What is Programming?

“The act of writing software to perform a particular task using a programming language”.
Computer programming is the process of designing and building an executable computer program to accomplish a specific computing result or to perform a particular task.

To program you need to learn a high-level programming language its:

- Syntax
- Semantics

Computer Program

Set of Instruction → Programming Language → Sequence → Computer Hardware → to perform a specific task.

- A computer program is a set of instructions written in computer programming languages to perform a specific task for a computer.
- A computer program tells the computer that what to do and in which order to do.
- Different type of programming languages is used to develop programs.
- Some commonly used programming languages are Python, C, C++, C#, Java etc.,

What is Syntax?

The set of rules that defines the combinations of symbols that are correct in that language.
Spelling and grammar rules in natural language

Example syntax rules for **assignment statement**:

- Assignment statements is variable = Expression
- Expression is a variable or a Number

Correct is price = 1299
Incorrect is price = A

What is Semantics?

- Semantics is the interpretation or meaning of a written or spoken sentence.
- Example: an assignment statement In a programming language

Price = 1299
Means:

- Assign/Copy the value 1299 to variable price
- Set the value of variable price to 1299
- If you are a programmer, you would say assign the number 1299 to a variable price.

Note: *Syntax is how you have to write statements that are correct, semantics indicates what they mean.*

What is Pseudo-code?

- A way of writing software in a human readable form.
- English-like statements to outline the logic of a program.

```
1. Input: the number of hours worked this week
2. Input: hourly pay rate
3. Multiply "hours worked" with "hourly pay rate" and store the result in "gross pay"
4. Print "hours worked", "hourly pay rate" and "gross pay"
```

Python Keywords

Reserved words used in programming that have special meanings to the compiler. Keywords are part of the syntax and they cannot be used as an identifier. For example: int money; here, int is a keyword that indicates money is a variable of type int(integer).

Python has a set of keywords that are reserved words that cannot be used as variable names, function names, or any other identifiers:

Keyword	Description
and	A logical operator
as	To create an alias
break	To break out of a loop
class	To define a class
continue	To continue to the next iteration of a loop
def	To define a function
del	To delete an object
elif	Used in conditional statements, same as else if
else	Used in conditional statements
except	Used with exceptions, what to do when an exception occurs
False	Boolean value, result of comparison operations
for	To create a for loop
from	To import specific parts of a module
global	To declare a global variable
if	To make a conditional statement
import	To import a module
in	To check if a value is present in a list, tuple, etc.
is	To test if two variables are equal
lambda	To create an anonymous function

None	Represents a null value
nonlocal	To declare a non-local variable
not	A logical operator
or	A logical operator
pass	A null statement, a statement that will do nothing
raise	To raise an exception
return	To exit a function and return a value
True	Boolean value, result of comparison operations
try	To make a try...except statement
while	To create a while loop

Instructions

An instruction is a set of codes that the computer processor can understand. The code is usually in 1s and 0s, or machine language. It contains instructions or task that control the movement of bits and bytes within the processor.
Example of instruction sets: **Add** (Add two numbers together.)

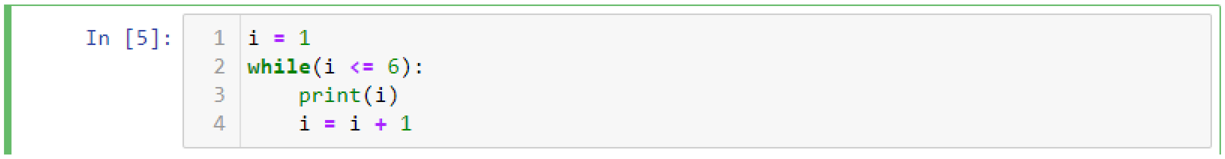
Indentation

Indentation in Python Programming is simply the spaces at the beginning of code line. Indentation in other languages like C, C++, C# etc., is just for readability, but in Python, the indentation is an essential and mandatory concept that should be followed when writing a python code; otherwise, the python interpreter throws Indentation Error.

What is Indentation in Python?

Indentation is the leading whitespace (spaces and tabs) before any statement in Python. The reason why indentation is important in python is that the indentation serves another purpose other than code readability. Python treats the statements with the same indentation level (statements with an equal number of whitespaces before them) as a single code block. So whereas in languages like c, c++, etc. a block of code is represented by Curly braces { }, in python a block is a group of statements that have the same Indentation level i.e same number of leading whitespaces.

Below is an example code snippet with correct indentation.

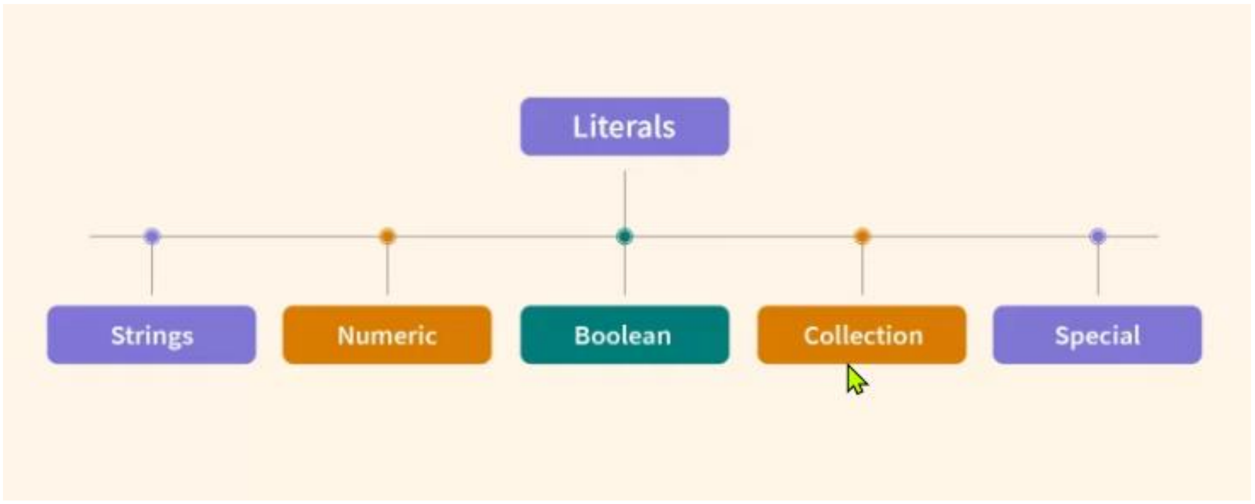

In [5]:
```
i = 1
while(i <= 6):
    print(i)
    i = i + 1
```

Literals

Generally, literals are a notation for representing a fixed value in source code. They can also be defined as raw value or data given in variables or constants.

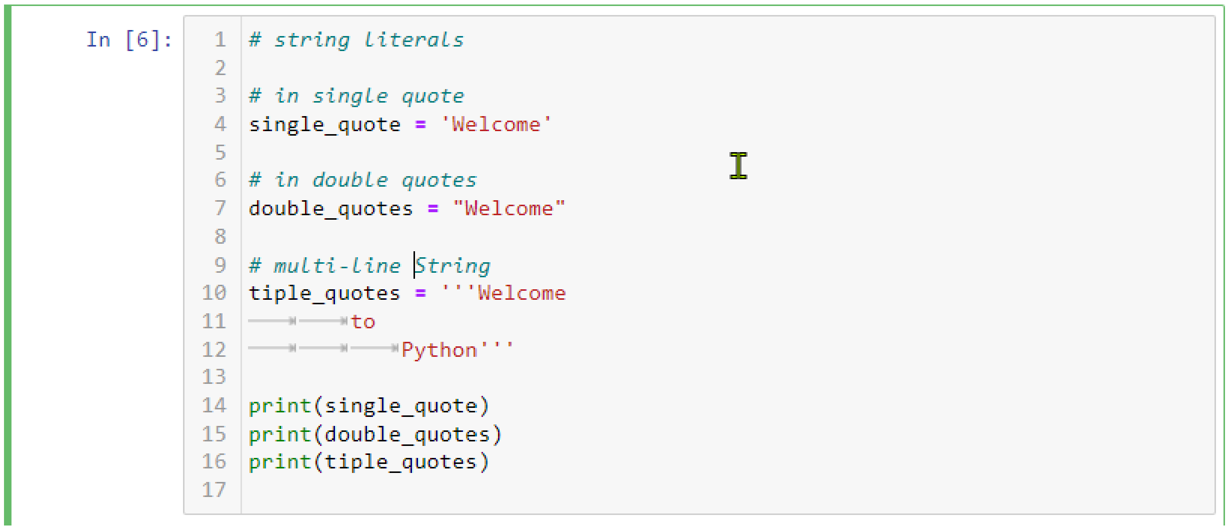
Python has different types of literals.

