

PYTHON CƠ BẢN (Buổi 1)

AI Academy Vietnam

Content

- **Python Introduction**
- Python Flow Control
- Python Functions
- Python Datatypes
- Python Files
- Python Object & Class
- Python Date & Time

Python introduction

1. Environment setup
2. Keywords and Identifier
3. Statements & Comments
4. Variables
5. Datatypes
6. Type Conversion
7. I/O and Import
8. Operators
9. Namespace

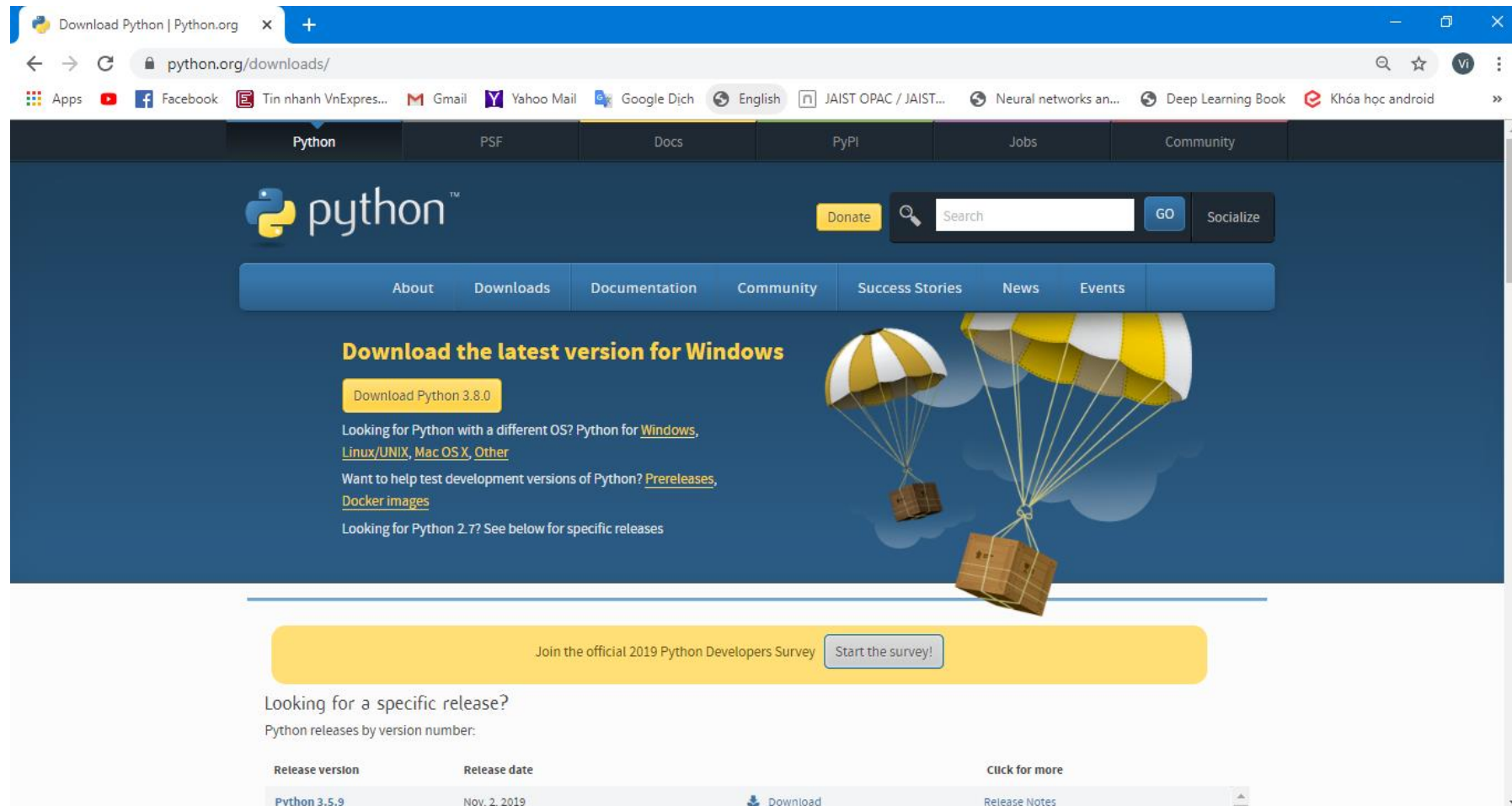
1. Environment setup



1. Install the Python 2.7.* or 3.*
2. Add the Python Directory to your System Path Environment Variable
3. Install pip/Anaconda to Manage Your Python Packages
4. Install virtualenv to Create Local Python Environments for Your Projects

