# Day - 3

2

# Python Scripting

The concept of SUITES

# Getting Input From Console

- The input() function :-
- Syntax :-

  input([prompt])

  - prompt – If specified, it is written to standard output without a trailing newline.
  - The function then reads a line from input.
  - Then converts it to string (stripping a trailing newline) and returns that.
  - When EOF is read, EOFError is raised.

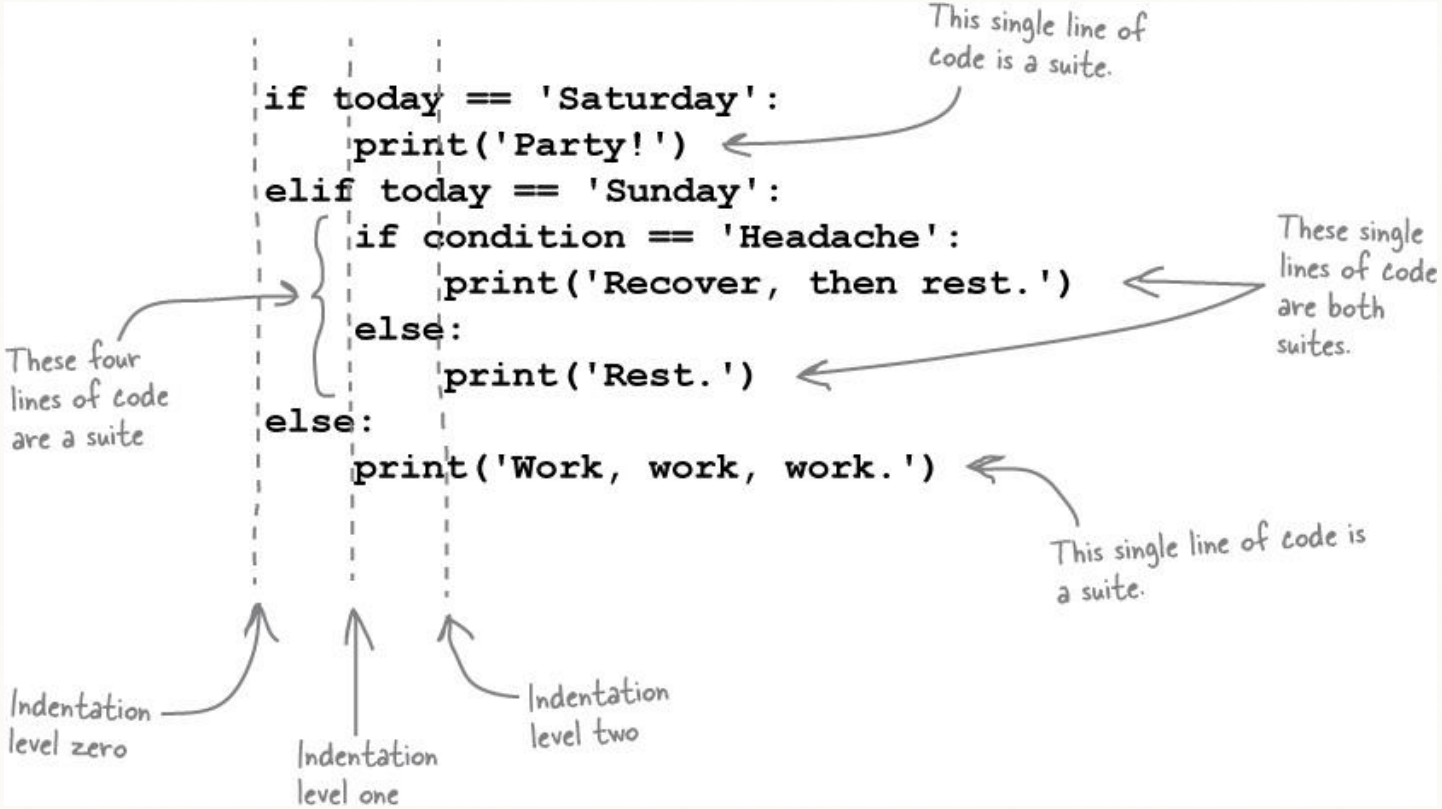
- Example:

```
a=input("Enter something in console")
print(type(a))
print(a)
```

```
Type "copyright", "credits" or "license()" f
>>>
================== RESTART: C:\Users\Himansh
Enter something in consolesomething
<class 'str'>
something
>>>
```

