# 1. Python - a tool, not a reptile

Python is a widely-used, interpreted, object-oriented, and high-level programming language with dynamic semantics, used for general-purpose programming.

And while you may know the python as a large snake, the name of the Python programming language comes from an old BBC television comedy sketch series called Monty Python's Flying Circus. At the height of its success, the Monty Python team were performing their sketches to live audiences across the world, including at the Hollywood Bowl. Since Monty Python is considered one of the two fundamental nutrients to a programmer (the other being pizza), Python's creator named the language in honor of the TV show.

Python was created by Guido van Rossum in the late 1980s.

# 2. Tipe Data, Variable dan Operasi Dasar

## 2.1. Fungsi print()

1. Fungsi print() adalah fungsi bawaan pada Python, yang berfungsi untuk mencetak pesan tertentu di screen/consol window.
2. Beberapa fungsi bawaan (bukan user-defined functions) sudah tersedia di Python tanpa harus diimport. Python 3.7.1 memiliki 69 fungsi bawaan (https://docs.python.org/3/library/functions.html).

In [ ]:

```
print("Hello, kamu !")
```

## 2.2. Tipe Data

## 2.2.1. Integers vs Floats

In [ ]:

```
print(1, "mempunyai tipe", type(1))
print(1.0, "mempunyai tipe", type(1.0))
```

#### Tambahan Penulisan Integer

Python versi 3 membolehkan pemisahan digit integer dengan underscore "_" agar mudah dibaca

In [ ]:

```
print(10_000_000)
```

#### Tambahan Penulisan Float

Float dipisahkan dengan .(titik) bukan ,(koma) cara penulisan float bisa 3 cara

- 4.0
- .4 (terbaca nol koma 4)
- 1. (terbaca 4 koma nol)

> 3e08 berarti $3x10^{8}$
> 3e-08 berarti $3x10^{-8}$

In [ ]:

```
print(0.000000003)
print(3e-8)
```

## 2.2.2. String

In [ ]:

```
print("I'm Monty Python")
print("Jum'at")
print('"Saya berpendapat python itu mudah", kata Monty')
print("1.5")
```

## 2.2.3. Booelan

Logika benar salah, 1 bernilai benar (True) dan 0 bernilai salah (False).

In [ ]:

```
print(True)
print(False)
print(3>0)
print(0>3)
```

In [ ]:

```
# Test
print((3>0) + (-5<-6) - 1)
```

# 2.3. Operator Aritmatika

| Operator | Arti | Contoh |
|---|---|---|
| + | Penambahan | x + y |
| - | Pengurangan | x - y |
| * | Perkalian | x * y |
| / | Pembagian | x / y |
| % | Modulo - Sisa Pembagian | x % y (sisa pembagian of x/y) |
| // | Pembagian dan dibulatkan ke bawah | x // y |
| ** | Eksponen - Pangkat | x**y (x pangkat y) |

In [ ]:

```
#eksponen/pangkat
print(2 ** 3)

#pembagian
print(6 / 3)

print(6 // 4)

#modulo/sisa bagi
print(14 % 4)
```

### Urutan Operator

| Prioritas | Operator | ket |
|---|---|---|
| 1 | +, - | unary |
| 2 | ** | |
| 3 | *, /, % | |
| 4 | +, - | binary |

In [ ]:

```
print((5 * ((25 % 13) + 100) / (2 * 13)) // 2)
```

# 2.4. Variabel

- A variable is a named location reserved to store values in the memory. A variable is created or initialized automatically when you assign a value to it for the first time.