1. Environment setup

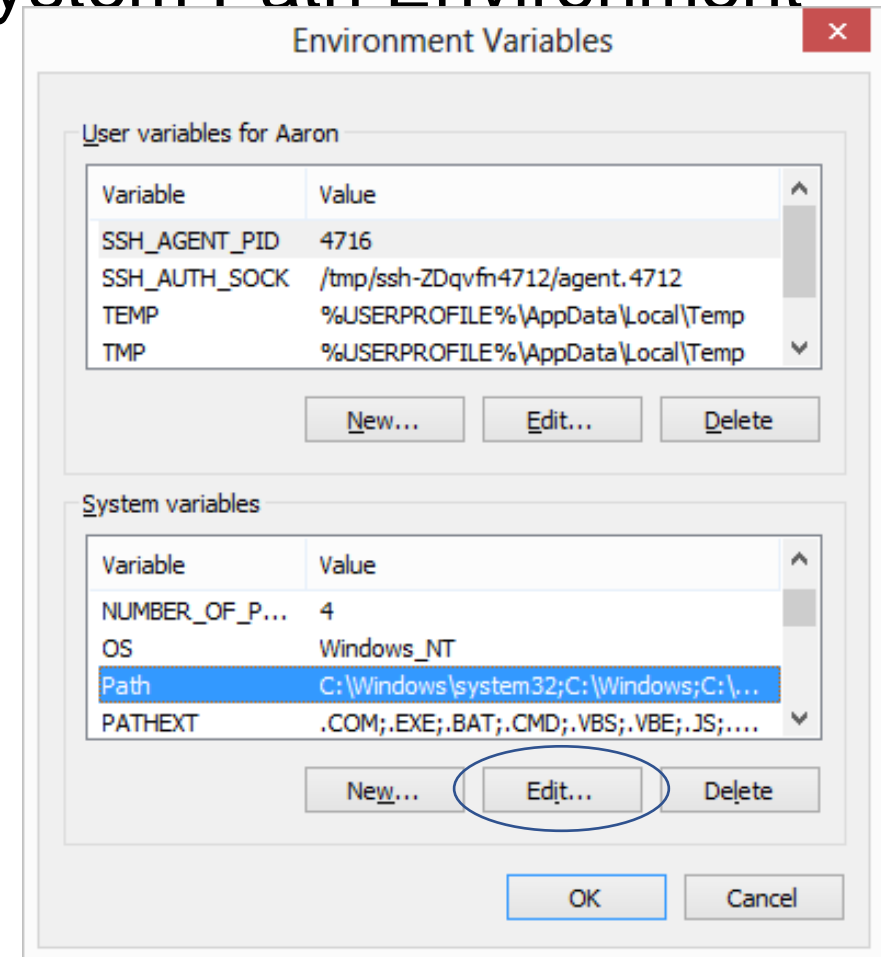
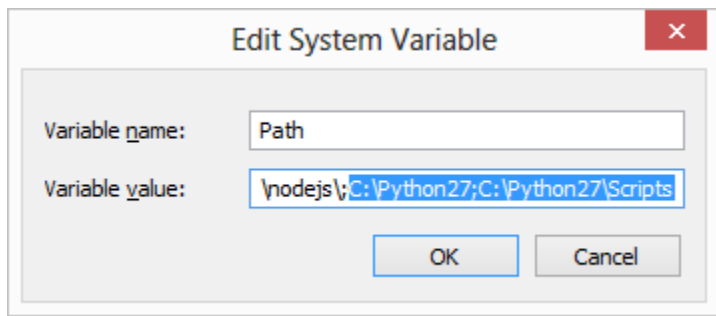
1. Install the Python 2.7.* or 3.* <https://www.python.org/downloads/>



1. Environment setup

2. Add the Python Directory to your System Path Environment Variable:

Control Panel → System Properties → Environment Variables and select the **PATH** variable

