In [1]:
```
import sys
import keyword
import operator
from datetime import datetime
import os
```

Keywords

Keywords are the reserved words in Python and can't be used as an identifier

In [2]:
```
print(keyword.kwlist) # List all Python Keywords
```

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

In [3]:
```
len(keyword.kwlist) # Python contains 35 keywords
```

Out[3]:
```
35
```

Identifiers

An identifier is a name given to entities like class, functions, variables, etc. It helps to differentiate one entity from another.

In [4]:
```
1var = 10 # Identifier can't start with a digit
```

```
  Cell In[4], line 1
    1var = 10 # Identifier can't start with a digit
    ^
SyntaxError: invalid decimal literal
```

In [ ]:
```
val2@ = 35 # Identifier can't use special symbols
```

In [ ]:
```
import = 125 # Keywords can't be used as identifiers
```

In [ ]:
```
"""
Correct way of defining an identifier
(Identifiers can be a combination of letters in lowercase (a to z) or uppercase """
val2 = 10
```

In [ ]:
```
val_ = 99
```

Comments in Python

Comments can be used to explain the code for more readabilty

In [ ]:
```
# Single line comment
val1 = 10
```

In [ ]:
```
# Multiple # line
# comment
val1 = 10
```

In [ ]:
```
'''
Multiple line
comment '''
val1 = 10
```

In [ ]:
```
"""
Multiple line
comment """
val1 = 10
```

Statements

Instructions that a Python interpreter can execute

In [ ]:
```
p = 20 #Creates an integer object with value 20 and assigns the variable p to p
q = 20 # Create new reference q which will point to value 20. p & q will be poi
r = q # variable r will also point to the same location where p & q are pointin
p , type(p), hex(id(p)) # Variable P is pointing to memory location '0x7fff6d71a
```

In [ ]:
```
q , type(q), hex(id(q))
```

In [ ]:
```
r , type(r), hex(id(r))
```

In [ ]:
```
p = 20
p = p + 10 # Variable Overwriting
p
```

Variable Assigment

In [ ]:
```
intvar = 10 # Integer variable
floatvar = 2.57 # Float Variable
strvar = "Python Language" # String variable
print(intvar)
print(floatvar)
print(strvar)
```

Multiple Assignments

In [ ]:
```
intvar , floatvar , strvar = 10,2.57,"Python Language" # Using commas to separat
print(intvar)
print(floatvar)
print(strvar)
```

Data Types

Numeric

In [ ]:
```
val1 = 10 # Integer data type
print(val1)
print(type(val1)) # type of object
print(sys.getsizeof(val1)) # size of integer object in bytes
print(val1, " is Integer?", isinstance(val1, int)) # val1 is an instance of int
```

In [ ]:
```
val2 = 92.78 # Float data type
print(val2)
print(type(val2)) # type of object
print(sys.getsizeof(val2)) # size of float object in bytes
print(val2, " is float?", isinstance(val2, float)) # Val2 is an instance of floa
```

In [ ]:
```
val3 = 25 + 10j # Complex data type
print(val3)
print(type(val3)) # type of object
print(sys.getsizeof(val3)) # size of float object in bytes
print(val3, " is complex?", isinstance(val3, complex)) # val3 is an instance of
```

In [ ]:
```
sys.getsizeof(int()) # size of integer object in bytes
```

In [ ]:
```
sys.getsizeof(float()) # size of float object in bytes
```

In [ ]:
```
sys.getsizeof(complex()) # size of complex object in bytes
```

Boolean

Boolean data type can have only two possible values true or false.

In [ ]:
```
bool1 = True
```

In [ ]:
```
bool2 = False
```

In [ ]:
```
print(type(bool1))
```

In [ ]:
```
print(type(bool2))
```

In [ ]:
```
isinstance(bool1, bool)
```

In [ ]:
```
bool(0)
```

In [ ]:
```
bool(1)
```

In [ ]:
```
bool(None)
```

In [ ]:
```
bool (False)
```

Strings

In [ ]:
```
str1 = "HELLO PYTHON"
```

In [ ]:
```
print(str1)
```

In [ ]:
```
mystr = 'Hello World' # Define string using single quotes
print(mystr)
```

In [ ]:
```
mystr = "Hello World" # Define string using double quotes
print(mystr)
```

In [ ]:
```
mystr = '''Hello
World ''' # Define string using triple quotes
print(mystr)
```

In [ ]:
```
mystr = """Hello
World""" # Define string using triple quotes
print(mystr)
```