- Each variable must have a unique name - an **identifier**. A legal identifier name must be a non-empty sequence of characters, must begin with the underscore(_), or a letter, and it cannot be a Python keyword. The first character may be followed by underscores, letters, and digits. Identifiers in Python are case-sensitive.
- Python is a **dynamically-typed** language, which means you don't need to declare variables in it. (2.1.4.3) To assign values to variables, you can use a simple assignment operator in the form of the equal (=) sign, i.e., var = 1.
- You can also use compound assignment operators (shortcut operators) to modify values assigned to variables, e.g., var += 1, or var /= 5 * 2

In [ ]:

```
var1 = 5
var2 = "Saya"
```

In [ ]:

```

```

## Shortcut operators

```
 variable = variable op expression
variable op= expression
```

```
 sheep = sheep + 1
sheep += 1
```

```
 x = x * 2
x *= 2
```

In [ ]:

```
var  = 5
print(var)

var +=5
print(var)
```

In [ ]:

```

```

## 2.5. Comment

1. Comments can be used to leave additional information in code. They are omitted at runtime. The information left in source code is addressed to human readers. In Python, a comment is a piece of text that begins with #. The comment extends to the end of line.
2. If you want to place a comment that spans several lines, in spite of placing # in front of them all, you can add """ before and after the group of line.

In [ ]:

```
# Ini comment

"""
Ini juga comment
yang lebih dari satu baris
"""
```

In [ ]:

```

```

## 2.6. Fungsi input()

1. The print() function sends data to the console, while the input() function gets data from the console.
2. The input() function comes with an optional parameter: the prompt string. It allows you to write a message before the user input, e.g.:
3. When the input() function is called, the program's flow is stopped, the prompt symbol keeps blinking (it prompts the user to take action when the console is switched to input mode) until the user has entered an input and/or pressed the Enter key.

In [ ]:

```
name = input("Enter your name: ")
print("Hello, " + name + ". Nice to meet you!")
```

In [ ]:

```

```

# 3. Operator Kondisi, Perulangan (Looping), List dan Operasinya, Bitwise, Boolean

## 3.1. Operator Relasional

Operator relasional adalah operator yang digunakan untuk membandingkan dua buah nilai. Hasil yang akan diperoleh dari operasi perbandingan ini adalah logika (bertipe bool), yaitu True (benar) atau False (salah).

| Operator | Jenis Operasi | Contoh |
|---|---|---|
| > | Lebih besar | (5 > 2) = 1 |
| < | Lebih kecil | (5 < 2) = 0 |
| >= | Lebih besar atau sama dengan | (5 >= 5) = 1 |
| <= | Lebih kecil atau sama dengan | (5 <= 2) = 0 |
| == | Sama dengan | (5 == 2) = 0 |
| != | Tidak sama dengan | (5 != 2) = 1 |

In [ ]:

```
a = int(input("masukkan a = "))
b = int(input("masukkan b = "))
c = int(input("masukkan c = "))
d = int(input("masukkan d = "))

print("a==b",a==b)
print("a==c",a==c)
print("a!=b",a!=b)

print("a>d",a>d)
print("b<d",b<d)
print("b<=d",b<=d)
print("b>=d",b>=d)
```

## 3.2. Operator Aritmatika

Operator aritmatika adalah operator yang digunakan untuk melakukan perhitungan terhadap data numerik. Berikut merupakan operator aritmatika yang digunakan dalam python :

| Operator | Keterangan | Contoh |
|---|---|---|
| + | Penjumlahan | 2+2 |
| – | Pengurangan | 5-1 |
| * | Perkalian | 4*2 |
| / | Pembagian | 6/2 |
| % | Sisa pembagian (Modulus) | 9%3 |
| ** | Perpangkatan | 7**2 |