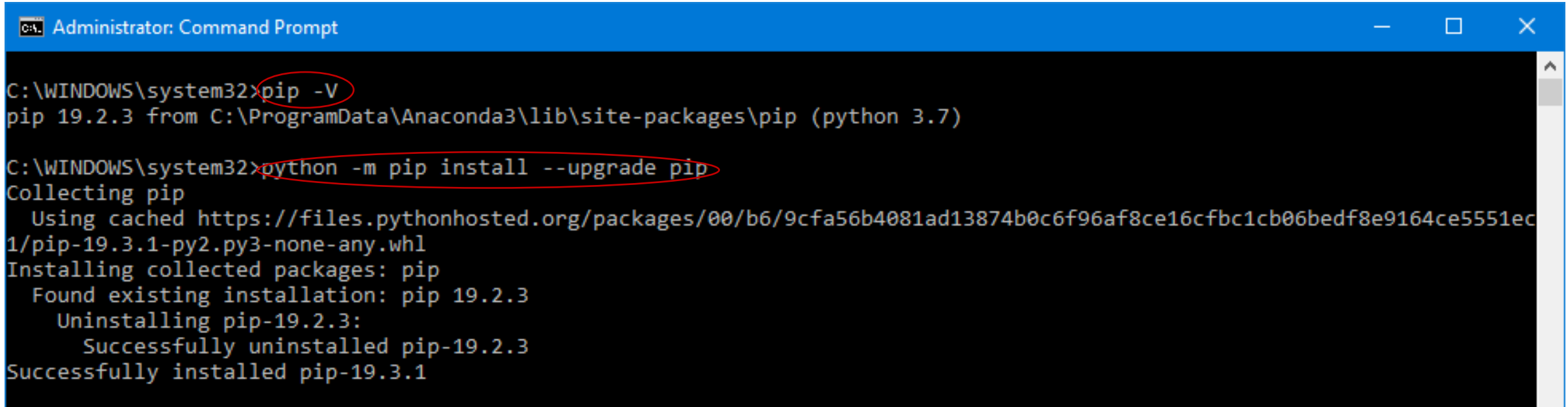


1. Environment setup

3. Install pip/Anaconda to Manage Your Python Packages

pip: <https://pypi.org/project/pip/>

anaconda: <https://www.anaconda.com/distribution/>



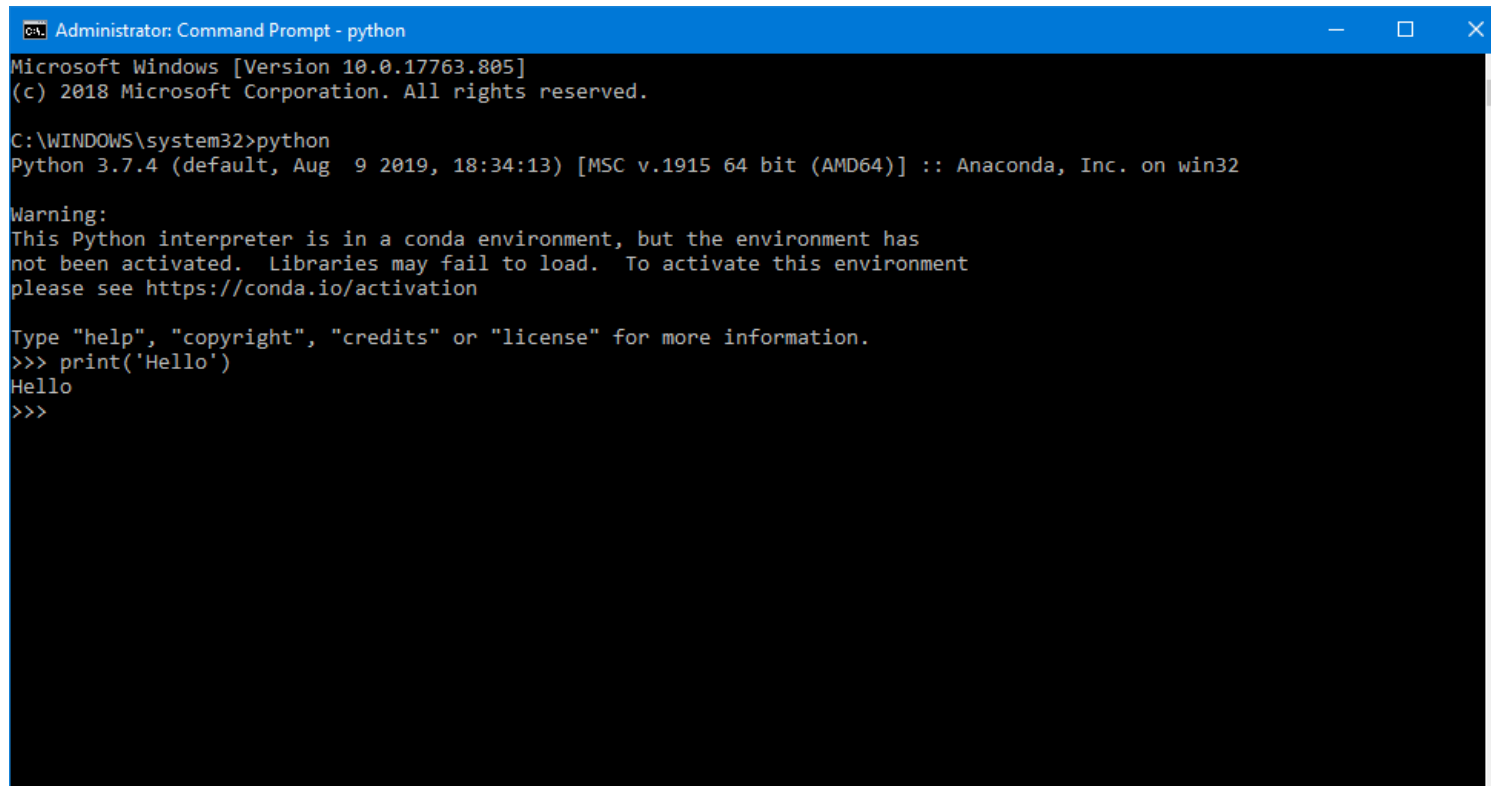
```
Administrator: Command Prompt

C:\WINDOWS\system32>pip -V
pip 19.2.3 from C:\ProgramData\Anaconda3\lib\site-packages\pip (python 3.7)

C:\WINDOWS\system32>python -m pip install --upgrade pip
Collecting pip
  Using cached https://files.pythonhosted.org/packages/00/b6/9cfa56b4081ad13874b0c6f96af8ce16cfbc1cb06bedf8e9164ce5551ec1/pip-19.3.1-py2.py3-none-any.whl
Installing collected packages: pip
  Found existing installation: pip 19.2.3
  Uninstalling pip-19.2.3:
    Successfully uninstalled pip-19.2.3
Successfully installed pip-19.3.1
```

