Python Variables & Data Types

Apa itu variable?

Variabel dalam program komputer digunakan untuk menyimpan, memanipulasi, dan memberi label data. Variabel adalah kontainer berlabel yang memiliki nama tertentu. Di dalam variabel ini ada data yang dapat berupa teks, angka, atau bahkan hal-hal yang lebih kompleks. Ketika menyimpan data dalam variabel, kita dapat memakainya kembali kemudian dengan cukup menyebutkan nama variabel.

```
if __name__ == "__main__":
    a = 42
    b = 23
    c = a * b
    print("c:", c)
    a = 89
    print("a:", a)
    print("c:", c)
```

Output

```
c: 966
a: 89
c: 966
```

Penamaan Variable

Penamaan variable pada python mengikuti standar PEP8 (Python Enterprise Proposal). PEP8 diciptakan oleh Kenneth Reitz yang memudahkan konvensi pengkodean untuk kode Python.

snake_case

- Semuanya dalam huruf kecil
- Tidak dimulai dengan karakter khusus seperti misalnya &(ampersand), $ (dolar)
- Nama variabel hanya dapat berisi karakter berikut A-Z, a-z, 0-9, dan _
- Jika nama berisi beberapa kata, itu harus dipisahkan oleh garis bawah (_) misalnya first_name
- Hindari variabel satu karakter misalnya a, b, c, dll
- Nama variabel tidak dapat dimulai dengan angka.

- Nama variabel peka huruf besar/kecil, yang berarti kucing, Kucing, dan CAT adalah variabel yang berbeda.

Misalnya kita memiliki kucing, dan kucing pasti memiliki karakteristik tertentu, jika kita ingin menyimpan informasi tentang kucing tersebut, kode dan nama variabel bisa terlihat seperti ini:

```
if __name__ == "__main__":
    species = "cat"
    breed = "british shorthair"
    color = "chocolate"
    name = "Mr. Fluffers"
    sex = "male"
    age_years = 7
    weight_kg = 6.1
    favorite_toy = "plush mouse"
    favorite_food = "tuna"
    sleeping = True
```

Dalam contoh kode ini, Anda dapat melihat bahwa semua nama variabel mewakili apa yang disimpan dalam variabel tersebut. Kedua, Anda melihat bahwa berbagai jenis data disimpan. Pada baris 7 usia dalam tahun disimpan dalam age_years sebagai bilangan bulat. Namun, berat dalam kg dalam weight_kg adalah angka desimal dan pada baris 11 sleeping mewakili keadaan yang True atau False. Semua variabel lainnya adalah teks.

```
print(name, "is a", sex, breed, species)
print(name, "has a", color, "color, is", age_years, "years old, and
weights", weight_kg, "kgs")
print(name, "loves to eat", favorite_food, "and enjoys playing with a",
favorite_toy)
print("Is", name, "sleeping?",
```

Comment & Docstring

Comment adalah pernyataan dalam kode yang tidak dijalankan. Jika kita ingin memasukkan komentar yang pas dalam satu baris dengan Python, mulailah baris dengan # (disebut tanda

pound atau hash). Misalnya, menambahkan komentar di depan pernyataan cetak yang mengklarifikasi apa yang sedang dicetak.

One-line comment

```
# prints text block mendeskripsikan hewan kesayangan anda
print(name, "is a", sex, breed, species)
print(name, "has a", color, "color, is", age_years, "years old, and
weights", weight_kg, "kgs")
print(name, "loves to eat", favorite_food, "and enjoys playing with a",
favorite_toy)
print("Is", name, "sleeping?", sleeping
```

Multi-line comment

```
# prints text block mendeskripsikan
# hewan kesayangan anda
# name = "garfield"
species = "cat"
breed = "british shorthair"
color = "chocolate"
```

Docstrings vs. Commenting

Comment adalah deskripsi yang membantu programmer lebih memahami maksud dan fungsionalitas program. Mereka benar-benar diabaikan oleh Python interpreter.