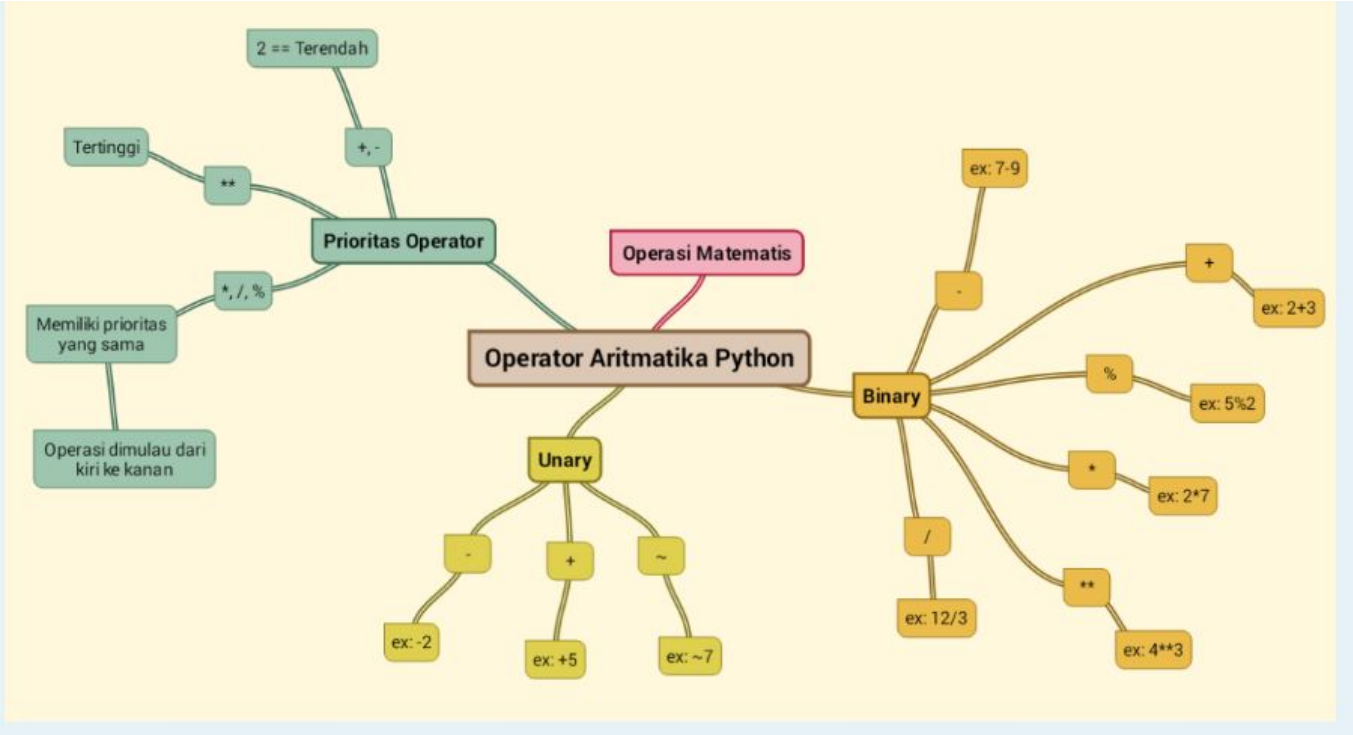
</b> Prioritas operator </b> Setiap operator aritmatika memiliki prioritas yang berbeda. Sebagai contoh :

</b> 3+57 *</b> Seperti yang diketahui, operator perkalian ()* memiliki prioritas yang lebih tinggi dibandingkan dengan operator penjumlahan (+) sehingga operasi dapat dituliskan sebagai berikut (3+(5*7)).

Untuk lebih jelasnya, tabel berikut memperlihatkan urutan prioritas dari operator aritmatika :

| Operator | Prioritas | Keterangan |
|---|---|---|
| ** | Tertinggi | |
| *, /, % | (Ketiganya memiliki prioritas yang sama) | Operasi dimulai dari kiri ke kanan |
| +, – | Terendah(keduanya memiliki prioritas yang sama) | Pengoperasian dari kiri ke kanan |

Berikut merupakan mindmap dari operasi aritmatika :




In [ ]:

```
x = 5 + 2 * 3 / (2 - 3) ** 2
print(x)
```

In [ ]:

```
x = 10
y = 3

print("x=",x)
print("y=",y)

print("x+y=", (x+y))
print("x-y=", (x-y))
print("x*y=", (x*y))
print("x/y=", (x/y))
print("x//y=", (x//y))
print("x%y=", (x%y))
print("x**y=", (x**y))
```

