrote when you see this SyntaxError.

SyntaxError

```
In [1]: print("He said, "Hello."")
          File "<ipython-input-1-f68bf0ca980d>", line 1
            print("He said, "Hello."")
                           ^
        SyntaxError: invalid syntax
```

2. TypeError

TypeError occurs when you try to do the operation with different data types. As mentioned before, the result of the input() function is always a string. Therefore, if you try to do the operation with a string and an integer together, you will have this TypeError.

TypeError

```
In [1]: string = input('Type a number. >> ')
        print(string + 100)

        Type a number. >> 20

        ---------------------------------------------------------------------------
        TypeError                                 Traceback (most recent call last)
        <ipython-input-1-9e1713219c88> in <module>
              1 string = input('Type a number. >> ')
        ----> 2 print(string + 100)

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


3. IndexError

IndexError is an error when an index is out of range. It occurs when you select an index or character that exceeds the number of elements in a list or string.

IndexError

```
In [1]: print('Hello'[10])

        ---------------------------------------------------------------------------
        IndexError                                Traceback (most recent call last)
        <ipython-input-1-4a642042c2bf> in <module>
        ----> 1 print('Hello'[10])

        IndexError: string index out of range
```

End of Document

Samsung Innovation Campus

SAMSUNG

Together for Tomorrow!
Enabling People
Education for Future Generations