Python Keywords

- Keywords are reserved words in python.
- We cannot use a keyword as variable name, function name or any other identifier.
- Keywords are case sensitive

In [19]:
```
#Here's a list of all keywords in Python Programming
import keyword
print(keyword.kwlist)
print("\nTotal Number Of Keywords: ",len(keyword.kwlist))
```

```
['False', 'None', 'True', 'and', 'as', 'assert', 'break', 'class', 'continu
e', 'def', 'del', 'elif', 'else', 'except', 'finally', 'for', 'from', 'globa
l', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'r
aise', 'return', 'try', 'while', 'with', 'yield']

Total Number Of Keywords:  33
```

Description of some Keywords in Python with examples

True, False

- True and False are truth values in Python.
- They are the results of comparison operations or logical (Boolean) operations in Python. For example:

In [20]:
```
1 == 1
```

Out[20]: True

In [21]:
```
5 > 7
```

Out[21]: False

In [22]:
```
True or False
```

Out[22]: True

In [23]:
```
True and False
```

Out[23]: False

None

- None is a special constant in Python that represents the absence of a value or a null value.
- It is an object of its own datatype, the NoneType. We cannot create multiple None objects but can assign it to variables. These variables will be equal to one another.
- We must take special care that None does not imply False, 0 or any empty list, dictionary, string etc. For example:

In [24]:
```
None == 0
```

Out[24]: False

In [25]:
```
None == []
```

Out[25]: False

In [26]:
```
None == False
```

Out[26]: False

In [27]:
```
x = None
y = None
x == y
```

Out[27]: True

Void functions that do not return anything will return a None object automatically. None is also returned by functions in which the program flow does not encounter a return statement. For example:

In [28]:
```
def a_void_function():
    a = 1
    b = 2
    c = a + b

x = a_void_function()
print(x)
```

```
None
```

This program has a function that does not return a value, although it does some operations inside. So when we print x, we get None which is returned automatically (implicitly). Similarly, here is another example:

In [29]:
```
def improper_return_function(a):
    if (a % 2) == 0:
        return True

x = improper_return_function(3)
print(x)
```

```
None
```

and, or , not

- and, or, not are the logical operators in Python.
- 'and' will result into True only if both the operands are True.
- 'or' will result into True if any of the operands is True.
- 'not' operator is used to invert the truth value.

In [30]:
```
True and False
```

Out[30]: False

In [31]:
```
True or False
```

Out[31]: True

In [32]:
```
not False
```

Out[32]: True

In [33]:
```
import math as myAlias
myAlias.cos(myAlias.pi)
```

Out[33]: -1.0

as

- 'as' is used to create an alias while importing a module. It means giving a different name (user-defined) to a module while importing it.

'As' for example, Python has a standard module called math. Suppose we want to calculate what cosine pi is using an alias. We can do it as follows using as:

assert

- assert is used for debugging purposes.
- While programming, sometimes we wish to know the internal state or check if our assumptions are true. assert helps us do this and find bugs more conveniently. assert is followed by a condition.
- If the condition is true, nothing happens. But if the condition is false, AssertionError is raised. For example:

In [34]:
```
a = ['a','e','i','o','u']
assert (, "The Value of a is too small"
```

```
  File "<ipython-input-34-682fd72f53e6>", line 2
    assert (, "The Value of a is too small"
            ^
SyntaxError: invalid syntax
```

There are various other keywords.We will cover them all as we proceeds through course.

Python Identifiers

- Python Identifier is the name we give to identify a variable, function, class, module or other object. That means whenever we want to give an entity a name, that's called identifier.
- Sometimes variable and identifier are often misunderstood as same but they are not. Well for clarity, let's see what is a variable?

Variable in Python

- A variable, as the name indicates is something whose value is changeable over time. In fact a variable is a memory location where a value can be stored. Later we can retrieve the value to use. But for doing it we need to give a nickname to that memory location so that we can refer to it. That's identifier, the nickname.

Rules for writing Identifiers

- Identifier can be a combination of letters in lowercase(a to z) or uppercase(A to Z) or digits(0 to 9) or an underscore(_).
- An identifier cannot start with a digit.
- Keywords cannot be used as identifiers.

***Though these are hard rules for writing identifiers, also there are some naming conventions which are not mandatory but rather good practices to follow*.**

- Class names start with an uppercase letter. All other identifiers start with a lowercase letter.
- Starting an identifier with a single leading underscore indicates the identifier is private.
- If the identifier starts and ends with two underscores, than means the identifier is language-defined special name.
- While c = 10 is valid, writing count = 10 would make more sense and it would be easier to figure out what it does even when you look at your code after a long time.
- Multiple words can be separated using an underscore, for example this_is_a_variable.

Here's a sample program for python identifiers:

```
In [35]: abc = 13
         12abc = 12
```

```
  File "<ipython-input-35-51894e5d2703>", line 2
    12abc = 12
        ^
SyntaxError: invalid syntax
```

We cannot use special symbols like !,@,#,$,% etc in our identifier.

In [36]:
```
a@ = 10
```

```
  File "<ipython-input-36-b80c4a969d81>", line 1
    a@ = 10
       ^
SyntaxError: invalid syntax
```