1. String Literals
2. Numeric Literals
3. Boolean Literals
4. Collection Literals
5. Special Literals



String Literals

A string literal can be created by writing a text (a group of characters) surrounded by the single(‘’), double(“”), or triple quotes. By using triple quotes we can write multi-line strings or display in the desired way.



In [6]:

```
# string literals

# in single quote
single_quote = 'Welcome'

# in double quotes
double_quotes = "Welcome"

# multi-line String
tiple_quotes = '''Welcome
		to
			Python'''

print(single_quote)
print(double_quotes)
print(tiple_quotes)
```

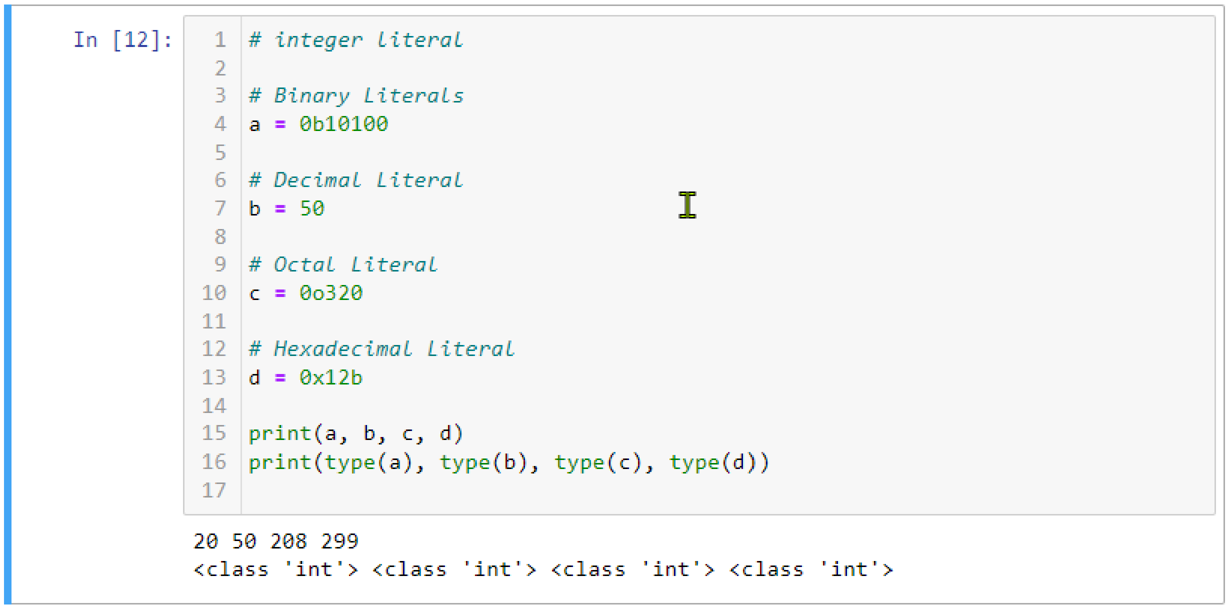
Numeric Literals

There are three types of numeric literal:

1. Integer
2. Float
3. Complex

Integer:

Both positive and negative number including 0. There should not be any fractional part.



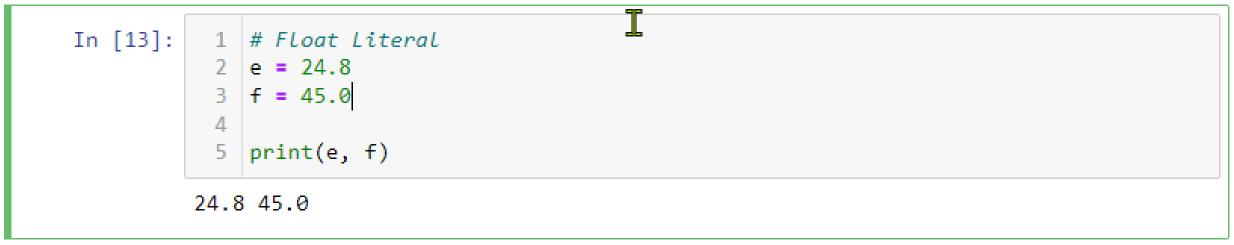
```python
# integer literal

# Binary Literals
a = 0b10100

# Decimal Literal
b = 50

# Octal Literal
c = 0o320

# Hexadecimal Literal
d = 0x12b

print(a, b, c, d)
print(type(a), type(b), type(c), type(d))