1. Environment setup

- Python - shell (interpreter): Open the command prompt, write python and press enter.



```
Administrator: Command Prompt - python
Microsoft Windows [Version 10.0.17763.805]
(c) 2018 Microsoft Corporation. All rights reserved.

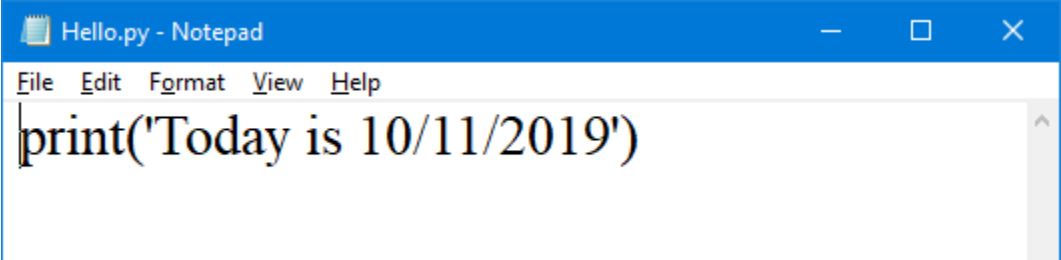
C:\WINDOWS\system32>python
Python 3.7.4 (default, Aug  9 2019, 18:34:13) [MSC v.1915 64 bit (AMD64)] :: Anaconda, Inc. on win32

Warning:
This Python interpreter is in a conda environment, but the environment has
not been activated. Libraries may fail to load. To activate this environment
please see https://conda.io/activation

Type "help", "copyright", "credits" or "license" for more information.
>>> print('Hello')
Hello
>>>
```