# 3.3. Struktur Pemilihan

Berbeda dengan bahasa pemrograman lainnya, Python hanya memiliki satu statement untuk melakukan proses pemilihan, yaitu dengan menggunakan if. Untuk mempermudah pembahasan tentang statement if, pembahasan akan dibagi menjadi tiga kelompok :

- Statement if untuk satu kasus
- Statement if untuk dua kasus
- Statement if untuk tiga kasus

## 3.3.1. Statement if untuk satu kasus

Bentuk umum penggunaan perintah if untuk satu kasus dalam Python adalah :

```
if kondisi:
    statement1
    statement2
    ...
```

Pada bentuk diatas, statement1, statement2 dan seterusnya hanya akan dieksekusi jika kondisi yang didefinisikan dalam perintah If terpenuhi (bernilai True). Jika kondisi bernilai False, statement tersebut akan diabaikan oleh program.

Perhatikan contoh berikut :

In [ ]:

```
a = 6
b = 6

if a==b:
    print("terkesekusi")
    print("Tes")

print("Halo")
```

### 3.3.2. Statement if untuk dua kasus

Untuk pemilihan yang mengandung dua kasus, bentuk umum yang digunakan adalah :

```
if kondisi :
    statement1
    statement2
    ...
else
    statement_alternatif1
    statement_alternatif2
    ...
```

Pada bentuk ini, jika kondisi bernilai True makan statement selanjutnya yang akan dieksekusi oleh program adalah statement1, stetement 2 dan seterusnya. Tapi jika kondisi bernilai False, maka program akan mengeksekusi statement yang berada pada bagian else, yaitu statement_alternatif1, statement_alternatif2 dan seterusnya.

Perhatikan contoh dibawah :

In [ ]:

```
a = 5
b = 5

if a==b:
    print("a dan b bernilai sama")
else:
    print("a dan b bernilai berbeda")
    print("Halo")
```

In [ ]:

```
# read two numbers
number1 = int(input("Enter the first number: "))
number2 = int(input("Enter the second number: "))

# choose the larger number
if number1 > number2:
    largerNumber = number1
else:
    largerNumber = number2

# print the result
print("The larger number is:", largerNumber)
```

### 3.3.3. Statement if untuk Tiga kasus atau lebih

Struktur pemilihan jenis ini merupakan bentuk yang paling kompleks jika dibandingkan dengan kedua jenis yang sudah di bahas diatas. Dalam struktur ini, ada beberapa kondisi yang harus diperiksa oleh program. Bentuk umum penggunaan perintah if untuk tiga kasus atau lebih adalah :

```
if kondisi1:
    statement1a
    statement1b
    ...
elif kondisi2:
    statement2a
    statement2b
    ...
else:
    statement_alternatif1
    statement_alternatif2
    ...
```

Perhatikan contoh dibawah ini :

In [ ]:

```
a = 80

if a > 80:
    print("Lulus")
elif a > 60:
    print("Mengulang Sebagian")
elif a > 40:
    print("Mengulang Keseluruhan")
else:
    print("Tidak Lulus")
```

# 3.4. Struktur perulangan

Python menyediakan dua statement untuk melakukan proses perulangan yaitu for dan while. Diantara kedua statement ini, secara umum for lebih banyak digunakan daripada while.

## 3.4.1. Statement while

Sama seperti kebanyakan bahasa pemrograman lainnya, Python juga mendukung statement while untuk melakukan perulangan satu atau sekelompok statement. Bentuk umum dari perulangan while adalah sebagai berikut :

```
while kondisi:
    statement1
    statement2
    statement1
    :
    :
    statementn
```

Pada bentuk perulangan diatas, statement1, statement2 dan seterusnya merupakan sekelompok statement yang akan diulang. Statement tersebut akan terus dieksekusi/ diulang selama kondisi bernilai True. Maka diperlukan sebuah statement yang akan bernilai False, agar proses dapat berhenti. Perhatikan syntax berikut :