Python **docstrings** adalah string yang ditulis tepat setelah definisi fungsi, metode, class, atau module (seperti dalam Contoh 1). **Docstrings** digunakan untuk mendokumentasikan kode. Docstring juga dapat dibuat secara multiline

```
>>> def square(n):
      '''Takes in a number n, returns the square of n'''
      return n**2

>>> square.__doc__
'''Takes in a number n, returns the square of n'''
```

Data Types

Primitive Data Types

- **Integer,** int, adalah semua bilangan bulat: 1, -2304, 0, dll.
- **Float,** float, adalah semua angka desimal: 23.5, -0.52, 3525.252, dll.
- **String,** str, adalah semua huruf dan teks: "a", "Ini adalah teks. Dan kalimat lain.", dll.
- **Boolean,** bool, adalah True atau False.

Dalam bahasa pemrograman lain seperti C atau Java, jenis variabel harus dinyatakan secara eksplisit, dan variabel hanya dapat menyimpan data dari jenis tersebut. Namun, Python adalah bahasa pemrograman yang diketik secara dinamis di mana jenis variabel tidak harus dinyatakan secara eksplisit. Jenis variabel Python diputuskan ketika nilai ditetapkan ke variabel.

```
print(type(species))
print(type(age_years))
print(type(weight_kg))
print(type(sleeping))
```

Output

```
<class 'str'>
<class 'int'>
<class 'float'>
<class 'bool'>
```

String

String adalah urutan karakter. String dalam python dikelilingi oleh tanda kutip tunggal, atau tanda kutip ganda.

Karakter hanyalah simbol. Misalnya, alfabet memiliki 26 karakter.

Komputer tidak berurusan dengan karakter, mereka berurusan dengan angka (biner). Meskipun kita mungkin melihat karakter di layar, secara internal itu disimpan dan dimanipulasi sebagai kombinasi dari 0s dan 1s.

```
print("Hello world")
print('Hello world')
```

Assign string ke suatu variable

```
string_1 = "Hello world"
print(string_1)
```

Multi-line String

```
string_2 = """Lorem ipsum dolor sit amet,
consectetur adipiscing elit"""
print(a)
```

Mengakses element dalam String

```
string_1 = "Hello world"
print(string_1[1])
```

Looping dalam String

Kita dapat me-looping karakter dalam string, dengan for loop.

```
for x in "hello":
  print(x)
```

Panjang String

Untuk mendapatkan panjang string, gunakan fungsi len().

```
string_1 = "Hello world"
print(len(string_1))
```

Fungsi enumerate() mengembalikan objek enumerate. Ini berisi indeks dan nilai semua item dalam string sebagai pasangan. Ini bisa berguna untuk iterasi.

```
str = 'cold'

# enumerate()
list_enumerate = list(enumerate(str))
```

Output

```
list(enumerate(str) =  [(0, 'c'), (1, 'o'), (2, 'l'), (3, 'd')]
```

Check String

Untuk memeriksa apakah terdapat karakter tertentu dalam string, kita dapat menggunakan **in** atau **not in:**

```
string_3 = "Selamat malam semua"
print("malam" in string_3)
```

```
string_3 = "Selamat malam semua"
print("siang" not in string_3)
```

String Operations

Ada banyak operasi yang dapat dilakukan dengan string yang menjadikannya salah satu tipe data yang paling banyak digunakan di Python.

```
# Python String Operations
str1 = 'Hello'
str2 ='World!'

# using +
print('str1 + str2 = ', str1 + str2)

# using *
print('str1 * 3 =', str1 * 3)
```

Output

```
str1 + str2 =  HelloWorld!
str1 * 3 = HelloHelloHello
```

Python String Formatting

Escape Sequence

Ketika kita ingin mencetak kalimat seperti `He said, "What's there?"` kita tidak dapat menggunakan tanda kutip tunggal atau tanda kutip ganda. Ini akan menghasilkan SyntaxError karena teks itu sendiri berisi tanda kutip tunggal dan ganda.

```
>>> print("He said, "What's there?"")
...
SyntaxError: invalid syntax
>>> print('He said, "What's there?"')
```

```
...
SyntaxError: invalid syntax
```

Hal ini dapat dilakukan dengan menggunakan cara berikut:

```
# using triple quotes
print('''He said, "What's there?"''')

# escaping single quotes
print('He said, "What\'s there?"')

# escaping double quotes
print("He said, \"What's there?\"")
```

Output

```
He said, "What's there?"
He said, "What's there?"
He said, "What's there?"
```

format() method

format() sangat serbaguna dalam memformat string.

```
# Python string format() method

# default(implicit) order
default_order = "{}, {} and {}".format('Joko','Budi','Susi')
print('\n--- Default Order ---')
print(default_order)

# order using positional argument
positional_order = "{1}, {0} and {2}".format('Joko','Budi','Susi'))
print('\n--- Positional Order ---')
print(positional_order)

# order using keyword argument
keyword_order = "{s}, {b} and {j}".format(j=Joko,b='Budi',s='Susi')
print('\n--- Keyword Order ---')
print(keyword_order)
```

Common Python String Methods

Beberapa metode yang umum digunakan adalah lower(), upper(), join(), split(), find(), replace() dll

```
>>> "DigitalSkola".lower()
'digitalskola'
>>> "DigitalSkola".upper()
'DIGITALSKOLA'
>>> "Digital Skola".split()
['Digital', 'Skola']
>>> ' '.join(['Digital', 'Skola'])
'Digital Skola'
>>> 'Digital Skola'.find('Sko')
8
>>> 'Musim Hujan'.replace('Hujan','Panas')
'Musim Panas'
```

Numbers

Ada tiga jenis data numerik dalam Python:

- int
- float
- complex

Variabel tipe numerik dibuat saat kita menetapkan nilai:

```
x = 100    # int
y = 1.5    # float
z = 10j    # complex
```

Untuk memverifikasi jenis objek apa pun di Python, gunakan fungsi type()

Int

Int adalah bilangan bulat, positif atau negatif, tanpa desimal, dengan panjang tak terbatas.

```
x = 1
print(type(x))
```

Float

Float, atau "floating point number" adalah angka, positif atau negatif, yang berisi satu atau lebih desimal.

```
x = 1.11
print(type(x))
```

Float juga bisa berupa angka ilmiah dengan "e" untuk menunjukkan pangkat 10.

```
x = 5e3
y = 2E-4
z = -7.7e10

print(type(x))
print(type(y))
print(type(z))
```

Rounding

```
# ✅ round a float to 2 decimals

result_1 = round(4.5678, 2)
print(result_1)  # 👉 4.57

result_2 = round(4.1234, 2)
print(result_2)  # 👉 4.12

# ------------------------------------

# ✅ print a float rounded to 2 decimals

my_float = 4.5678

my_str_1 = f'{my_float:.2f}'
print(my_str_1)  # 👉 '4.57'
```

Complex

Bilangan kompleks ditulis dengan "j" sebagai bagian imajiner:

```
x = 3+5j
y = 5j
z = -5j

print(type(x))
print(type(y))
print(type(z))
```

Random Numbers

Python tidak memiliki fungsi random() untuk membuat angka acak, tetapi Python memiliki modul bawaan yang disebut random yang dapat digunakan untuk membuat angka acak:

```
import random

print(random.randrange(1, 10))

print(random.randrange(10, 20))

x = ['a', 'b', 'c', 'd', 'e']

# Get random choice
print(random.choice(x))

# Shuffle x
random.shuffle(x)

# Print the shuffled x
print(x)

# Print random element
print(random.random())
```

Python Mathematics

Python menawarkan modul seperti math melakukan operasi matematika seperti trigonometri, logaritma, probabilitas dan statistik, dll.

```
import math

print(math.pi)

print(math.cos(math.pi))

print(math.exp(10))

print(math.log10(1000))

print(math.sinh(1))

print(math.factorial(6))
```

Converting (casting) data types

Nilai dari berbagai jenis data juga dapat dikonversi menjadi yang lain atau casting. Casting int ke dalam float dengan Python hampir selalu terjadi secara implisit, jadi tidak perlu menulis kode

secara explicit.

```
a = 1
b = 2
c = a/b
type(a)
type(b)
type(c)
```

Output

```
<class 'int'>
<class 'int'>
<class 'float'>
```

Casting secara implisit juga berfungsi ketika menggabungkan boolean dan angka. Boolean True and False biasanya disingkat dengan 1 dan 0, di mana 1 mewakili True dan 0 adalah False. Karena ini, kita dapat memasukkan boolean dalam perhitungan:

```
t = True
f = False
i = 10
i_t = 10 * t
i_f = 10 * f
type(t)
type(f)
type(i)
type(i_t)
type(i_f)
```

