SAMSUNG

Together for Tomorrow!
Enabling People
Education for Future Generations

Samsung Innovation Campus

Artificial Intelligence Course

Together for Tomorrow!
Enabling People
Education for Future Generations

Chapter 2.

Python Programming

UNIT 3.

Python III

Samsung Innovation Campus

UNIT 3.
Python III

What this unit is about:

- In this unit you will learn about structures that control the flow of a program.
- You will write our own functions and make the code more reusable.

Expected outcome:

- Ability to write short codes that involve control structures, functions and file input and

How to check your progress:

- Coding Exercises.
- Quiz.

Chapter 2.

Python Programming

Together for Tomorrow!
Enabling People
Education for Future Generations

UNIT 3. Python III

3.1. Control Structures.

3.2. Python Functions.

3.3. Python Input and Output.

Unit 4. Python IV

4.1. Classes and Objects.

4.2. Exception Handling.

Unit 5. Python V

5.1. Algorithms.

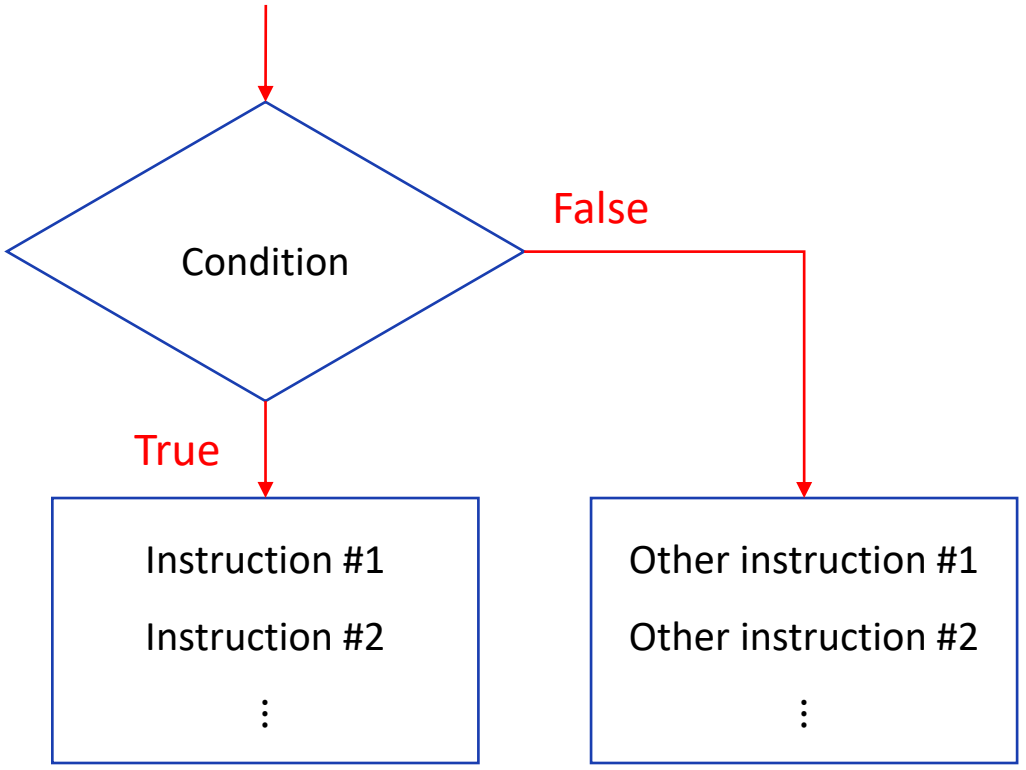
5.2. Data Structures.

5.3. Working with Files.

5.4. Working with Excel, Word, PDF Documents.

Conditional Structure (1/8)

Conditional structure if-else:



Conditional Structure (2/8)

Conditional structure if-else:

```
if <Condition>:
    <Instruction #1>                    # Indented code block.
    <Instruction #2>
        ...
else:                                   # Run code block below when <Condition> is False.
    <Instruction #3>                    # Indented code block.
    <Instruction #4>
        ...
```

Conditional Structure (3/8)

Conditional structure if-elif-else:

```
if <Condition A>:
    <Instruction #1>                    # Indented block.
    <Instruction #2>
        ...
elif <Condition B>:                     # Evaluate <Condition B> when <Condition A> is False.
    <Instruction #3>                    # Indented code block.
    <Instruction #4>
        ...
else:                                   # Run code block below when <Condition B> is also False.
    <Instruction #5>                    # Indented code block.
    <Instruction #6>
        ...
```

Conditional Structure (4/8)

Comparison operators can be used to construct conditional expressions.

Comparison Operator	Explanation
x < y	True if x is less than y.
x > y	True if x is greater than y.
x == y	True if x and y are equal.
x != y	True if x and y are unequal.
x >= y	True if x is greater than or equal to y.
x <= y	True if x is less than or equal to y.

Conditional Structure (5/8)

Membership operators can be used to construct conditional expressions.

Membership Operator	Explanation
x in y	True if x is present in y.
x not in y	True if x is not present in y.

Conditional Structure (6/8)

Membership operators can be used to construct conditional expressions.

```
In[1] : 1 in [1, 2, 3]                    # Whether 1 is present in the list [1, 2, 3].
Out[1]: True
In[2] : 1 not in (1, 2, 3)                # Whether 1 is present in the tuple (1, 2, 3).
Out[2]: False
In[3] : 'o' in 'Python'                   # Whether letter 'o' is present in the string 'Python'.
Out[3]: True
```

Conditional Structure (7/8)

Identity operators can be used to construct conditional expressions.

Identity Operator	Explanation
x is y	True if x and y are the same object.
x is not y	True if x and y are not the same object.

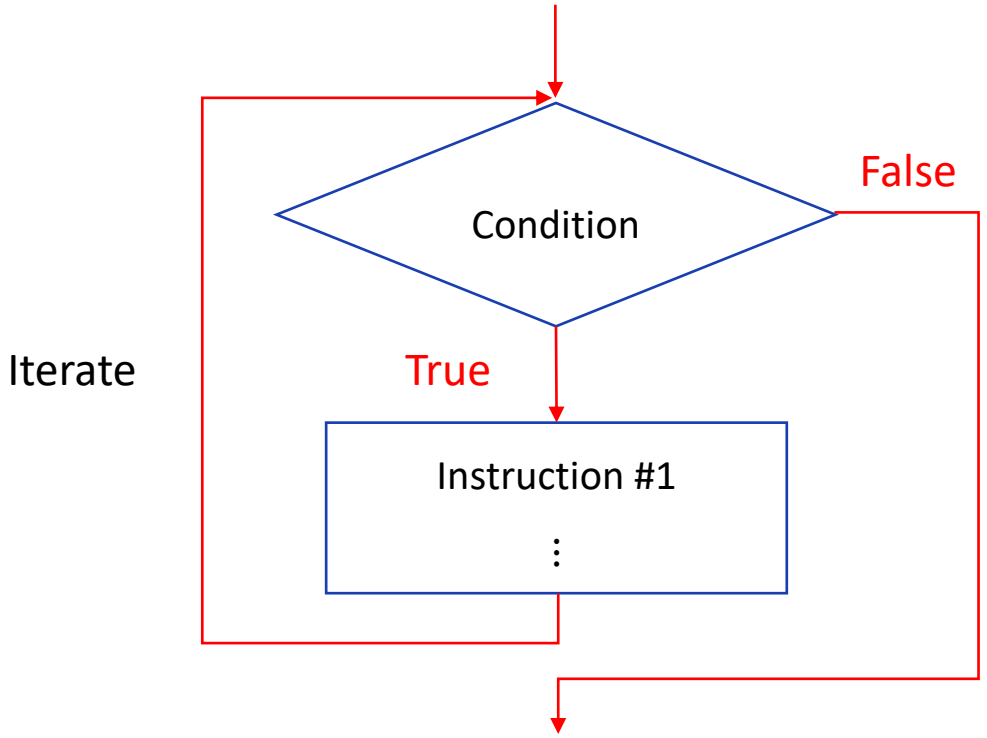
Conditional Structure (8/8)

Boolean operators can be used to construct conditional expressions.

Boolean Operator	Explanation
x and y	True if both x and y are True.
x or y	True if either x or y is True.
not x	True if x is False or vice versa.