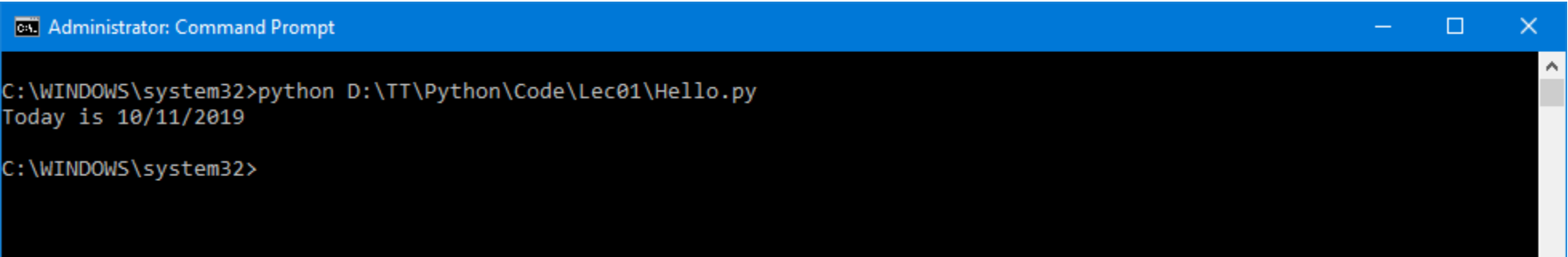

1. Environment setup

- Python - shell (interpreter):
 - Execute Python Script



```
File Edit Format View Help
print('Today is 10/11/2019')
```

- Create a Python file with extension .py



```
C:\WINDOWS\system32>python D:\TT\Python\Code\Lec01\Hello.py
Today is 10/11/2019
C:\WINDOWS\system32>
```

1. Environment setup

- Python - IDE
 - Python IDLE
 - **PyCharm**
 - **Jupyter**
 - Spider
 - Visual Studio Code
 - PyDev
 - Atom IDE
 - Wing Python
 - PyScripter

1. Environment setup

- Python - IDE
 - Python IDLE
 - **PyCharm**
 - **Jupyter**
 - Spider
 - Visual Studio Code
 - PyDev
 - Atom IDE
 - Wing Python
 - PyScripter

1. Environment setup

- Jupyter Notebook

- **Pros:**

- Produce rich and interactive output
 - Edit snippets before running them

- **Cons:**

- Complex for running long asynchronous tasks
 - The installation process is complex

- PyCharm

- **Pros:**

- It has a Version Control System Integration with many external plug-in supports.
 - Codes can be easily run, edit or debug without any external requirement

- **Cons:**

- Memory Intensive
 - Initial set-up can be time-consuming

2. Keywords and identifiers



VINBIGDATA



VINGROUP



AI Academy
Vietnam

- Keywords are the reserved words in Python.
- Cannot use a keyword as a variable name, function name or any other identifier
- The above keywords may get altered in different versions of Python.

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

2. Keywords and identifiers



VINBIGDATA



VINGROUP



AI Academy
Vietnam

- Python identifier is a name used to identify a variable, function, class, module or other object.
- An identifier starts with a letter A to Z or a to z or an underscore (_) followed by zero or more letters, underscores and digits (0 to 9).
- Python does not allow punctuation characters such as @, \$, and % within identifiers.
- Python is a case sensitive programming language.

2. Keywords and identifiers



VINBIGDATA



- Here are naming conventions for Python identifiers –
 - Class names start with an uppercase letter. All other identifiers start with a lowercase letter.
 - Starting an identifier with a single leading underscore indicates that the identifier is private.
 - Starting an identifier with two leading underscores indicates a strongly private identifier.
 - If the identifier also ends with two trailing underscores, the identifier is a language-defined special name.

3. Statements and comments

- Instructions that a Python interpreter can execute are called statements
- Statements in Python typically end with a new line.
- Multi-Line Statements
 - Python, however, allows the use of the line continuation character (\) to denote that the line should continue

```
if 5 > 2:  
    print("Five is greater than two!")  
if 5 > 2:  
    print("Five is greater than two!")
```

- The statements contained within the [], {}, or () brackets do not need to use the line continuation character.

Syntax Error:

```
if 5 > 2:  
    print("Five is greater than two!")  
    print("Five is greater than two!")
```


3. Statements and comments

- Multiple Statements on a Single Line

```
import sys; x = 'foo'; sys.stdout.write(x + '\n')
```

- Multiple Statement Groups as Suites

```
if expression :  
    suite  
elif expression :  
    suite  
else :  
    suite
```

3. Statements and comments

- Python Indentation

- Python uses indentation to indicate a block of code.

```
if 5 > 2:  
    print("Five is greater than two!")
```

- Python will give you an error if you skip the indentation:

Syntax Error:

```
if 5 > 2:  
print("Five is greater than two!")
```

3. Statements and comments

- Python Indentation

- The number of spaces is up to you as a programmer, but it has to be at least one.

```
if 5 > 2:
    print("Five is greater than two!")
if 5 > 2:
    print("Five is greater than two!")
```

- You have to use the same number of spaces in the same block of code.

