Looping

- For-Loops
- While-loops

1. While-Loop

Python provides several constructs to repeatedly executes block of statements so long as some conditions remain true.

- **`while`** loop
- **`for`** loop

1.1 While-Loop

As long as `condition` remains `True`, `statement` will be executed repeatedly, i.e. in infinite loop.

- It is important that the condition will eventually become `False`
- The statement may not execute at all if condition is `False`

```
while <condition>:
    <statement>
    <update-condition>
```

Exercise:

Use while-loop to print out `0 1 2 3 end`.

In [2]:
```
1 i = 0
2 while i < 3:
3     print(i, end=' ')
4     i = i + 1
5
6 print('end')
```

```
0 1 2 end
```

Exercise:

Sum up all values in a list `data = [1,2,3,4]` using `while-loop`.

- Use result to print out message `Sum of [1, 2, 3, 4] = 10`.

In [2]:
```
# Sum all the items in the list
data = [1,2,3,4]
result = 0

i = 0
while i < len(data):
    result += data[i]
    i += 1

print('Sum of {} = {}'.format(data, result))
```

```
Sum of [1, 2, 3, 4] = 10
```

1.2 Break from Loop

If we set the `condition` to `True`, the while-loop becomes a infinite loop.

```
while True:
    <statement_1>
```

To break out from a while-loop, use `break` clause. It is commonly used together with an `if` statement.

- If `condition` is `True`, execution will break from the while loop and `statement_2` will not be executed.

```
while True:
    <statement_1>
    if <condition>:
        break
    <statement_2>
```

Exercise:

Use `while True` and `break` to print out `0 1 2 3 end`.

In [4]:
```
i = 0
while True:
    print(i, end=' ')
    if i>=3:
        break
    i = i + 1

print('end')
```

```
0 1 2 3 end
```

1.3 Skip An Iteration

While in the loop, you can use `continue` clause to skip current iteration and continue to the next iteration.

```
while <condition_1>:
    <statement_1>
    if <condition_2>:
        continue # Skip current iteration
    <statement_2>
```

In [5]:
```
i = 0
while i < 10:
    i = i + 1
    if i % 2 == 0:
        continue
    print(i, end = ' ')
```

```
1 3 5 7 9
```

2. For-Loop

A `for` loop provides a mean to perform actions for all items in an iterables.

Iterables can be strings, tuples, lists, dictionaries, ranges, etc.

```
for <item> in <iterable>:
    statement
```

In [7]:
```
for x in [1,2,3]:
    print(x, end=' ')
```

```
1 2 3
```

Loops can be nested together.

```
for <item> in <iterable>:
    for <item> in <iterable>:
        <statement>
```

Exercise:

Use nested loop to print nested list `num = [[1,2,3],[4,5,6,7],[8,9]]` .

Output:

```
1 2 3
4 5 6 7
8 9
```

In [8]:
```
num = [[1,2,3],[4,5,6,7],[8,9]]

for x in num:
    for y in x:
        print(y, end=' ')
    print()
```

```
1 2 3
4 5 6 7
8 9
```

2.1 Function `range()`

The `range()` function is used to generate a sequence of numbers. It takes in parameter `start`, `stop` and `step`

```
range([start,] stop [, step]) -> range object
```

- The `start` parameter is optional, with default value = 0
- The `stop` value is an exclusive bound
- The `step` parameter is optional, with default value = 1

`range(5)` generates numbers `[0, 1, 2, 3, 4]`.

A `range` object can be converted to `list` object using `list()`.

Exercise:

Generate a list of integer numbers between 5 and 9.

In [10]:
```
list(range(5,10))
```

Out[10]: `[5, 6, 7, 8, 9]`

2.2 Use `range()` in For Loop

For-loop automatically convert `range` object to `iterable` object, and iterate through its elements.

In [12]:
```
for i in range(5):
    print(i, end=' ')
```

```
0 1 2 3 4
```

2.3 Break from Loop

The same `break` clause can be used to break out of `for` loop.

```
for <item> in <iterable>:
    statements
    if <condition>:
        break  # Break the iteration
    statements
```

Exercise:

Find first integer can be divided by 2, 3 and 5.

In [13]:
```
for i in range(1,100):
    if i%2==0 and i%3==0 and i%5==0:
        break
print(i)
```

```
30
```

Exercise:

Use nested-for loop to print following patterns.

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
1 2 3 4 5 6
1 2 3 4 5 6 7
1 2 3 4 5 6 7 8
1 2 3 4 5 6 7 8 9
```

In [14]:
```
for x in range(1, 10):
    for y in range(1, 10):
        print(y, end=' ')
        if(y>=x):
            break
    print()
```

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
1 2 3 4 5 6
1 2 3 4 5 6 7
1 2 3 4 5 6 7 8
1 2 3 4 5 6 7 8 9
```

2.4 Skip an Iteration

Use `continue` clause to terminate current iteration and continue to the next iteration of the loop.

```
for <item> in <iterable>:
    statements
    if <condition>:
        continue  # Skip current iteration
    statements
```

Exercise:

Print all numbers between 1 and 99 which can be divided by 2, 3 and 5.

In [15]:

```
for i in range(1, 100):
    if i%2!=0 or i%3!=0 or i%5!=0:
        continue
    print(i, end=' ')
```

```
30 60 90
```