1.2 Creating variables and assigning values

Python is a Dynamically typed language. It means based on the value we assign to a variable, it sets the datatype to it.

Now the question is "How do we assign a value to a variable?🤔". It's pretty easy.

```
<variable_name> = <value>
```

We have a big list of data types that come as builtins in Python.

- None
- bytes
- int
- bool
- float
- complex
- string
- tuple
- list
- set
- dict

Apart from the above prominent data types, we have a few other data types like namedtuple, frozensets, etc..

Let's create examples for the above data types, will be little bored in just seeing the examples. We would be covering in depth about these data types in upcoming chapters :)

Few things to know before getting into the examples:😉

1. `print` function is used to print the data on to the console. We used `f` inside the print function which is used to format the strings as `{}`, these are known as f-strings.
2. `type` function is used to find the type of the object or datatype.

In [1]:
```
# None
none_datatype = None
print(f"The type of none_datatype is {type(none_datatype)}")
```

```
The type of none_datatype is <class 'NoneType'>
```

In [2]:
```
# int
int_datatype = 13
print(f"The type of int_datatype is {type(int_datatype)}")
```

```
The type of int_datatype is <class 'int'>
```

In [3]:
```
# bytes
bytes_datatype = b"Hello Python!"
print(f"The type of bytes_datatype is {type(bytes_datatype)}")
```

```
The type of bytes_datatype is <class 'bytes'>
```

In [4]:
```
# bool
# bool datatype can only have either True or False. Integer value of True is 1 and False is 0.
bool_datatype = True
print(f"The type of bool_datatype is {type(bool_datatype)}")
```

```
The type of bool_datatype is <class 'bool'>
```

In [5]:
```
# float
float_datatype = 3.14
print(f"The type of float_datatype is {type(float_datatype)}")
```

```
The type of float_datatype is <class 'float'>
```

In [6]:
```
# complex
complex_datatype = 13 + 5j
print(f"The type of complex_datatype is {type(complex_datatype)}")
```

```
The type of complex_datatype is <class 'complex'>
```

In [7]:
```
# str
str_datatype = "Hey! Welcome to Python."
print(f"The type of str_datatype is {type(str_datatype)}")
```

```
The type of str_datatype is <class 'str'>
```

In [8]:
```
# tuple
tuple_datatype = (None, 13, True, 3.14, "Hey! Welcome to Python.")
print(f"The type of tuple_datatype is {type(tuple_datatype)}")
```

```
The type of tuple_datatype is <class 'tuple'>
```

In [9]:
```
# list
list_datatype = [None, 13, True, 3.14, "Hey! Welcome to Python."]
print(f"The type of list_datatype is {type(list_datatype)}")
```

```
The type of list_datatype is <class 'list'>
```

In [10]:
```
# set
set_datatype = {None, 13, True, 3.14, "Hey! Welcome to Python."}
print(f"The type of set_datatype is {type(set_datatype)}")
```

```
The type of set_datatype is <class 'set'>
```

In [11]:
```
# dict
dict_datatype = {
    "language": "Python",
    "Inventor": "Guido Van Rossum",
    "release_year": 1991,
}
print(f"The type of dict_datatype is {type(dict_datatype)}")
```

```
The type of dict_datatype is <class 'dict'>
```

Tidbits

The thing which I Love and Hate the most about Python is the dynamic typing. We might not know what are the types of parameters we might pass to a function or method. If you pass any other type of object as a parameter, **boom** you might see Exceptions raised 👻. Let's remember that **With great power comes great responsibility** 🕷

To help the developers with this, from Python 3.6 we have Type Hints(PEP-484).

We will get through these in the coming chapters. Stay tuned 😇