In [ ]:

```
i = 1
while i<10:
    print(i**2)
    i = i+2
```

### 3.4.2. Statement for

Selain menggunakan statement while, kita juga dapat melakukan perulangan dengan statement for. Perintah for biasanya digunakan untuk mengambil atau menelusuri data (item) yang terdapat pada tipe-tipe koleksi seperti string, list, tuple, dictionary dan set. Dalam pemrograman, proses penelusuran data dalam sebuah koleksi disebut iterasi. Selain itu, for juga dapat melakukan perulangan normal (sama seperti while), yaitu dengan menggunakan fungsi range(). Dengan menggunakan range(), kita akan dapat dengan mudah melakukan perulangan dari indeks awal ke indeks tertentu.

Bentuk perintah for dapat dituliskan sebagai berikut :

1. Statement for untuk perintah koleksi

```
for indeks tipe koleksi:
 statement1
 statement2
 ...
```

2. Statement for untuk rentang nilai tertentu

```
for indeks in range (nilai_awal, nilai_akhir, step):
 statement1
 statement2
 ...
```

Perhatikan contoh-contoh dibawah ini :

In [ ]:

```
for i in range(5):
    print(i)
```

In [ ]:

```
for i in range(2, 8):
    print(i)
```

In [ ]:

```
for i in range(2, 18, 3):
    print(i)
```

**For - else**

In [ ]:

```
for i in range(5):
    print(i)
else:
    print("masuk else")
```

# 3.5. Statement Loncat

Statement loncat merupakan perintah yang digunakan untuk memindahkan eksekusi program dari satu bagian tertentu ke bagian lain. Python menyediakan tiga statement yaitu : break, continue dan return.

## 3.5.1. Statement break

Statement break digunakan untuk menghentikan secara paksa proses perulangan, meskipun kondisi perulangan sebenarnya masih bernilai True. Perhatikan contoh-contoh berikut ini :

In [ ]:

```
for i in range (11):
    print(i,end=' ')
    if i == 7:
        print("Masih dalam for ")
        break
```

## 3.5.2. Statement continue

Statement continue berguna untuk memaksa pengulangan agar memroses indeks selanjutnya meskipun statement didalam blok pengulangan sebenarna belum semua di eksekusi. Dengan kata lain, statement yang ada dibawah statement continue akan diabaikan (tidak dieksekusi)

In [ ]:

```
for i in range (1,11):
    if i % 2 ==0:
        continue
    print(i, end='\t')
```

# 3.6. Operasi Logika

Operasi Logika `and`

| Argument A | Argument B | A and B |
|---|---|---|
| False | False | False |
| False | True | False |
| True | False | False |
| True | True | True |

Operasi Logika `or`

| Argument A | Argument B | A or B |
|---|---|---|
| False | False | False |
| False | True | True |
| True | False | True |
| True | True | True |

Operasi Logika `not`

| Argument | not Argument |
|---|---|
| False | True |
| True | False |

In [ ]:

```
x = 1
y = 0

z = ((x == y) and (x == y)) or not(x == y)
print(not(z))
```

# 3.7. List dan Array

Tipe list dalam python sebenernya mirip dengan array normal pada bahasa pemrograman lain. List juga dapat menampung beberapa tipe data sekaligus.

Bentuk umum untuk membuat list dalam python adalah :

```
nama_list = [nilai1, nilai2, nilai3,...]
```

Perhatikan contoh dibawah ini :

In [ ]:

```
var1 = 10
var2 = 5
var3 = 7
var4 = 2
var5 = 1

# VERSUS #

var = [10, 5, 7, 2, 1]
print(var)
```

In [ ]:

```

```

In [ ]:

```
#bisa menampung berbagai tipe data
var = ["Digital", 90 , 1000.50 , "Talent"]
print(var)
```

In [ ]:

```

```

### 3.7.1. Indexing

Dalam list, elemen tidak diindeks berdasarkan key tertentu yang namanya bisa didefinisikan sendiri melainkan berdasarkan indeks bilangan yang diawali dengan nilai nol(0). Perhatikan contoh berikut :

