Data Science & Machine Learning with Python
Class 1: Python

Environment Setup, Variables, Data Types & Type Casting

- Python Environment Setup
- Basic Syntax of Python
  - Statements
  - Indentation
  - Comments
- Python Variables
- Data Types in Python
- Type Casting in Python

<u>Environment Setup</u>

1. Download and Install PyCharm

- Go to the JetBrains website and download PyCharm. Choose the appropriate version for your operating system (Windows, macOS, or Linux). You can opt for the free Community edition or the paid Professional edition, which offers more features.
- Run the installer and follow the on-screen instructions to complete the installation.

2. Configure PyCharm for Python Development

- **Start PyCharm:** Launch PyCharm after installation.
- **Create a New Project:**
  1. Click on "New Project" on the welcome screen.
  2. Choose a location for your project and set the project name.
  3. **Virtual Environment:**
     - By default, PyCharm will create a virtual environment. You can specify the Python interpreter by clicking on "New environment using" and selecting a Python version installed on your system. This helps isolate your project dependencies.
     - Alternatively, you can select an existing interpreter from the "Existing interpreter" option.
  4. Click "Create" to set up the project.

3. Set Up Python Interpreter

- PyCharm usually sets the interpreter automatically based on your project settings. However, if you need to set it manually:
    1. Go to `File > Settings` (or `PyCharm > Preferences` on macOS).
    2. Navigate to `Project: <Your Project Name> > Python Interpreter`.
    3. Click the gear icon and select "Add".
    4. Choose the interpreter type (Virtualenv, Conda, System Interpreter) and configure it as needed.
    5. Click "OK" to save the settings.

Basic Syntax of Python

Python is known for its readability and simplicity. Its syntax emphasizes readability, using indentation to define code blocks instead of braces or keywords. Here are the basics:

1. Statements

In Python, a statement is a logical instruction that the interpreter can execute. Python statements are usually written one per line, and they do not require a semicolon (`;`) at the end. However, multiple statements can be written on a single line using semicolons to separate them.

Examples:

```
# Single statement

x = 5



# Multiple statements in one line

x = 5; y = 10; z = x + y
```

2. Indentation

Indentation is a key aspect of Python's syntax. It defines the structure of the code, indicating code blocks such as those for loops, conditional statements, functions, and classes. Unlike many other programming languages, Python uses whitespace (usually four spaces) for indentation instead of braces `{}`.

Example:

```
if x > 0:

   print("x is positive")

   y = x + 2

else:

   print("x is not positive")

   y = 0
```

In the example above, the lines inside the `if` and `else` blocks are indented, indicating that they belong to those respective blocks.

3. Comments

Comments are used to explain the code and make it more readable. They are not executed by the interpreter. In Python, comments start with the `#` symbol for single-line comments. For multi-line comments, triple quotes (`'''` or `"""`) are used.

Examples:

```
# This is a single-line comment



"""

This is a multi-line comment.

It can span multiple lines.

"""



'''Another multi-line comment example.'''
```

Variables

Variables are containers for storing data values.

Creating Variables

Python has no command for declaring a variable.

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

Examples:

```
x = 5
y = "sayem"
print(x)
print(y)
```

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

Examples:

```
x = 4       # x is of type int

x = "Sayem" # x is now of type str

print(x)
```

Casting

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

```
x = str(3)    # x will be '3'

y = int(3)    # y will be 3

z = float(3)  # z will be 3.0
```

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)
- A variable name cannot be any of the Python keywords.

Example

Legal variable names:

```
person_age = 30

city = "Dhaka"

_country = "Bangladesh"

personOccupation = "Data Analyst"

COUNTRY_CODE = 880

favorite_number = 7
```

Example

Illegal variable names:

```
2myvar = "cow"

my-var = "dog"

my var = "jar"
```

Many Values to Multiple Variables

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

Example

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

print(x)

print(y)

print(z)
```

One Value to Multiple Variables

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

Example

```
x = y = z = "Orange"
print(x)
print(y)
print(z)
```

Output Variables

The Python `print()` function is often used to output variables.

Example

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

You can also use the + operator to output multiple variables:

Example

```
language = "JavaScript "
verb = "is "
adjective = "fantastic"
print(language + verb + adjective)
```

Global Variables

Variables that are created outside of a function (as in all of the examples in the previous pages) are known as global variables.

Global variables can be used by everyone, both inside of functions and outside.

Example

```
x = "awesome"

def myfunc():
 print("Python is " + x)

myfunc()
```

If you create a variable with the same name inside a function, this variable will be local, and can only be used inside the function. The global variable with the same name will remain as it was, global and with the original value.

Example

```
x = "awesome"

def myfunc():
 x = "fantastic"
 print("Python is " + x)

myfunc()

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

Python Data Types

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`
None Type:	`NoneType`

Example

Print the data type of the variable x:

```
x = 5
print(type(x))
```

Python Casting

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

```
x = int(1)   # x will be 1
y = int(2.8) # y will be 2
z = int("3") # z will be 3
```

Example

```
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
```