Output

```
<class 'bool'>
<class 'bool'>
<class 'int'>
<class 'int'>
```

Kita bahkan dapat menggabungkan string dan int. Saat mengalikan variabel tipe str dengan int, hasilnya adalah string yang berulang.

```
l = "a"
x = 10
s = l * x
s
```

```
type(l)
type(x)
type(s)
```

Output

```
'aaaaaaaaaa'
<class 'str'>
<class 'int'>
<class 'str'>
```

Casting eksplisit yang tersedia untuk tipe data Python primitif cukup mudah diingat karena dinamai menurut tipe data:

- int(...) untuk casting variabel ke dalam bilangan bulat,
- float(...) untuk casting variabel ke dalam float,
- str(...) untuk casting variabel ke dalam string,
- bool(...) untuk casting variabel menjadi boolean.

Bilangan bulat, angka floating-point, dan boolean dapat di-casting ke dalam tipe data lain:

```
i = 10
float(i)
str(i)
bool(i)

f = 23.42
int(f)
str(f)
bool(f)

b = True
int(b)
float(b)
str(i)
```

Output

```
# i
10.0
'10'
True

# f
23
```

```
'23.42'
True

# b
1
1.0
'True'
```

```
e = ""
s = "This is a string"
bool(e)
bool(s)
```

Output

```
False
True
```

Boolean

Booleans mewakili salah satu dari dua nilai: True atau False.
Dalam pemrograman kita perlu tahu apakah suatu ekspresi itu Benar atau Salah.

Kita dapat mengevaluasi ekspresi apa pun dengan Python, dan mendapatkan salah satu dari dua jawaban, Benar atau Salah.

```
print(10 > 9)
print(10 == 9)
print(10 < 9)
```

Evaluasi Nilai dan Variabel

fungsi bool() memungkinkanuntuk mengevaluasi nilai apa pun, dan memberi True atau False sebagai outputnya,

```
print(bool("Hello"))
print(bool(15))
```

Hampir semua nilai dievaluasi ke True jika memiliki konten.

- String apa pun adalah True, kecuali string kosong.
- Angka apa pun adalah True, kecuali 0.
- Dict, tuple, set, dan list apa pun adalah True, kecuali yang kosong.

```
bool("abc")
bool(123)
bool(["apple", "banana"])
```

Faktanya, tidak banyak nilai yang dievaluasi ke False, kecuali nilai kosong, seperti (), [], {}, "", angka 0, dan nilai None. Dan tentu saja nilai yang dievaluasi False ke False.

```
bool(False)
bool(None)
bool(0)
bool("")
bool(())
bool([])
bool({})
```

Non-Primitive Data Type

List

List di Python adalah salah satu tipe data yang paling serbaguna yang memungkinkan kita untuk bekerja dengan banyak elemen sekaligus. Misalnya:

```
# a list of programming languages
['Python', 'C++', 'JavaScript']
```

List adalah koleksi yang dapat diurutkan dan diubah. Memungkinkan adanya duplikat.

Membuat Python List

Di Python, list dibuat dengan menempatkan elemen di dalam tanda kurung siku [], dipisahkan oleh koma (,).

```
# list of integers
my_list = [1, 2, 3]
```

List dapat memiliki sejumlah item dan mungkin dari berbagai jenis (bilangan bulat, float, string, dll.).

```
# empty list
my_list = []

# list with mixed data types
my_list = [1, "Hello", 3.4]
```

List juga dapat memiliki list lain sebagai item. Ini disebut nested list.

```
# nested list
my_list = ["mouse", [8, 4, 6], ['a']]
```

Mengakses Elemen dari List

Ada berbagai cara di mana kita dapat mengakses elemen-elemen dari list di Python.

List Index

Kita dapat menggunakan operator indeks [] untuk mengakses item dalam list. Dengan Python, indeks mulai dari 0. Jadi, daftar yang memiliki 5 elemen akan memiliki indeks dari 0 hingga 4.

Mencoba mengakses indeks selain ini akan meningkatkan IndexError. Indeks harus berupa bilangan bulat. Kita tidak bisa menggunakan float atau jenis lainnya, ini akan menghasilkan TypeError.