# The Python SUITE

- The role of Indentation is very significant in Python



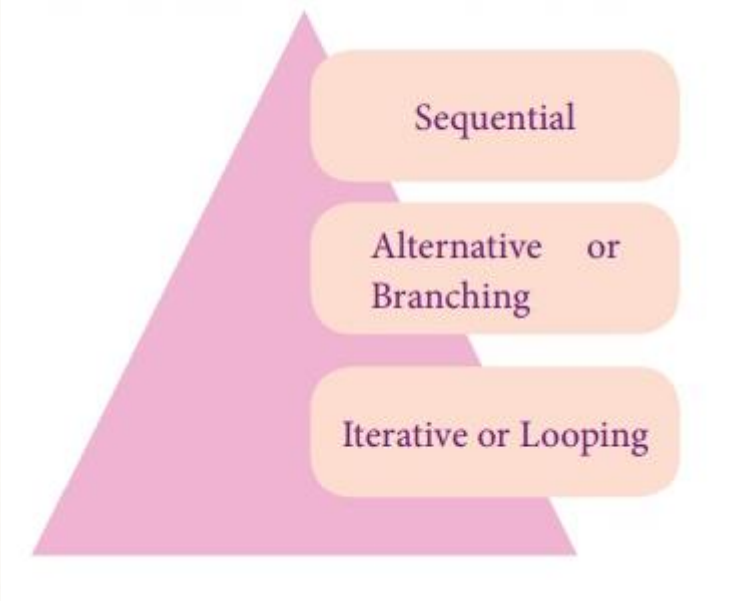
5

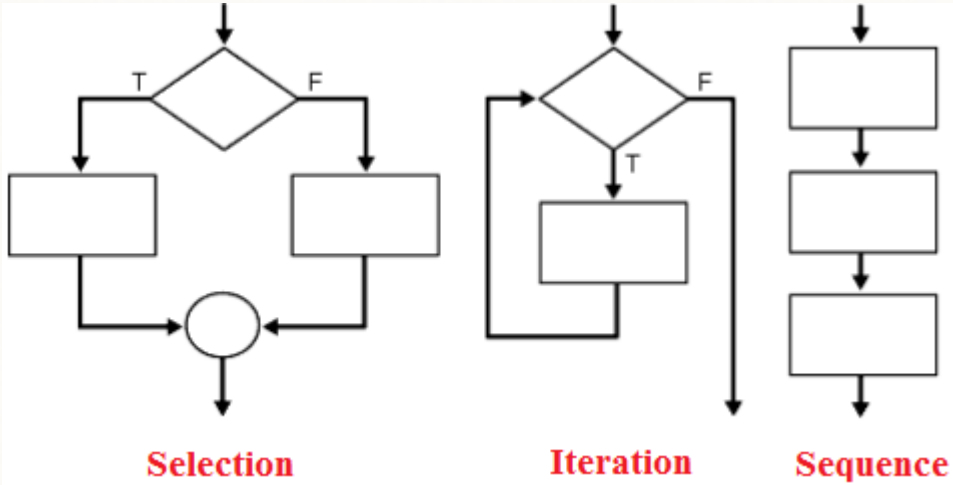
# Control Statements

Python Programming

# Types

- There are three important Control Structures in Python



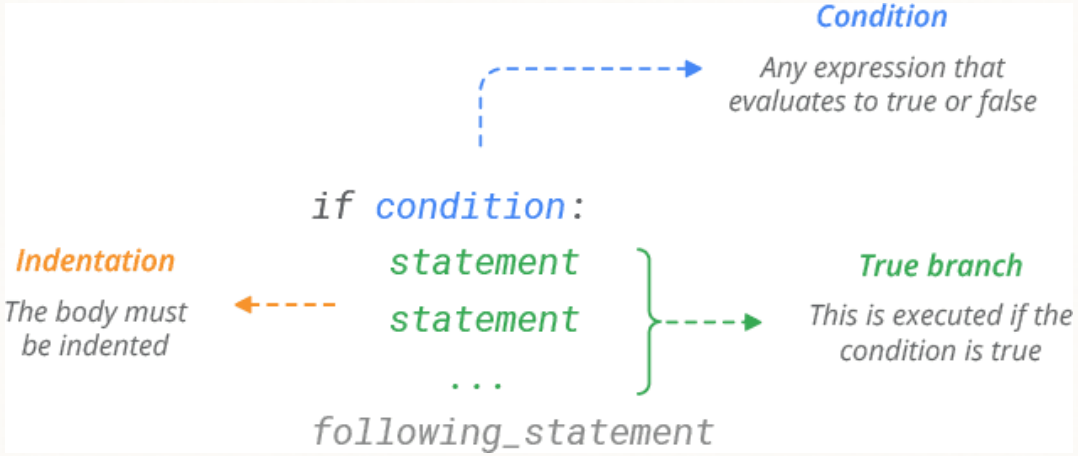



7

# IF-ELIF-ELSE

Control Statements

# if statement FLOW




```
x, y = 7, 5
if not x < y:
    print('x is greater')

# Prints x is greater
```

```
x, y = 7, 5
if x > y:
    print('x is greater')

# Prints x is greater
```

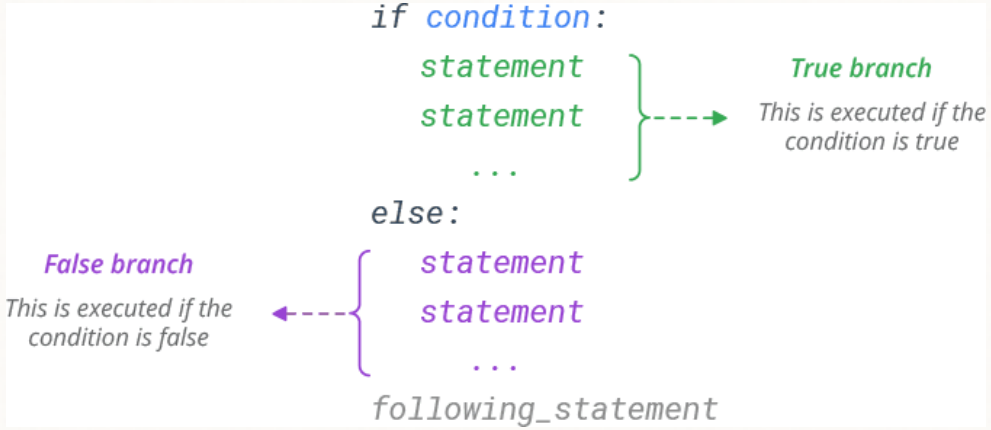
```
x, y, z = 7, 4, 2
if x > y and x > z:
    print('x is greater')

# Prints x is greater
```

```
x, y, z = 7, 4, 9
if x > y or x > z:
    print('x is greater than y or z')

# Prints x is greater than y or z
```

# if-else FLOW




```
x, y = 7, 5
if x < y:
    print('y is greater')
else:
    print('x is greater')

# Prints x is greater
```

# if-elif-else FLOW

```
if condition:
    statement
    statement
        ...
elif condition:
    statement
    statement
        ...
else:
    statement
    statement
        ...
following_statement
```

**First condition**

*This is executed if the first condition is true*

**New condition**

*A new condition to test if previous condition isn't true*

**False branch**

*This is executed if none of the conditions are true*

```python
x, y = 5, 5
if x > y:
    print('x is greater')
elif x < y:
    print('y is greater')
else:
    print('x and y are equal')

# Prints x and y are equal
```

# Conditional Expression (Ternary Operator)

```
variable = statement if condition else statement
```

*True branch*

*Execute this statement, if the condition is true*

*False branch*

*Execute this statement, if the condition is false*

```python
x, y = 7, 5
print('x is greater') if x > y else print('y is greater')

# Prints x is greater
```

```python
x, y = 7, 5
max = x if x > y else y
print(max)
# Prints 7
```