Syntax Error:

```
if 5 > 2:
    print("Five is greater than two!")
    print("Five is greater than two!")
```

3. Statements and comments

- Comments start with a #, and Python will render the rest of the line as a comment:

```
#!/usr/bin/python  
  
# First comment  
print "Hello, Python!" # second comment
```

- Following triple-quoted string is also ignored by Python interpreter and can be used as a multiline comments:

```
...  
This is a multiline  
comment.  
...
```

3. Statements and comments

- Quotation
 - Python accepts single ('), double (") and triple (''' or """) quotes to denote string literals, as long as the same type of quote starts and ends the string
 - The triple quotes are used to span the string across multiple lines

```
word = 'word'  
sentence = "This is a sentence."  
paragraph = """This is a paragraph. It is  
made up of multiple lines and sentences."""
```

4. Variables

- Creating Variables

- Variables are containers for storing data values.
- Python has no command for declaring a variable.
- A variable is created the moment you first assign a value to it.

```
x = 5  
y = "John"
```

- Variables do not need to be declared with any particular type and can even change type

```
x = 4 # x is of type int  
x = "Sally" # x is now of type str
```

4. Variables

- Assign Value to Multiple Variables

- Python allows you to assign values to multiple variables in one line:

```
x, y, z = "Orange", "Banana", "Cherry"
```

- can assign the *same* value to multiple variables in one line

```
x = y = z = "Orange"
```

4. Variables

- Output Variables

- Python print statement is often used to output variables
- Combine both text and a variable, Python uses the + character

```
x = "awesome"  
print("Python is " + x)
```

- You can also use the + character to add a variable to another variable:

```
x = "Python is "  
y = "awesome"  
z = x + y  
print(z)
```

- For numbers, the + character works as a mathematical operator.
- Python will give you an error when trying to combine a string and a number

5. Data Types

- Built-in Data Types

Text Type: `str`

Numeric Types: `int`, `float`, `complex`

Sequence Types: `list`, `tuple`, `range`

Mapping Type: `dict`

Set Types: `set`, `frozenset`

Boolean Type: `bool`

Binary Types: `bytes`, `bytearray`, `memoryview`

5. Data Types

- Setting the Data Type: the data type is set when you assign a value to a variable:

<code>x = "Hello World"</code>	<code>str</code>
<code>x = 20</code>	<code>int</code>
<code>x = 20.5</code>	<code>float</code>
<code>x = 1j</code>	<code>complex</code>
<code>x = ["apple", "banana", "cherry"]</code>	<code>list</code>
<code>x = ("apple", "banana", "cherry")</code>	<code>tuple</code>
<code>x = range(6)</code>	<code>range</code>
<code>x = {"name" : "John", "age" : 36}</code>	<code>dict</code>
<code>x = {"apple", "banana", "cherry"}</code>	<code>set</code>
<code>x = frozenset({"apple", "banana", "cherry"})</code>	<code>frozenset</code>
<code>x = True</code>	<code>bool</code>
<code>x = b"Hello"</code>	<code>bytes</code>
<code>x = bytearray(5)</code>	<code>bytearray</code>
<code>x = memoryview(bytes(5))</code>	<code>memoryview</code>

5. Data Types

- Setting the Specific Data Type

<code>x = str("Hello World")</code>	<code>str</code>
<code>x = int(20)</code>	<code>int</code>
<code>x = float(20.5)</code>	<code>float</code>
<code>x = complex(1j)</code>	<code>complex</code>
<code>x = list(("apple", "banana", "cherry"))</code>	<code>list</code>
<code>x = tuple(("apple", "banana", "cherry"))</code>	<code>tuple</code>
<code>x = range(6)</code>	<code>range</code>
<code>x = dict(name="John", age=36)</code>	<code>dict</code>
<code>x = set(("apple", "banana", "cherry"))</code>	<code>set</code>
<code>x = frozenset(("apple", "banana", "cherry"))</code>	<code>frozenset</code>
<code>x = bool(5)</code>	<code>bool</code>
<code>x = bytes(5)</code>	<code>bytes</code>
<code>x = bytearray(5)</code>	<code>bytearray</code>
<code>x = memoryview(bytes(5))</code>	<code>memoryview</code>

