1. Built-in Data Types

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

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

Format	Data Type
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

In Python, you can get the data type of any object by using the type() function.
Example:

```
x = 1
print(type(x))
```

In Python, the data type of the variable is being set when you assign a value to it:

Below is the table shows how you can assign different data type values to any variables:

Example	Data Type
`x = "Hello World"`	str
`x = 20`	int
`x = 20.5`	float

`x = 1j`	complex
`x = ["Byte", "Up", "Academy"]`	list
`x = ("Byte", "Up", "Academy")`	tuple
`x = range(6)`	range
`x = {"name" : "John", "age" : 12}`	dict
`x = {"Byte", "Up", "Academy"}`	set
`x = frozenset({"Byte", "Up", "Academy"})`	frozenset
`x = True`	bool
`x = b"Hello"`	bytes
`x = bytearray(5)`	bytearray
`x = memoryview(bytes(5))`	memoryview
`x = None`	NoneType

Also, if you want to specify the data type, you c an do it like below:

```
x = str("Hello World")
x = int(20)
x = float(20.5)
```

We will be using most of these data types along the way of our upcoming sessions.

2. Python Casting

There are times when you want to change a type of a value. 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.

- **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)

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

- **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)

```
x = float(1)     # x will be 1.0
y = float(2.8)   # y will be 2.8
w = float("4.2") # w will be 4.2
```

- **str()** - constructs a string from a wide variety of data types, including strings, integer literals and float literals

```
x = str("s1") # x will be 's1'
y = str(2)    # y will be '2'
z = str(3.0)  # z will be '3.0'
```

3. string - Quotes inside Quotes

You can use quotes inside a string, as long as they don't match the quotes surrounding the string:

```
print("It's alright")
print("It’s called 'ByteUp Academy")
print('It is called "ByteUp Academy"')
```

4. Multiline Strings

You can assign a multiline string to a variable by using three quotes:

```
a = """I am learning the,
Python language with
NyteUp Academy."""
print(a)
```

5. String Slicing

You can return a range of characters by using the slice syntax. Specify the start index and the end index, separated by a colon, to return a part of the string.

```
x = "Hello World!"
print(x[2:5])
```

- Please note that the first character has the index of 0.

6. String Modification/Manipulation

Python has a set of built-in methods that you can use on strings.

- print(a.lower()) → returns the string in lower case
- print(a.upper()) → returns the string in upper case
- print(a.strip()) → removes any whitespace from beginning or end
- print(a.replace("H", "J")) → replaces a string with another string
- print(a.split(",")) → splits the string into substrings if it finds instances of the separator

There are so many string functions available to use in Python. They are below:

Method	Description
capitalize()	Converts the first character to upper case
casefold()	Converts string into lower case
center()	Returns a centered string
count()	Returns the number of times a specified value occurs in a string
encode()	Returns an encoded version of the string
endswith()	Returns true if the string ends with the specified value
expandtabs()	Sets the tab size of the string
find()	Searches the string for a specified value and returns the position of where it was found
format()	Formats specified values in a string
format_map()	Formats specified values from a dictionary in a string
index()	Searches the string for a specified value and returns the position of where it was found
isalnum()	Returns True if all characters in the string are alphanumeric
isalpha()	Returns True if all characters in the string are in the alphabet
isascii()	Returns True if all characters in the string are ascii characters
isdecimal()	Returns True if all characters in the string are decimals
isdigit()	Returns True if all characters in the string are digits
isidentifier()	Returns True if the string is an identifier
islower()	Returns True if all characters in the string are lower case
isnumeric()	Returns True if all characters in the string are numeric
isprintable()	Returns True if all characters in the string are printable
isspace()	Returns True if all characters in the string are whitespaces
istitle()	Returns True if the string follows the rules of a title

isupper()	Returns True if all characters in the string are upper case
join()	Converts the elements of an iterable into a string
ljust()	Returns a left justified version of the string
lower()	Converts a string into lower case
lstrip()	Returns a left trim version of the string
maketrans()	Returns a translation table to be used in translations
partition()	Returns a tuple where the string is parted into three parts
replace()	Returns a string where a specified value is replaced with a specified value
rfind()	Searches the string for a specified value and returns the last position of where it was found
rindex()	Searches the string for a specified value and returns the last position of where it was found
rjust()	Returns a right justified version of the string
rpartition()	Returns a tuple where the string is parted into three parts
rsplit()	Splits the string at the specified separator, and returns a list
rstrip()	Returns a right trim version of the string
split()	Splits the string at the specified separator, and returns a list
splitlines()	Splits the string at line breaks and returns a list
startswith()	Returns true if the string starts with the specified value
strip()	Returns a trimmed version of the string
swapcase()	Swaps cases, lower case becomes upper case and vice versa
title()	Converts the first character of each word to upper case
translate()	Returns a translated string
upper()	Converts a string into upper case
zfill()	Fills the string with a specified number of 0 values at the beginning

7. Escape Character

To insert characters that are illegal in a string, use an escape character. An escape character is a backslash \ followed by the character you want to insert. An example of an illegal character is a double quote inside a string that is surrounded by double quotes: Example

```
a = "We are in the “middle" of nowhere”
```

- You will get an error if you use double quotes inside a string that is surrounded by double quotes:

To fix this problem, use the escape character \" like below:

```
a = "We are in the \“middle\" of nowhere”
```

Below are the various escape characters offered in Python:

Code	Result
\'	Single Quote
****	Backslash
\n	New Line
\r	Carriage Return
\t	Tab
\b	Backspace
\f	Form Feed
\ooo	Octal value
\xhh	Hex value