```

```
20 50 208 299
<class 'int'> <class 'int'> <class 'int'> <class 'int'>
```

In the above code we assigned integer literals (0b10100, 50, 0o320, 0x12b) into different variables. Here, 'a' is binary literal, 'b' is a decimal literal, 'c' is an octal literal and 'd' is a hexadecimal literal. But on using print function to display value or to get output they were converted into decimal.

Float

These are real numbers having both integer and fractional parts.

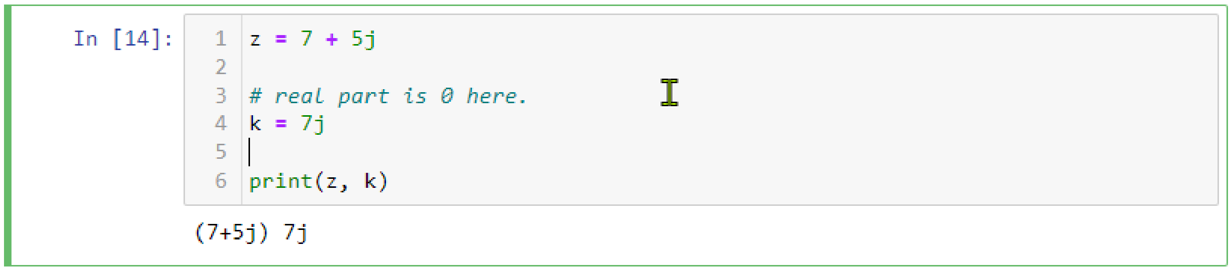


```python
# Float Literal
e = 24.8
f = 45.0

print(e, f)
```

```
24.8 45.0
```

Complex Literal

The numerals will be in the form of a + bj, where ‘a’ is the real part and ‘b’ is the complex part.

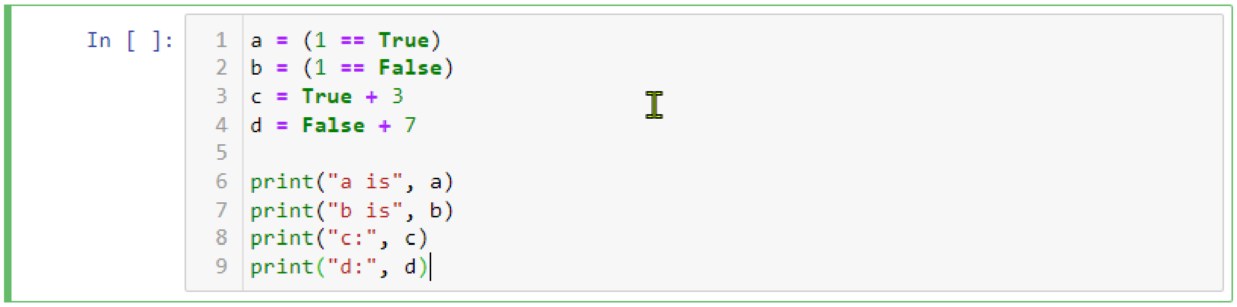


```
In [14]:
z = 7 + 5j

# real part is 0 here.
k = 7j

print(z, k)
```

```
(7+5j) 7j
```

Boolean Literals

There are only two Boolean literals in Python. They are true and false.

