Lists

In [1]:
```
current_values = [5, 6, 7]
```

In [2]:
```
print(current_values)
```

```
[5, 6, 7]
```

In [3]:
```
current_values[1] = 2
print(current_values)
```

```
[5, 2, 7]
```

In [4]:
```
current_values[-1]
```

Out[4]:
```
7
```

In [5]:
```
current_values[1]
len(current_values)
```

Out[5]:
```
3
```

In [6]:
```
5 in current_values
```

Out[6]:
```
True
```

In [7]:
```
100 in current_values
```

Out[7]:
```
False
```

In [8]:
```
def start_car(todo):
    for task in todo:
        print(task, "is working correctly", end=' ')

    print("This futuristic car is very obliiged to start now..", end=' ')
```

In [9]:
```
tasks = ['GPS system', 'Brakes', 'Sensors', 'Engine']
start_car(tasks)
```

```
GPS system is working correctly Brakes is working correctly Sensors is working correctly Engine is working correctl
y This futuristic car is very obliiged to start now..
```

Other ways for List representation

In [10]:
```
voltage_val = [10, 20, 30, 40]
#               0   1   2   3
#              -4  -3  -2  -1
```

In [11]:
```
voltage_val[ : :2]
```

Out[11]:
```
[10, 30]
```

Nested Sequences

In [12]:
```
wires = [
            [5, 0],
            [10, 0],
            [15, 0]
                        ]
```

In [13]:
```
for i in range(0,3):
    for j in range(0, 2):
        print(wires[i][j])
```

```
5
0
10
0
15
0
```

In [14]:
```
for pair in wires:
    print(pair)
    pos = pair[0]
    neg = pair[1]
    print(pos, neg, '\n')
```

```
[5, 0]
5 0

[10, 0]
10 0

[15, 0]
15 0
```

In [15]:
```
for pos, neg in wires:
    print(pos, neg)
```

```
5 0
10 0
15 0
```

In [16]:
```
help(list.insert)   # Or can use list.insert?
```

```
Help on method_descriptor:

insert(...)
    L.insert(index, object) -- insert object before index
```

In [17]:
```
help(list)
```

```
 |      Return self>value.
 |
 |  __iadd__(self, value, /)
 |      Implement self+=value.
 |
 |  __imul__(self, value, /)
 |      Implement self*=value.
 |
 |  __init__(self, /, *args, **kwargs)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  __iter__(self, /)
 |      Implement iter(self).
 |
 |  __le__(self, value, /)
 |      Return self<=value.
 |
 |  __len__(self, /)
 |      Return len(self).
 |
```

Dictionary

In [18]:
```
cables = {
    "wire_1": "Network cable",    # Key must be immutable i.e number, string, tuple  but  Value can be anything
    "wire_2": "Main office cable"
}
```

In [19]:
```
cables["wire_1"]
```

Out[19]:
```
'Network cable'
```

In [20]:
```
for key in cables:
    print(key, " = ", cables[key])
```

```
wire_1  =  Network cable
wire_2  =  Main office cable
```

In [21]:
```
cables.items()  # A dict object which has list of tuples
```

Out[21]:
```
dict_items([('wire_1', 'Network cable'), ('wire_2', 'Main office cable')])
```

In [22]:
```
for key, value in cables.items():   # Because we know that each element is tuple
    print(key, " = ", value)
```

```
wire_1  =  Network cable
wire_2  =  Main office cable
```