In [ ]:

```
numbers = [10, 5, 7, 2, 1]
print(numbers)
print(numbers[2])
print(numbers[0])
```

In [ ]:

```
numbers = [10, 5, 7, 2, 1]
print("Original list content:", numbers) # printing original list content

numbers[0] = 111
print("\nPrevious list content:", numbers) # printing previous list content

numbers[1] = numbers[4] # copying value of the fifth element to the second
print("Previous list content:", numbers) # printing previous list content

print("\nList length:", len(numbers)) # printing the list's length
```

Indeks dapat juga bernilai negatif yaitu menghitung indeks mulai dari kanan. Perhatikan contoh berikut :

In [ ]:

```
lst = [100,200,300,400,500]
print(lst[-1])
print(lst[-2])
print(lst[-5])
```

In [ ]:

```

```

## 3.7.2. Slicing List

In [ ]:

```
lst = [4, 1, 6, 9, 10, 8]
print(lst[0:2])
print(lst[1:3])
print(lst[-3:])
print(lst[:-3])
print(lst[:])
```

In [ ]:

```

```

**Menambah Elemen ke dalam list**

Terdapat tiga cara untuk menambahkan elemen baru kedalam sebuah list yaitu :

- Menggunakan metode append() : Metode ini digunakan untuk menambah elemen tunggal dibagian akhir list. Perhatikan contoh dibawah :

In [ ]:

```
buah = ["durian", "mangga", "apel"]
buah.append("jeruk")
print(buah)
```

In [ ]:

```
myList = [] # creating an empty list

for i in range(5):
    myList.append(i + 1)

print(myList)
```

- Menggunakan metode insert() : Metode ini digunakan untuk menambah elemen tunggal di indeks / posisi tertentu. Perhatikan contoh dibawah :

In [ ]:

```
buah = ["durian", "mangga", "apel"]
buah.insert(1,"kiwi")
print (buah)
```

- Menggunakan metode extend(). Metode ini digunakan untuk menyambung atau menggabungkan suatu list dengan list yang lain. Perhatikan contoh berikut :

In [ ]:

```
buah = ["durian", "mangga", "apel"]
buah.extend(["jeruk","melon"])
print(buah)
```

In [ ]:

```

```

**Menghapus Elemen kedalam list**

Menghapus elemen dapat dilakukan dengan :

```
nama_list.remove(nilai)
```

atau dengan memberikan indeksnya menggunakan instruksi delete :

```
del nama_list[indeks]
```

In [ ]:

```
numbers = [5,4,3,2,1]

del numbers[1] # removing the second element from the list
print("New list's length:", len(numbers)) # printing new list length
print("\nNew list content:", numbers) # printing current list content
```

**1. List Dua Dimensi (Array)**

List dapat berisi berbagai tipe data seperti strings, boolean maupun list lainnya. Kita sering sekali menemukan bentuk array seperti ini dikehidupan kita, salah satu contohnya adalah papan catur. Papan catur terdiri dari baris dan kolom, terdapat 8 baris dan juga 8 kolom. Setiap kolom ditandai oleh huruf A sampai dengan H, dan setiap baris ditandai oleh 1 sampai dengan 8.

Perhatikan kode berikut :

In [ ]:

```
k = [[3,5,1],[6,1,8],[9,10,4]]
print(k[1])
print(k[1][0])
print(k[-1][-2:])
```

**2. List Multidimensi (Array)**

Untuk mencari sebuah elemen pada list dua dimensi adalah dengan menggunakan dua koordinat yaitu nomer baris dan juga nomer kolom. Bayangkan ketika anda akan mengembangkan sebuah program untuk perekam cuaca otomatis. Alat ini akan mampu merekam suhu udara tiap jam sampai dengan satu bulan. Maka total data yang akan direkam adalah 24 jam x 31 hari = 744 data.