Loop Structure (1/9)

Looping with while:



Loop Structure (2/9)

Looping with while:

```
while <Condition>:
    <Instruction #1>                                    # Indented code block.
    <Instruction #2>
        ...
```

Loop Structure (3/9)

Looping with while: an example.

```
In[1] : i = 0
In[2] : while i < 4:                    # Exit the loop as soon as i becomes 4 .
   ... :     print(i)
   ... :     i += 1                     # i increases within the code block.
0
1
2
3
```

Loop Structure (4/9)

Looping with while: break out of the loop.

```
while <Condition A>:
    <Instruction #1>
    <Instruction #2>
        ...
    if <Condition B>:
        break                                   # Exit the loop when <Condition B> is True.
    <Instruction #3>
        ...
```

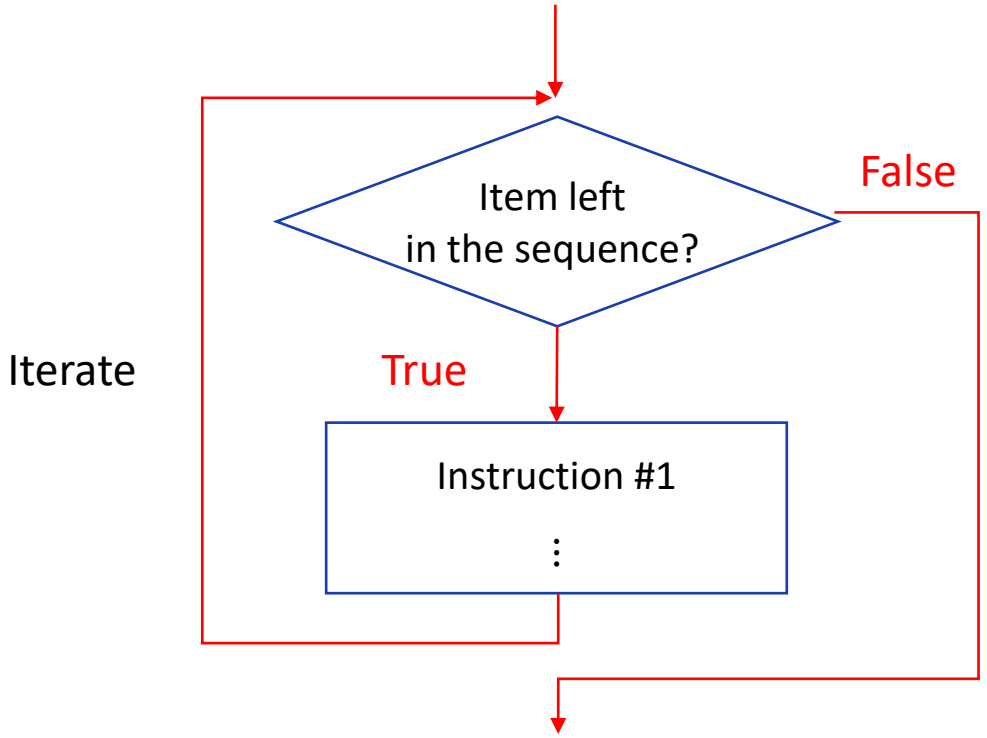
Loop Structure (5/9)

Looping with while: continue with the next iteration.

```
while <Condition A>:
    <Instruction #1>
    <Instruction #2>
        ...
    if <Condition B>:
        continue                    # If <Condition B> is True, jump to the next iteration.
    <Instruction #3>
        ...
```

Loop Structure (6/9)

Looping with for:



Loop Structure (7/9)

Looping with for:

```
for <variable> in <list, tuple, set, string, etc.>:          # Use of an iterable object.
    <Instruction #1>                                         # Indented code block.
    <Instruction #2>
        ...
```

Loop Structure (8/9)

Looping with for: an example.

```
In[1] : X = ['You', 'need', 'Python!']
In[2] : for x in X:
   ... :     print(x)
You
need
Python!
```

Loop Structure (9/9)

Looping with for: an example.

```
In[1] : sum = 0
In[2] : for x in range(1, 11):                    # Iterate from 1 to 10.
   ... :     sum += x
   ... :
In[3] : print(sum)
55
```

Coding Exercise #0105

Follow practice steps on 'ex_0105.ipynb'

Python Programming

Python Functions (1/17)

Reasons to use function:

- To keep the program organized and easy to read
- To make the code more reusable.
- To reduce the development time.
- To facilitate the maintenance.

Python Functions (2/17)

Python built-in functions:

- They are always available.
- The don’t need to be called in or imported.

Python Functions (3/17)

Some of Python's built-in functions:

abs	enumerate	int	max	range
all	eval	isinstance	min	sorted
any	filter	lambda	oct	str
chr	hex	len	open	tuple
dir	id	list	ord	type
divmod	input	map	pow	zip

For a full list and detailed explanation, please go to https://docs.python.org/3/library/functions.html

Python Functions (4/17)

Python functions can be organized in libraries or modules:

- The libraries should be imported for the functions to be available.
- Some of the most used libraries are: os, sys, numpy, pandas, scipy, matplotlib, etc.

Python Functions (5/17)

Python functions can be organized in libraries or modules:

```
In[1] : import os                                          # Import os library.
In[2] : os.getcwd()                                        # Current working directory.
Out[2]: ' C:\\Users\\MyComputer \\Documents\\Python Scripts '
In[3] : os.chdir( ' .. ' )                                 # Change the working directory.
In[4] : os.getcwd()                                        # Current working directory.
Out[4]: ' C:\\Users\\MyComputer \\Documents '
In[5] : os.listdir( ' . ' )                                # List the current directory.
['.ipynb_checkpoints',
'Thumbs.db',
 'Untitled.ipynb',
 'Untitled1.ipynb',
 'Untitled2.ipynb',
 'Untitled3.ipynb']
```

Python Functions (6/17)

User defined functions (UDFs):

```
def  <Function name>(Argument #1, Argument #2, ...):
    <Instruction #1>                                          # Indented code block.
    <Instruction #2>
        ...
      return <Return value>
```

Python Functions (7/17)

User defined functions (UDFs): an example.

```
In[1] : def times2(a):
   ... :     x = 2*a
   ... :     return x
   ... :
In[2] : times2(11)
Out[2]: 22
In[3] : times2(7)
Out[3]: 14
```

Python Functions (8/17)

User defined functions (UDFs):

- When the number of arguments is undefined, the variable should be preceded with an asterisk.

```
In[1] : def sum(*vals):                    # Notice the asterisk.
   ... :     total = 0
   ... :     for x in vals:                # vals is a tuple.
   ... :         total += x
   ... :     return total
   ... :
In[2] : sum(1,2,3)
Out[2]: 6
In[3] : sum(1, 2, 3, 4, 5)
Out[3]: 15
```

Python Functions (9/17)

User defined functions (UDFs):

- Functions may return no value at all, one value or a tuple (more than one value).

```
In[1] : def ArithmeticOprs(a,b):
   ... :     return a + b, a – b, a * b, a/b
   ... :
In[2] : type(ArithmeticOprs(4,2))
Out[2]: tuple                                   # A tuple is returned.
In[3] : ArithmeticOprs(4,2)
Out[3]: (6, 2, 8, 2)                                    # A tuple is an immutable sequence of values.
```

Python Functions (10/17)

User defined functions (UDFs):

- It is possible to set default values for the arguments.

```
In[1] : def ArithmeticOprs(a=2, b=1):
   ... :     return a + b, a – b, a * b, a/b
   ... :
In[2] : ArithmeticOprs()                    # Use all the default values.
Out[2]: (3, 1, 2, 2)
In[3] : ArithmeticOprs(3)                   # Use the default value for b.
Out[3]: (4, 2, 3, 3)
In[3] : ArithmeticOprs(b=3)                 # Argument matched by explicit name.
Out[3]: (5, -1, 6, 0)
```

Python Functions (11/17)

Variable scope:

- Variables defined inside a function have local scope. Their scope is limited within the function.
- When two variables with the same name exist inside and outside of the function, they are treated as different variables.

Python Functions (12/17)

Variable scope:

```
In[1] : result = 333
In[2] : def average(*a):
   ... :         result = 0                          # result is a local variable.
   ... :     for x in a:
   ... :         result += x
   ... :     result /= len(a)
   ... :     return result
   ... :
In[3] : average(1,2,3,4,5)
Out[3]: 3
In[4] : result                                  #The variable defined defined outside the function.
Out[4]: 333
```

Python Functions (13/17)

Variable scope:

- A variable can be declared global within a function to make it available outside.

```
In[1] : result = 0
In[2] : def average(*a):
   ... :         global result                    # result is declared a global variable.
   ... :     for x in a:
   ... :         result += x
   ... :     result /= len(a)
   ... :     return                                # There is no return value.
   ... :
In[3] : average(1,2,3,4,5)
In[4] : result
Out[4]: 3
```

Python Functions (14/17)

Lambda functions:

- Does not require the keyword def.
- Use lambda functions where the keyword def cannot be placed.

```
In[1] : def makeMyFunc(a):
   ... :     return lambda x: a*x
   ... :
In[2] : myFunc = makeMyFunc(3)
In[3] : type(myFunc)                                    # Returned object is a function.
Out[3]: function
In[4] : myFunc(4)
Out[4]: 12
```

Python Functions (15/17)

Custom Modules:

- UDFs, variables, classes, etc. can be gathered in a separate file called modules.
- Users can write their own custom modules.

```
# Content of the module1.py
def sum(a, b):
    return a + b
def subtract(a, b):
    return a – b
def product(a, b):
    return a * b
def divide(a, b):
    return a / b
```

Python Functions (16/17)

Custom Modules:

- UDFs, variables, classes, etc. can be gathered in a separate file called modules.
- Users can write their own custom modules.

```
In[1] : import module1 as md1                    # Import module1.py as md1.
In[2] : md1.sum(3,4)
Out[2]: 7
In[3] : md1.product(4, 5)
Out[3]: 20
```

Python Functions (17/17)

Custom Modules:

- __name__ == "__main__" is True when the module is run directly instead of being imported.

```
# Content of the module1.py
def sum(a, b):
    return a + b
def subtract(a, b):
    return a – b
def product(a, b):
    return a * b
def divide(a, b):
    return a / b
if __name__ == "__main__":
    print(sum(3, 4))
    print(product(4, 5))
```

Coding Exercise #0106

Follow practice steps on 'ex_0106.ipynb'

Python Programming

Python Input and Output (1/7)

User input from keyboard:

```
In[1] : x = input('Enter anything : ')          # Input a string value.
Enter anything : Hello Python!
In[2] : print(x)                                # Display.
Hello Python!
```

```
In[1] : x = input('Enter a value : ')           # x is a string type variable.
Enter a value : 123
In[2] : print(5 + int(x))                       # int() converts a string into an integer.
125
```

Python Input and Output (2/7)

Open and close a file:

```
In[1] : f = open('new_file.txt', 'w')          # Create a file and then open it.
In[2] : f.close()                              # Close the file.
```

Open Mode	Explanation
r	Open a text file for reading.
w	Open a text file for writing. If the file does not exist, create it first.
a	Open an existing text file for appending at the end.
rb	Open a binary file for reading.
wb	Open a binary file for writing.

Python Input and Output (3/7)

Read a file:

```
In[1] : f = open('my_file.txt', 'r')          # Open a text file in read mode.
In[2] : all = f.read()                        # Read in all at once
In[3] : print(all)                            # Print out all at once.
Life
is
short,
You
need
Python!
In[4] : f.close()                             # Close the file.
```

Python Input and Output (4/7)

Read a file:

```
In[1] : f = open('my_file.txt', 'r')                    # Open a text file in read mode.
In[2] : all = f.read()                                  # Read in all at once.
In[3] : for line in all:
   ... :     print(line)                                # Print out line by line.
   ... :
Life
is
short,
You
need
Python!
In[4] : f.close()                                       # Close the file.
```

Python Input and Output (5/7)

Read a file:

```
In[1] : f = open('my_file.txt', 'r')                    # Open a text file in read mode.
In[2] : while True:
   ... :     line = f.readline()                        # Read in line by line.
   ... :     if not line: break                         # Repeat until the end is reached.
   ... :     print(line)
   ... :
Life
is
short,
You
need
Python!
In[3] : f.close()                                       # Close the file.
```

Python Input and Output (6/7)

Create a file and then write:

```
In[1] : f = open('new_file.txt', 'w')          # Create and open a new text file for writing.
In[2] : f.write('This is a new file.')         # Write a line of text.
In[3] : f.close()                              # Close the file.
```

```
In[4] : f = open('new_file.txt', 'w')          # Open an existing file for overwriting.
In[5] : f.write('This is another file!')       # Overwrite a new line of text.
In[6] : f.close()                              # Close the file.
```

Python Input and Output (7/7)

Append to a file:

```
In[1] : f = open('my_file.txt', 'a')                    # Open a file in append mode.
In[2] : f.write('This is the end!')                     # Append a line of text at the end.
In[3] : f.close()                                       # Close the file.
```

```
In[1] : with open('my_file.txt', 'a') as f:             # Use with structure.
   ... :        f.write('This is the added line #1 \n')     # Append line #1.
   ... :        f.write('This is the added line #2 \n')     # Append line #2.
   ... :        f.write('This is the added line #3 \n')     # Append line #3.
In[2] : f.close()                                       # f.close() is not required.
```