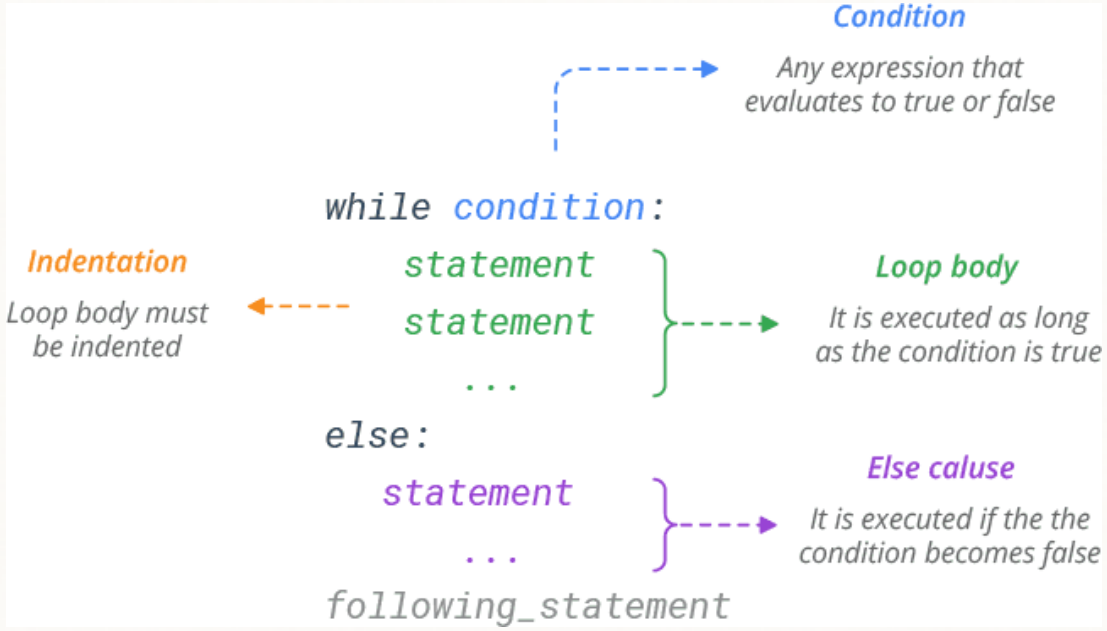
12

# WHILE

Loop Statements

# while

- A while loop is used when you want to perform a task indefinitely, until a particular condition is met. It's a condition-controlled loop.




```python
# Iterate until string is empty
x = 'blue'
while x:
    print(x)
    x = x[1:]
# Prints blue
# Prints lue
# Prints ue
# Prints e
```

# While loop examples

```python
# Iterate until x becomes 0
x = 6
while x:
    print(x)
    x -= 1
# Prints 6 5 4 3 2 1
```

```python
# Iterate until list is empty
L = ['red', 'green', 'blue']
while L:
    print(L.pop())
# Prints blue green red
```

```python
# Iterate until string is empty
x = 'blue'
while x:
    print(x)
    x = x[1:]
# Prints blue
# Prints lue
# Prints ue
# Prints e
```

```python
# Exit condition is false at the start
x = 0
while x:
    print(x)
    x -= 1
```

# Break and Continue

- Python break statement is used to exit the loop immediately. It simply jumps out of the loop altogether, and the program continues after the loop.

```
# Exit when x becomes 3
x = 6
while x:
    print(x)
    x -= 1
    if x == 3:
        break
# Prints 6 5 4
```

- The continue statement skips the current iteration of a loop and continues with the next iteration.

```
# Skip odd numbers
x = 6
while x:
    x -= 1
    if x % 2 != 0:
        continue
    print(x)
# Prints 4 2 0
```

# Else in While Loop

```
x = 6
while x:
    print(x)
    x -= 1
else:
    print('Done!')
# Prints 6 5 4 3 2 1
# Prints Done!
```

```
x = 0
while x:
    print(x)
    x -= 1
else:
    print('Done!')
# Prints Done!
```

- If the loop terminates prematurely with break, the else clause won't be executed.

```
x = 6
while x:
    print(x)
    x -= 1
    if x == 3:
        break
else:
    print('Done!')
# Prints 6 5 4
```

# Infinite Loop

```python
# Infinte loop with while statement
while True:
    print('Press Ctrl+C to stop me!')
```

```python
# Loop runs until the user enters 'stop'
while True:
    name = input('Enter name:')
    if name == 'stop': break
    print('Hello', name)

# Output:
# Enter name:Bob
# Hello Bob
# Enter name:Sam
# Hello Sam
# Enter name:stop
```