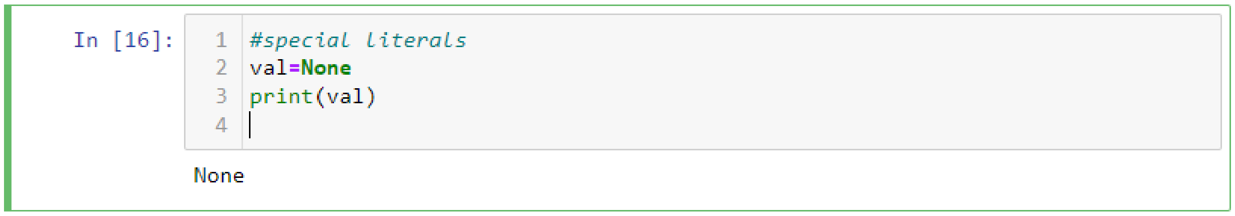


```
In [ ]:
a = (1 == True)
b = (1 == False)
c = True + 3
d = False + 7

print("a is", a)
print("b is", b)
print("c:", c)
print("d:", d)
```

Special Literal

Python literals have one special literal known as None. This literal in Python is used to signify that a particular field is not created.

Python will print None as output when we print the variable with no value assigned to it. None is also used for end of lists in Python.



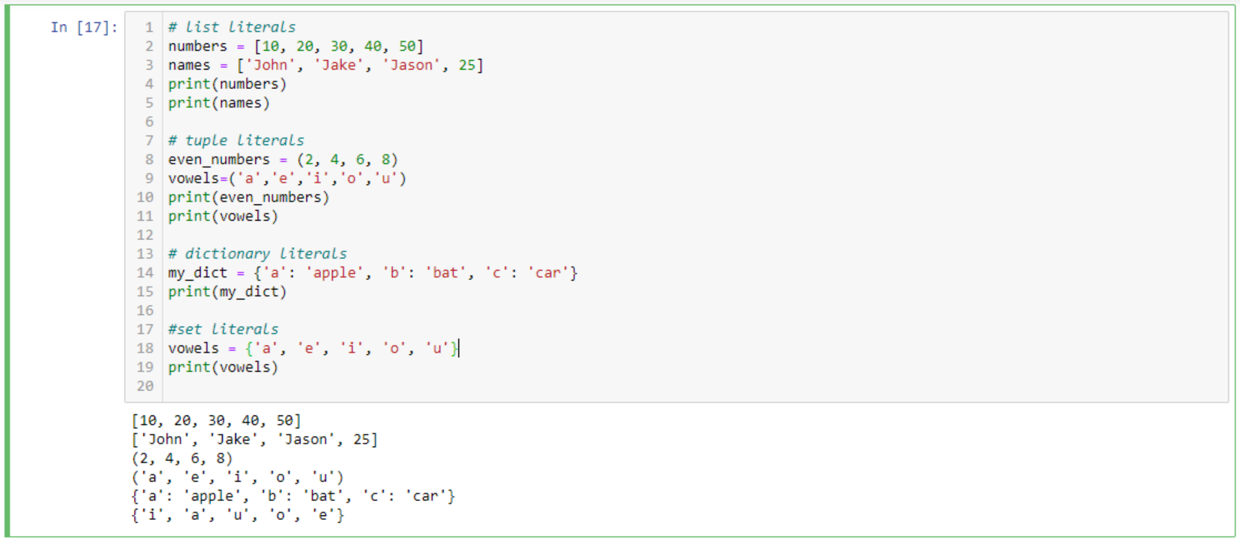
```
In [16]:
#special literals
val=None
print(val)
```

```
None
```

Literal Collection

If we wish to work with more than one value, then we can go for literal collections in Python. Literal collections in python are of four types.

1. List Literals
2. Tuple Literals
3. Dictionary Literal
4. Set Literal



In [17]:

```python
# list literals
numbers = [10, 20, 30, 40, 50]
names = ['John', 'Jake', 'Jason', 25]
print(numbers)
print(names)

# tuple literals
even_numbers = (2, 4, 6, 8)
vowels=('a','e','i','o','u')
print(even_numbers)
print(vowels)

# dictionary literals
my_dict = {'a': 'apple', 'b': 'bat', 'c': 'car'}
print(my_dict)

#set literals
vowels = {'a', 'e', 'i', 'o', 'u'}
print(vowels)

```

```
[10, 20, 30, 40, 50]
['John', 'Jake', 'Jason', 25]
(2, 4, 6, 8)
('a', 'e', 'i', 'o', 'u')
{'a': 'apple', 'b': 'bat', 'c': 'car'}
{'i', 'a', 'u', 'o', 'e'}
```