Pertama-tama, kita harus memutuskan tipe data yang akan tepat untuk aplikasi ini. Pada kasus ini, tipe float adalah yang paling sesuai karena suhu biasanya diukur sampai dengan akurasi satu dibelakang koma (0.1). Selanjutnya, kita akan menentukan bahwa baris akan merepresentasikan jam (sehingga ada 24 baris) dan setiap baris akan direpresentasikan untuk satu hari (sehingga kita membutuhkan lagi 31 baris)

In [ ]:

```
temps = [[0.0 for h in range(24)] for d in range(31)]
print (temps)
```

**3. Array tiga dimensi**

Python tidak memiliki batasan dalam list-in-list.

# 4. Function Tuple Dictionaries and Data Processing

## 4.1. Fungsi

Fungsi merupakan blok coding yang mengerjakan tugas tertentu.

Sintaks Fungsi :

```
def functionName(parameters):
    functionBody
    return result
```

In [ ]:

```
def luas_persegipanjang(panjang,lebar):
    luas = panjang*lebar
    print(luas)
```

In [ ]:

```
luas_persegipanjang(5,4)
```

In [ ]:

```
def luas_persegipanjang(panjang,lebar):
    luas = panjang*lebar
    return luas
```

In [ ]:

```
x = luas_persegipanjang(5,4)
print(x)
```

# 4.2. Tuples and Dictionaries

List datanya bisa dirubah selama eksekusi program, sifatnya bernama mutable Sebaliknya tuple tidak bisa, sifatnya dinamakan immutable. Persamaannya keduanya sama-sama bisa menyimpan banyak nilai sekaligus dalam satu tipe data (sequence type)

## 4.2.1. Tuples

Tuples are ordered and unchangeable (immutable) collections of data. They can be thought of as immutable lists. They are written in round brackets:

In [ ]:

```
myTuple = (1, 2, True, "a string", (3, 4), [5, 6], None)
print(myTuple)

myList = [1, 2, True, "a string", (3, 4), [5, 6], None]
print(myList)
```

In [ ]:

```
tuple1 = (1, 2, 4, 8)
tuple2 = 1., .5, .25, .125

print(tuple1)
print(tuple2)
```

## 4.2.2. Dictionaries

In Python's world, the word you look for is named a key. The word you get from the dictionary is called a value.

Dictionaries are unordered*, changeable (mutable), and indexed collections of data. (*In Python 3.6x dictionaries have become ordered by default.

This means that a dictionary is a set of key-value pairs. Note:

- each key must be unique - it's not possible to have more than one key of the same value;
- a key may be data of any type: it may be a number (integer or float), or even a string;
- a dictionary is not a list - a list contains a set of numbered values, while a dictionary holds pairs of values;
- the len() function works for dictionaries, too - it returns the numbers of key-value elements in the dictionary;
- a dictionary is a one-way tool - if you have an English-French dictionary, you can look for French equivalents of English terms, but not vice versa.

In [ ]:

```
phoneNumbers = {'boss' : 5551234567, 'Suzy' : 22657854310}

print(phoneNumbers)
```

In [ ]:

```
phoneNumbers['boss']
```

In [ ]:

```
for key in phoneNumbers.keys():
    print(key, "->", phoneNumbers[key])
```