18

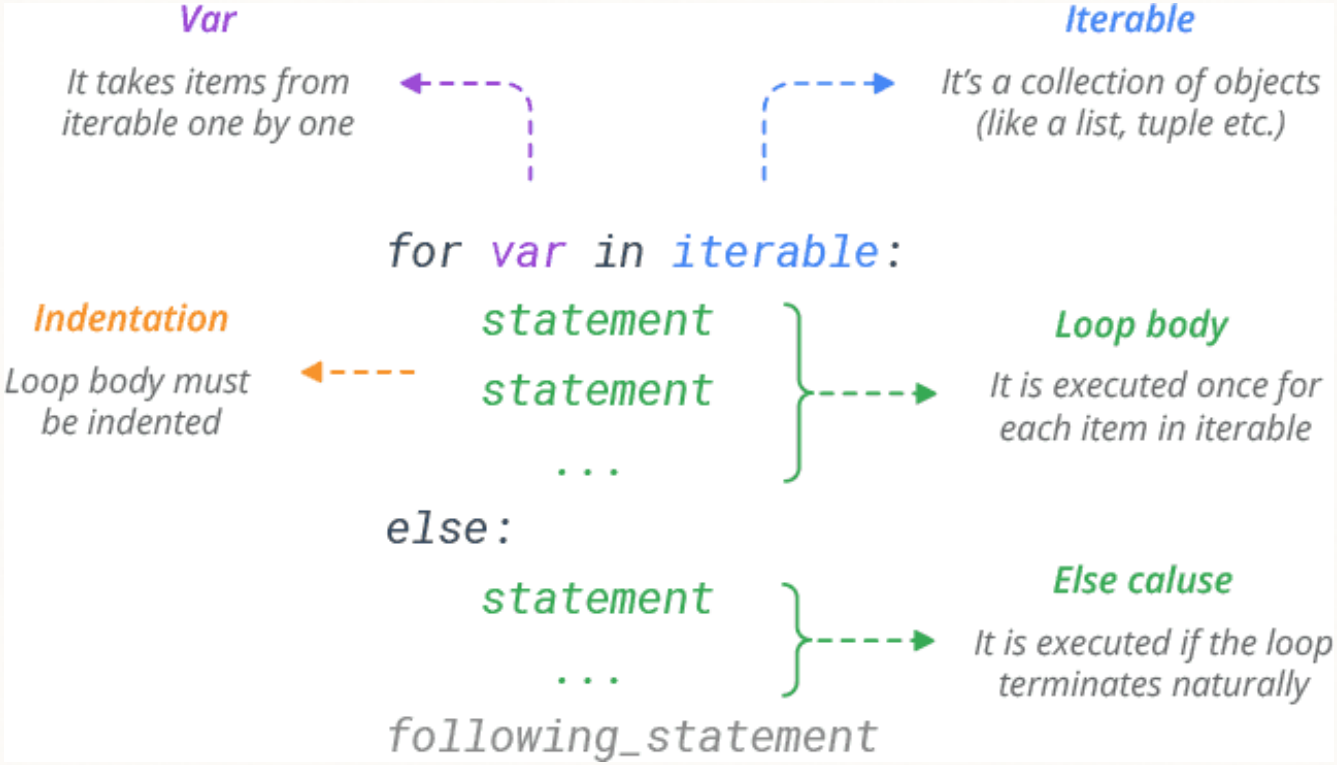
# FOR

Loop Statements

# FOR Loop

- The for statement in Python is a bit different from what you usually use in other programming languages.
- Rather than iterating over a numeric progression, Python's ***for*** statement iterates over the items of any *iterable* (list, tuple, dictionary, set, or string).
- The items are iterated in the order that they appear in the *iterable*.

# For Loop contd.




```python
# Iterate through a list
colors = ['red', 'green', 'blue', 'yellow']
for x in colors:
    print(x)
# Prints red green blue yellow
```

```python
# Iterate through a string
S = 'python'
for x in S:
    print(x)
# Prints p y t h o n
```

```python
# Flatten a nested list
list = [[1, 2, 3],[4, 5, 6],[7, 8, 9]]
for sublist in list:
    for number in sublist:
        print(number)
# Prints 1 2 3 4 5 6 7 8 9
```

# Break and Continue in For Loop

```python
# Break the loop at 'blue'
colors = ['red', 'green', 'blue', 'yellow']
for x in colors:
    if x == 'blue':
        break
    print(x)
# Prints red green
```

```python
# Skip 'blue'
colors = ['red', 'green', 'blue', 'yellow']
for x in colors:
    if x == 'blue':
        continue
    print(x)
# Prints red green yellow
```

# Else in FOR Loop

```
colors = ['red', 'green', 'blue', 'yellow']
for x in colors:
    print(x)
else:
    print('Done!')
# Prints red green blue yellow
# Prints Done!
```

```
colors = ['red', 'green', 'blue', 'yellow']
for x in colors:
    if x == 'blue':
        break
    print(x)
else:
    print('Done!')
# Prints red green
```

# Using range( ) function

- The range(start,stop,step) function generates a sequence of numbers from 0 up to (but not including) specified number.

```
# Print 'Hello!' three times
for x in range(3):
    print('Hello!')
# Prints Hello!
# Prints Hello!
# Prints Hello!
```

```
# Generate a sequence of numbers from 2 to 6
for x in range(2, 7):
    print(x)
# Prints 2 3 4 5 6
```

```
for x in range(-5,0):
    print(x)
# Prints -5 -4 -3 -2 -1
```

```
# Increment the range with 2
for x in range(2, 7, 2):
    print(x)
# Prints 2 4 6
```

# FOR Loop Operations

- Lopping through multiple Lists

```
# Loop through two lists at once
name = ['Bob', 'Sam', 'Max']
age = [25, 35, 30]
for x, y in zip(name, age):
    print(x, y)
# Prints Bob 25
# Prints Sam 35
# Prints Max 30
```

- Access Index in Loop

```
colors = ['red', 'green', 'blue']
for index in range(len(colors)):
    print(index, colors[index])
# Prints 0 red
# Prints 1 green
# Prints 2 blue
```

```
colors = ['red', 'green', 'blue']
for index, value in enumerate(colors):
    print(index, value)
# Prints 0 red
# Prints 1 green
# Prints 2 blue
```

25

# Comprehension

Loop Statements

# Comprehensions

- The Comprehensions are one-liner elegant approach to create the respective object
- In Python we have four comprehensions available:
  - List [output for-loop ]
  - Dictionary { key:value for-loop }
  - Set { output for-loop }
  - Generator ( output for-loop )
- The basic syntax of any comprehension does have two forms
  - `Bracket_open <output> for-loop Bracket_close`
  - `Bracket_open <output> for-loop if-condition Bracket_close`

# List Comprehension

```
[expression for var in iterable]
```

| *Expression* | *Var* | *Iterable* |
| --- | --- | --- |
| *It is evaluated once for each item in iterable* | *It takes items from an iterable one by one* | *It's a collection of objects (like a list, tuple etc.)* |

```python
L = [x*3 for x in 'RED']
print(L)
# Prints ['RRR', 'EEE', 'DDD']
```

```python
# Convert list items to absolute values
vec = [-4, -2, 0, 2, 4]
L = [abs(x) for x in vec]
print(L)
# Prints [4, 2, 0, 2, 4]
```

```python
L = [(x, x**2) for x in range(4)]
print(L)
# Prints [(0, 0), (1, 1), (2, 4), (3, 9)]
```

# List Comprehensions with IF

```
[expression for var in iterable if_clause]
```

```
# Filter list to exclude negative numbers
vec = [-4, -2, 0, 2, 4]
L = [x for x in vec if x >= 0]
print(L)
# Prints [0, 2, 4]
```

```
vec = [-4, -2, 0, 2, 4]
L = []
for x in vec:
    if x >= 0:
        L.append(x)
print(L)
# Prints [0, 2, 4]
```

# Nested List Comprehension

```
[expression for var in iterable]
```

```
[expression for var in iterable for var in iterable]
```

```python
# With list comprehension
vector = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
L = [number for list in vector for number in list]
print(L)
# Prints [1, 2, 3, 4, 5, 6, 7, 8, 9]

# equivalent to the following plain, old nested loop:
vector = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
L = []
for list in vector:
    for number in list:
        L.append(number)
print(L)
# Prints [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

# Dictionary Comprehension

```
D = {}
for x in range(5):
    D[x] = x**2

print(D)
# Prints {0: 0, 1: 1, 2: 4, 3: 9, 4: 16}
```

```
D = {x: x**2 for x in range(5)}
print(D)
# Prints {0: 0, 1: 1, 2: 4, 3: 9, 4: 16}
```

```
{key:value for var in iterable}
```

| *key: value* | *var* | *Iterable* |
|---|---|---|
| *Key & value can be any expression & evaluated once for each item in iterable* | *It takes items from an iterable one by one* | *It's a collection of objects (like a list, tuple etc.)* |

```
D = {0: 'A', 1: 'B', 2: 'C', 3: 'D', 4: 'E', 5: 'F'}
removeKeys = [0, 2, 5]

X = {k: D[k] for k in D.keys() - removeKeys}

print(X)
# Prints {1: 'B', 3: 'D', 4: 'E'}
```