6. Type conversion

- The process of converting the value of one data type (integer, string, float, etc.) to another data type is called type conversion.
- Python has two types of type conversion.
 1. Implicit Type Conversion (automatically converts the lower data type (integer) to the higher data type (float) to avoid data loss)
 2. Explicit Type Conversion (users convert the data type of an object to required data type):

`<required_datatype>(expression)`

7. I/O and Import

- use the `print()` function to output data to the standard output device (screen)

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

- Here, `objects` is the value(s) to be printed.
- The `sep` separator is used between the values. It defaults into a space character.
- After all values are printed, `end` is printed. It defaults into a new line.
- The `file` is the object where the values are printed and its default value is `sys.stdout` (screen) .

7. I/O and Import

- Example:

```
print('This sentence is output to the screen')
```

```
a = 5
```

```
print('The value of a is', a)
```

```
print(1, 2, 3, 4)
```

```
print(1, 2, 3, 4, sep='*')
```

```
print(1, 2, 3, 4, sep='#', end='&')
```

7. I/O and Import

- Output formatting
 - Sometimes we would like to format our output to make it look attractive.
 - This can be done by using the `str.format()` method.
 - This method is visible to any string object.
 - The curly braces `{}` are used as placeholders

```
>>> x = 5; y = 10
```

```
>>> print('The value of x is {} and y is {}'.format(x,y))
```

```
The value of x is 5 and y is 10
```

7. I/O and Import

- Output formatting
 - Can specify the order in which they are printed by using numbers (tuple index).

```
print('I love {0} and {1}'.format('bread','butter'))  
print('I love {1} and {0}'.format('bread','butter'))
```

- can even use keyword arguments to format the string.

```
>>> print('Hello {name}, {greeting}'.format(greeting = 'Goodmorning', name = 'John'))  
Hello John, Goodmorning
```


7. I/O and Import

- Python allows for user input.
- Uses the `input()` method.

```
>>> num = input('Enter a number: ')
Enter a number: 10
>>> num '10'
```

- The entered value 10 is a string, not a number. To convert this into a number we can use `int()` or `float()` functions.

7. I/O and Import

- When our program grows bigger, it is a good idea to break it into different modules.
- A module is a file containing Python definitions and statements. Python modules have a filename and end with the extension .py.
- Definitions inside a module can be imported to another module or the interactive interpreter in Python. We use the import keyword to do this

7. I/O and Import

- Examples:

```
import math
```

```
import math  
print(math.pi)
```

```
>>> from math import pi  
>>> pi 3.141592653589793
```

8. Operators

- Python Arithmetic Operators

Operator	Name	Example
+	Addition	<code>x + y</code>
-	Subtraction	<code>x - y</code>
*	Multiplication	<code>x * y</code>
/	Division	<code>x / y</code>
%	Modulus	<code>x % y</code>
**	Exponentiation	<code>x ** y</code>
//	Floor division	<code>x // y</code>

8. Operators

- Python Assignment Operators

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3
&=	x &= 3	x = x & 3
=	x = 3	x = x 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

8. Operators

- Python Comparison Operators

Operator	Name	Example
<code>==</code>	Equal	<code>x == y</code>
<code>!=</code>	Not equal	<code>x != y</code>
<code>></code>	Greater than	<code>x > y</code>
<code><</code>	Less than	<code>x < y</code>
<code>>=</code>	Greater than or equal to	<code>x >= y</code>
<code><=</code>	Less than or equal to	<code>x <= y</code>

8. Operators

- Python Logical Operators

Operator	Description	Example
and	Returns True if both statements are true	<code>x < 5 and x < 10</code>
or	Returns True if one of the statements is true	<code>x < 5 or x < 4</code>
not	Reverse the result, returns False if the result is true	<code>not(x < 5 and x < 10)</code>

- Python Identity Operators

Operator	Description	Example
is	Returns true if both variables are the same object	<code>x is y</code>
is not	Returns true if both variables are not the same object	<code>x is not y</code>

8. Operators

- Python Membership Operators

Operator	Description	Example
<code>in</code>	Returns True if a sequence with the specified value is present in the object	<code>x in y</code>
<code>not in</code>	Returns True if a sequence with the specified value is not present in the object	<code>x not in y</code>