In [ ]:

```
for val in phoneNumbers.values():
    print(val)
```

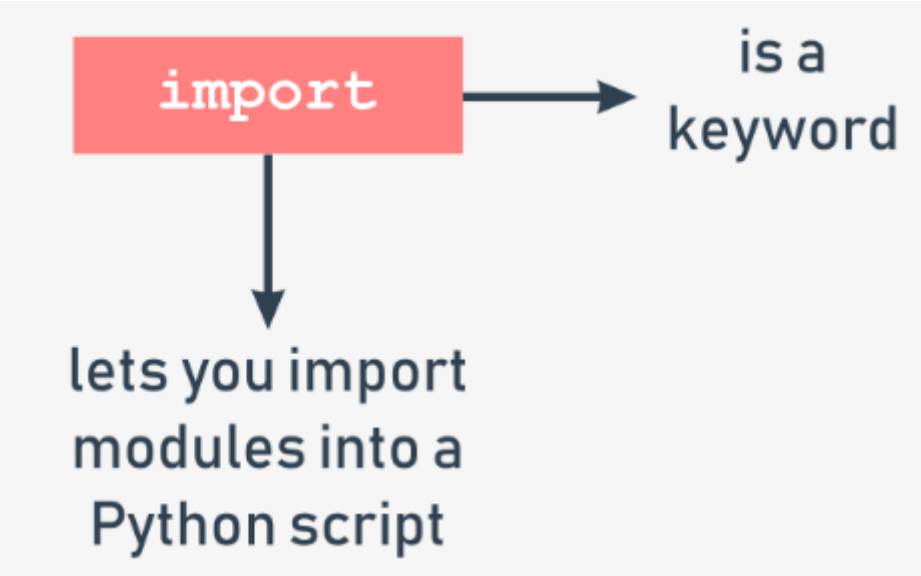
# 5. Modules dan Packages

## 5.1. Import Modul

Proes mengimport sebuah modul dapat dilakukan dengan instruksi yaitu `import`

Adapun sintaksnya berisi :

- `import` keyword;
- `nama` modul yang akan diimport.




In [ ]:

```
import numpy
import math
import scipy
```

In [ ]:

```
import numpy, math, scipy
```

In [ ]:

```
dir(numpy)
```

In [ ]:

```
numpy.array?
```

In [ ]:

```
print(math.pi)
print(math.e)
print(numpy.pi)
print(scipy.pi)
```

## 5.2. Import Sebagian/Semua Fungsi dari Modul

In [ ]:

```
from math import pi,e


print(pi)
print(e)
```

In [ ]:

```
from math import *

print(tan(0))
```

Nama dari entitas digantikan dengan asterisk tunggal `*`

`*` merupakan instruksi untuk meng-import semua entitas yang ada

## 5.3. Import Modul dengan Aliasing

Untuk nama file yang akan di `import` kan dapat dilakukan proses `aliasing`

`Aliasing` menyebabkan `modul` diidentifikasi dengan `nama yang berbeda` dari `aslinya`

`import` module `as` alias

`as` merupakan kata kunci untuk melakukan `aliasing`

Jika kita ingin merename `math` , dengan `m` dapat dilakukan dengan cara sebagai berikut.

In [ ]:

```
import math as m

print(m.pi)
```

# 6. Modul Pandas

Sebagai salah satu modul di Python, banyak hal yang dapat dilakukan oleh Pandas.

Pandas adalah rumah, kerangka bagi data, yang akan mengubah tampilan data menjadi dataframe.

Fitur-fitur pada Pandas :

- Kalkulasi Statistik
- Cleaning Data
- Visualisasi Data (dengan Matplotlib, seaborn, etc.)
- Menyimpan Data dalam berbagai bentuk

In [ ]:

```
import pandas as pd
```

In [ ]:

```
df = pd.read_csv('Automobile_data.csv')
```

In [ ]:

```
df.info()
```

In [ ]:

```
df.head()
```

In [ ]:

```
df['aspiration'].value_counts()
```

In [ ]:

```
df['aspiration'].unique()
```

## 6.1. Data Aggregation and Group Operations

### 6.1.1. Filtering Data

In [ ]:

```
df[df['symboling']==2]
```

In [ ]:

```
```

## 6.1.2. Sorting Data

In [ ]:

```
df.sort_values(by=['symboling'])
```

In [ ]:

```
```

## 6.1.3. Grouping Data

In [ ]:

```
df.groupby(['symboling','aspiration']).mean()
```

In [ ]:

```
```

## 6.2. Exploratory Data Analysis (EDA) with Real Case

1. Statistik Deskriptif
2. Checking Outlier from Boxplot
3. Checking Distribution from Histogram
4. Correlation from Heatmap

In [ ]:

```
df.describe()
```

In [ ]:

```
```