In [ ]:
```
mystr = ('Happy '
'Monday ' 'Everyone')
print(mystr)
```

In [ ]:
```
mystr2 = 'Woohoo '
mystr2 = mystr2*6
mystr2
```

In [ ]:
```
len(mystr2) # Length of string
```

String Indexing

Forward Indexing

Backward Indexing

In [ ]:
```
str1
```

In [ ]:
```
str1[0] # First character in string "str1"
```

In [ ]:
```
str1[len(str1)-1] # Last character in string using len function
```

In [ ]:
```
str1[-1] # Last character in string
```

In [ ]:
```
str1[6] #Fetch 7th element of the string
```

In [ ]:
```
str1[5]
```

String Slicing

In [ ]:
```
str1[0:5] # String slicing - Fetch all characters from 0 to 5 index location exc
```

In [ ]:
```
str1[6:12] # String slicing - Retreive all characters between 6 - 12 index loc e
```

In [ ]:
```
str1[-4:] # Retreive last four characters of the string
```

In [ ]:
```
str1[-6:] # Retreive last six characters of the string
```

In [ ]:
```
str1[:4] # Retreive first four characters of the string
```

In [ ]:
```
str1[:6] # Retreive first six characters of the string
```

Update & Delete String

In [ ]:
```
str1
```

In [ ]:
```
#Strings are immutable which means elements of a string cannot be changed once t
str1[0:5] = 'HOLAA'
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

2 Aug 2025

In [6]:
```
l1 = []
```

In [7]:
```
l1 = [10, 2.3, 1+2j, True, 'hello', [1,2,3]]
```

In [8]:
```
l1
```

Out[8]:
```
[10, 2.3, (1+2j), True, 'hello', [1, 2, 3]]
```

In [9]:
```
l1.count(10)
```

Out[9]:
```
1
```

In [10]:
```
l1.count(2.3)
```

Out[10]:
```
1
```

In [11]:
```
l1
```

Out[11]:
```
[10, 2.3, (1+2j), True, 'hello', [1, 2, 3]]
```

In [12]:
```
l1.remove(1+2j)
```

In [13]:
```
l1
```

Out[13]:
```
[10, 2.3, True, 'hello', [1, 2, 3]]
```

```
In [14]: l1.pop()
Out[14]: [1, 2, 3]

In [15]: l1
Out[15]: [10, 2.3, True, 'hello']

In [17]: l1.pop()
Out[17]: 'hello'

In [18]: l1
Out[18]: [10, 2.3, True]

In [19]: l1.remove(True)

In [20]: l1
Out[20]: [10, 2.3]

In [22]: l1
Out[22]: [10, 2.3]

In [31]: l1
Out[31]: [10, 2.3]

In [32]: l1.insert(3,'nit')

In [33]: l1
Out[33]: [10, 2.3, 'nit']

In [34]: l1.append('hello')

In [35]: l1
Out[35]: [10, 2.3, 'nit', 'hello']

In [36]: l2 = [6, 7, 9, 'nit']

In [37]: l2
Out[37]: [6, 7, 9, 'nit']

In [38]: print (l1, l2)
         [10, 2.3, 'nit', 'hello'] [6, 7, 9, 'nit']
```

In [40]:
```
l1.extend(l1)
```

In [41]:
```
l1
```

Out[41]:
```
[10, 2.3, 'nit', 'hello', 10, 2.3, 'nit', 'hello']
```

In [42]:
```
l1
```

Out[42]:
```
[10, 2.3, 'nit', 'hello', 10, 2.3, 'nit', 'hello']
```

In [43]:
```
l1.reverse()
```

In [44]:
```
l1
```

Out[44]:
```
['hello', 'nit', 2.3, 10, 'hello', 'nit', 2.3, 10]
```

In [45]:
```
l5 = [300, 3, 34, 9, 100]
```

In [46]:
```
l5
```

Out[46]:
```
[300, 3, 34, 9, 100]
```

In [50]:
```
l5.sort()
```

In [51]:
```
l5
```

Out[51]:
```
[3, 9, 34, 100, 300]
```

In [52]:
```
l5.sort(reverse=True)
```

In [53]:
```
l5
```

Out[53]:
```
[300, 100, 34, 9, 3]
```

In [54]:
```
l5[0]= 3000
```

In [55]:
```
l5
```

Out[55]:
```
[3000, 100, 34, 9, 3]
```

list is completed

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: