Control Flow Statements (Part 2)

- 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>
```

Try Code:

```
i = 0
while i < 3:
    print(i)
    i = i + 1

print('the end')
```

In [ ]:

```
```

Exercise:

Sum up all values in a list `data = [1,2,3,4]` using `while-loop` . Print out the result `Sum of [1, 2, 3, 4] = 10` .

In [ ]:
```
# 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))
```

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>
```

Try Code:

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

In [ ]:
```
```

Exercise:

Ask user to input a list of messages, stop when user enters a entry message. Print out the list of messages.

Sample Outputs:

```
Type multiple lines of messages: (Enter empty string to end)
Hello world
Hi there

['Hello world', 'Hi there']
```

In [ ]:

```
print('Type multiple lines of messages: (Enter empty string to end)')
messages = []
while True:
    msg = input()
    if msg == '':
        break
    messages.append(msg)

print(messages)
```

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>
```

Try Code: Print out all odd numbers below 10.

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

In [ ]:

```

```

Exercise:

Ask user to input 3 positive numbers. Use `while-loop` to collect the numbers, skip if user enter negative number.

```
Enter 3 positive numbers:
10
-5
Positive value only
20
30
[10, 20, 30]
```

In [ ]:
```
i = 0
print("Enter 3 positive numbers:")
result = []
while i < 3:
    num = int(input())
    if num <=0:
        print('Positive value only')
        continue
    result.append(num)
    i += 1
print(result)
```

```
Enter 3 positive numbers:
```

1.4 What Else if Loop Ends *Naturally* (Optional)

Use `else` clause to execute some statements when the `while` loop terminates *naturally*.

```
while <condition>:
    <statement>
else:
    <statement> # Execute at the end of the last iteraction
```

Try Out:

Try running following code; Uncomment the 3 lines and try again.

```
data = [1,2,3,4,5]
sum = 0

i = 0
while i < len(data):
    sum = sum + data[i]
    i += 1
#     if i == 3:
#         print('break out of while')
#         break
else:  # These statements are executed at the end of the loop
    print(sum)
```

```
In [ ]:
```

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
```

Try Code:

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

```
In [ ]:
```

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

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

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 between 0 and 4.

- Start value = 0
- Stop value = 5
- Step value = 1

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

In [ ]:
```
r = range(5)
print(type(r))
print(list(r))
```

Exercise:

Generate a list of integer numbers between 5 and 9.

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

Exercise:

Generate a list of even numbers between 20 and 30.

In [ ]:
```
list(range(20,31,2))
```

2.2 Use `range()` in For Loop

It's common to use range to setup iteration in for-loop.

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

Try Code:

```
for i in range(5):
    print(i, end=' ')
```

In [ ]:
```
```

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

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

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

2.5 What Else if Loop Ends *Naturally* (Optional)

Use `else` clause to execute some statements when the `for` loop terminates *naturally*.

```
for <item> in <iterable>:
    statements
else:
    statements # Execute after last iteraction
```

Try Code:

Try following code. Uncomment 3 lines and try again. Is the sum still printed out?

```
data = [1,2,3,4,5,6,7,8,9,10]
sum = 0

for n in data:
    sum += n
#     if n > 5:
#         print('Break out of loop')
#         break
else:
    print(sum)
```

In [ ]:
```
```