- Python Bitwise Operators

Operator	Name	Description
<code>&</code>	AND	Sets each bit to 1 if both bits are 1
<code> </code>	OR	Sets each bit to 1 if one of two bits is 1
<code>^</code>	XOR	Sets each bit to 1 if only one of two bits is 1
<code>~</code>	NOT	Inverts all the bits
<code><<</code>	Zero fill left shift	Shift left by pushing zeros in from the right and let the leftmost bits fall off
<code>>></code>	Signed right shift	Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off

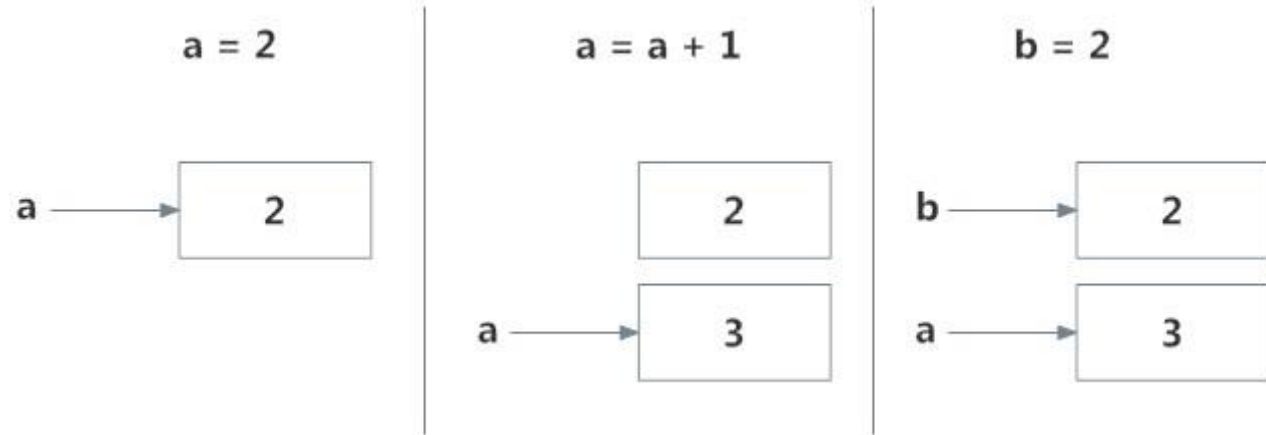
9. Namespace

- Name (also called identifier) is simply a name given to objects. Everything in Python is an object. Name is a way to access the underlying object.

```
a = 2
print('id(2) =', id(2))
print('id(a) =', id(a))
```

9. Namespace

```
a = 2  
  
print('id(a) =', id(a))  
  
a = a + 1  
  
print('id(a) =', id(a))  
print('id(3) =', id(3))  
  
b = 2  
  
print('id(b) =', id(b))  
print('id(2) =', id(2))
```



9. Namespace

- A namespace: a collection of names.
- A namespace: a mapping of every name you have defined to corresponding objects.
- Different namespaces can co-exist at a given time but are completely isolated.

9. Namespace

- A namespace containing all the built-in names is created when we start the Python interpreter and exists as long as the interpreter runs.
- Each module creates its own global namespace.
- A local namespace is created when a function is called, which has all the names defined in it



9. Namespace

- Python Variable Scope

- A scope is the portion of a program from where a namespace can be accessed directly without any prefix.
- At any given moment, there are at least three nested scopes.
 - Scope of the current function which has local names
 - Scope of the module which has global names
 - Outermost scope which has built-in names
- When a reference is made inside a function, the name is searched in the local namespace, then in the global namespace and finally in the built-in namespace.
- If there is a function inside another function, a new scope is nested inside the local scope.

9. Namespace

- Python Variable Scope

```
def outer_function():  
    b = 20  
    def inner_func():  
        c = 30  
a = 10
```

9. Namespace

- Python Variable Scope

```
def outer_function():  
    a = 20  
    def inner_function():  
        a = 30  
        print('a =', a)  
    inner_function()  
    print('a =', a)  
  
a = 10  
outer_function()  
print('a =', a)
```

9. Namespace

- Python Variable Scope

```
def outer_function():  
    global a  
    a = 20  
    def inner_function():  
        global a  
        a = 30  
        print('a =', a)  
    inner_function()  
    print('a =', a)  
  
a = 10  
outer_function()  
  
print('a =', a)
```