ALBUKHARY INTERNATIONAL UNIVERSITY

Artificial Intelligence

LAB One

Introduction to Python

What is Python?

Python is a popular programming language. It was created by Guido van Rossum, and released in 1991.

Python is a high-level, general-purpose, interpreted programming language.

1) High-level Python is a high-level programming language that makes it easy to learn. Python doesn't require you to understand the details of the computer in order to develop programs efficiently.

2) General-purpose Python is a general-purpose language. It means that you can use Python in various domains including:

- Web applications
- Big data applications
- Testing
- Automation
- Data science, machine learning, and AI
- Desktop software
- Mobile apps

3) Interpreted Python is an interpreted language. To develop a Python program, you write Python code into a file called source code.

To execute the source code, you need to convert it to the machine language that the computer can understand. And the Python interpreter turns the source code, line by line, once at a time, into the machine code when the Python program executes.

Compiled languages like Java and C# use a compiler that compiles the whole source code before the program executes.

Why Python

- Python increases your productivity. Python allows you to solve complex problems in less time and fewer lines of code. It's quick to make a prototype in Python.
- Python becomes a solution in many areas across industries, from web applications to data science and machine learning.
- Python is quite easy to learn in comparison with other programming languages. Python syntax is clear and beautiful.
- Python has a simple syntax similar to the English language.
- Python has a large ecosystem that includes lots of libraries and frameworks.
- Python is cross-platform. Python programs can run on Windows, Linux, and macOS.
- Python has a huge community. Whenever you get stuck, you can get help from an active community.
- Python developers are in high demand.

Good to know

The most recent major version of Python is Python 3, which we shall be using in this tutorial. However, Python 2, although not being updated with anything other than security updates, is still quite popular.

Python Syntax compared to other programming languages

- Python was designed for readability, and has some similarities to the English language with influence from mathematics.
- Python uses new lines to complete a command, as opposed to other programming languages which often use semicolons or parentheses.
- Python relies on indentation, using whitespace, to define scope; such as the scope of loops, functions and classes. Other programming languages often use curly-brackets for this purpose.

PYTHON INSTALLATION

ANACONDA INSTALLATION

In [1]:
```
#Example
print("Hello, World!")
```

```
Hello, World!
```

In [2]:
```
print('Hello AI Class')
```

```
Hello AI Class
```

1. Comments

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

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

Example

Comments in Python:

In [3]:
```
#This is a comment.
print("Hello, World!")
```

```
Hello, World!
```

1.1 Python Comments

Comments can be used to explain Python code.

Comments can be used to make the code more readable.

Comments can be used to prevent execution when testing code.

1.1.1 Creating a Comment

Comments starts with a #, and Python will ignore them:

Example

In [4]:
```
#This is a comment
print("Hello, World!")
```

```
Hello, World!
```

Comments can be placed at the end of a line, and Python will ignore the rest of the line:

Example

In [5]:
```
print("Hello, World!") #This is a comment
```

```
Hello, World!
```

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

Example

In [6]:
```
#print("Hello, World!")
print("Welcome, AI Class!")
```

```
Welcome, AI Class!
```

1.1.2 Multi Line Comments

Python does not really have a syntax for multi line comments.

To add a multiline comment you could insert a # for each line:

Example

In [7]:
```
#This is a comment
#written in
#more than just one line
print("Hello, World!")
```

```
Hello, World!
```

Or, not quite as intended, you can use a multiline string.

Since Python will ignore string literals that are not assigned to a variable, you can add a multiline string (triple quotes) in your code, and place your comment inside it:

Example

In [8]:
```
"""
This is a comment
written in
more than just one line
"""
print("Hello, World!")
```

```
Hello, World!
```

As long as the string is not assigned to a variable, Python will read the code, but then ignore it, and you have made a multiline comment.

2. Variables

Variables are containers for storing data values.

2.1 Creating Variables

Python has no command for declaring a variable.

A variable is created the moment you first assign a value to it.

Example

In [9]:
```
x = 5
y = "Artificial Intelligence"
print(x)
print(y)
```

```
5
Artificial Intelligence
```

In [10]:
```
x = 'AI'
print(x)
```

```
AI
```

Variables do not need to be declared with any particular type, and can even change type after they have been set.

Example

In [11]:
```
x = 5       # x is of type int
x = "AI" # x is now of type str
print(x)
```

```
AI
```

2.1.1 Casting

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

Example

In [12]:
```
x = str("4")    # x will be '4'
y = int(4)    # y will be 4
z = float(4)  # z will be 4.0
```

2.1.2 Get the Type

You can get the data type of a variable with the type() function.

Example

In [14]:
```
x = 5
y = "AI"
print(type(x))
print(type(y))
```

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

You will learn more about data types and casting later in this tutorial.

2.1.3 Single or Double Quotes?

String variables can be declared either by using single or double quotes:

Example

In [15]:
```
x = "AI"
# is the same as
x = 'AI'
```

2.1.4 Case-Sensitive

Variable names are case-sensitive.

Example

This will create two variables:

In [16]:
```
a = 4
A = "AI"
#A will not overwrite a
```

2.2 Variable Names

A variable can have a short name (like x and y) or a more descriptive name (age, carname, total_volume). Rules for Python variables:

- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- Variable names are case-sensitive (age, Age and AGE are three different variables)

Example

Legal variable names:

In [17]:
```
myvar = "AI"
my_var = "AI"
_my_var = "AI"
myVar = "AI"
MYVAR = "AI"
myvar2 = "AI"
```

In [ ]:
```
Example
Illegal variable names:

2myvar = "AI"
my-var = "AI"
my var = "AI"
```

Remember that variable names are case-sensitive

2.3 Multi Words Variable Names

Variable names with more than one word can be difficult to read.

There are several techniques you can use to make them more readable:

2.3.1 Camel Case

Each word, except the first, starts with a capital letter:

In [19]:
```
myVariableName = "AI"
```

2.3.2 Pascal Case

Each word starts with a capital letter:

In [20]:
```
MyVariableName = "AI"
```

2.3.3 Snake Case

Each word is separated by an underscore character:

In [21]:
```
my_variable_name = "AI"
```

2.4 Many Values to Multiple Variables

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

Example

In [22]:
```
x, y, z = "Orange", "Banana", "Cherry"
print(x)
print(y)
print(z)
```

```
Orange
Banana
Cherry
```

Note: Make sure the number of variables matches the number of values, or else you will get an error.

2.4.1 One Value to Multiple Variables

And you can assign the same value to muliple variables in one line:

Example

In [23]:
```
x = y = z = "Orange"
print(x)
print(y)
print(z)
```

```
Orange
Orange
Orange
```

2.4.2 Output Variables

The Python print statement is often used to output variables.

To combine both text and a variable, Python uses the + character:

Example

In [26]:
```
x = "awesome"
print("Python is " + x)
```

```
Python is awesome
```

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

Example

In [27]:
```
x = "Python is "
y = "awesome"
z =  x + y
print(z)
```

```
Python is awesome
```

For numbers, the + character works as a mathematical operator:

Example

In [28]:
```
x = 5
y = 10
print(x + y)
```

```
15
```

If you try to combine a string and a number, Python will give you an error:

In [ ]:
```
Example
x = 5
y = "John"
print(x + y)
```

3. Python Data Types

3.1 Built-in Data Types

In programming, data type is an important concept.

Variables can store data of different types, and different types can do different things.

Python has the following data types built-in by default, in these categories:

- 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

Getting the Data Type

You can get the data type of any object by using the type() function:

Example Print the data type of the variable x:

In [30]:
```
x = 5
print(type(x))
```

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

3.1.1 Python Numbers

There are three numeric types in Python:

- int
- float
- complex

Variables of numeric types are created when you assign a value to them:

Example

In [31]:
```
x = 1    # int
y = 2.8  # float
z = 1j   # complex
```

To verify the type of any object in Python, use the type() function:

Example

In [32]:
```
print(type(x))
print(type(y))
print(type(z))
```

```
<class 'int'>
<class 'float'>
<class 'complex'>
```

3.1.1.1 Int

Int, or integer, is a whole number, positive or negative, without decimals, of unlimited length.

Example

Integers:

In [33]:
```
x = 1
y = 35656222554887711
z = -3255522

print(type(x))
print(type(y))
print(type(z))
```

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

3.1.1.2 Float

Float, or "floating point number" is a number, positive or negative, containing one or more decimals.

Example Floats:

In [34]:
```
x = 1.10
y = 1.0
z = -35.59

print(type(x))
print(type(y))
print(type(z))
```

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

Float can also be scientific numbers with an "e" to indicate the power of 10.

Example Floats:

In [35]:
```
x = 35e3
y = 12E4
z = -87.7e100

print(type(x))
print(type(y))
print(type(z))
```

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

3.1.1.3 Complex

Complex numbers are written with a "j" as the imaginary part:

Example Complex:

In [36]:
```
x = 3+5j
y = 5j
z = -5j

print(type(x))
print(type(y))
print(type(z))
```

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

4. Type Conversion

You can convert from one type to another with the int(), float(), and complex() methods:

Example

Convert from one type to another:

In [37]:
```
x = 1    # int
y = 2.8  # float
z = 1j   # complex

#convert from int to float:
a = float(x)

#convert from float to int:
b = int(y)

#convert from int to complex:
c = complex(x)

print(a)
print(b)
print(c)

print(type(a))
print(type(b))
print(type(c))
```

```
1.0
2
(1+0j)
<class 'float'>
<class 'int'>
<class 'complex'>
```

Note: You cannot convert complex numbers into another number type.

Python Casting

5. Specify a Variable Type

There may be times when you want to specify a type on to a variable. This can be done with casting. Python is an object-orientated language, and as such it uses classes to define data types, including its primitive types.

Casting in python is therefore done using constructor functions:

- int() - constructs an integer number from an integer literal, a float literal (by removing all decimals), or a string literal (providing the string represents a whole number)
- float() - constructs a float number from an integer literal, a float literal or a string literal (providing the string represents a float or an integer)
- str() - constructs a string from a wide variety of data types, including strings, integer literals and float literals

Example

Integers:

In [38]:
```
x = int(1)   # x will be 1
y = int(2.8) # y will be 2
z = int("3") # z will be 3
```

Example

Floats:

In [39]:
```
x = float(1)     # x will be 1.0
y = float(2.8)   # y will be 2.8
z = float("3")   # z will be 3.0
w = float("4.2") # w will be 4.2
```

Example

Strings:

In [40]:
```
x = str("s1") # x will be 's1'
y = str(2)    # y will be '2'
z = str(3.0)  # z will be '3.0'
```

In [ ]:
```
```