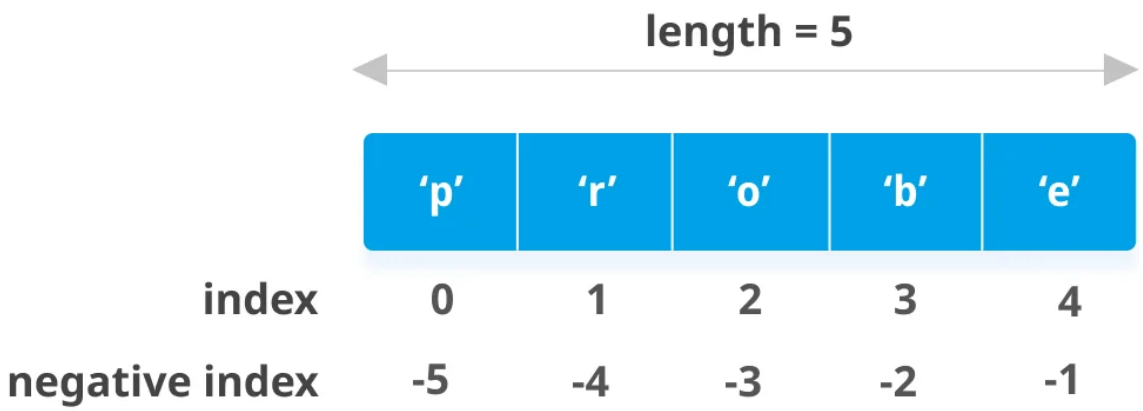
```
my_list = ['j', 'a', 'k', 'a', 'r', 't', 'a']

# first item
print(my_list[0])  # j

# third item
print(my_list[2])  # k

# fifth item
```

```
print(my_list[4])  # r

# Nested List
n_list = ["Happy", [2, 0, 1, 5]]

# Nested indexing
print(n_list[0][1])

print(n_list[1][3])

# Error! Only integer can be used for indexing
print(my_list[4.0])
```

Negative Indexing

Python memungkinkan pengindeksan negatif untuk urutannya. Indeks -1 mengacu pada item terakhir, -2 ke item terakhir kedua dan seterusnya.
Negative indexing in lists

```
my_list = ['p','r','o','b','e']

# last item
print(my_list[-1])

# fifth last item
print(my_list[-5])
```



List Slicing

Kita dapat mengakses berbagai item dalam list dengan menggunakan operator slicing **:** .

```
# List slicing in Python

my_list = ['j', 'a', 'k', 'a', 'r', 't', 'a']

# elements from index 2 to index 4
print(my_list[2:5])

# elements from index 5 to end
print(my_list[5:])

# elements beginning to end
print(my_list[:])
```

Saat kita melakukan slicing list, indeks awal akan termasuk tetapi indeks akhir tidak akan termasuk. Misalnya, my_list[2: 5] mengembalikan list dengan elemen pada indeks 2, 3 dan 4, tetapi tidak 5.

```
>>> list_1 = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 16, 17, 18,
19]
```

Dan jika kita menginginkan list dari awal hingga indeks tertentu, jangan masukkan indeks pertama:

```
>>> list_1[:7]
[0, 1, 2, 3, 4, 5, 6]
```

Dimungkinkan untuk hanya mendapatkan setiap elemen ke-n dari list dengan pengirisan list dengan menambahkan titik dua lain : setelah indeks kedua yang menunjukkan banyaknya langkah. Jika kita hanya ingin setiap elemen ke-3 dari sub list list_1[5:18] gunakan yang berikut:

```
>>> i[5:18:3]
[5, 8, 11, 14, 18]

>>> i[::3]
[0, 3, 6, 9, 12, 16, 19]
```

Nested List

List juga dapat berisi list lain:

```
>>> list_1 = [3, 4, 5]
```

```
>>> list_2 = [6, 7, 8]
>>> all_list = [list_2, list_2]
>>> all_list
[[3, 4, 5], [6, 7, 8]]


>>> l[1][0]
6
```

Menambahkan atau Mengubah List

List bersifat mutable atau dapat diubah, tidak seperti String atau Tuple yg bersifat immutable. Kita dapat menggunakan operator = untuk mengubah item atau berbagai item.

```
# Correcting mistake values in a list
odd = [2, 4, 6, 8]

# change the 1st item
odd[0] = 1

print(odd)

# change 2nd to 4th items
odd[1:4] = [3, 5, 7]

print(odd)
```

Kita dapat menambahkan satu item ke list menggunakan metode append() atau menambahkan beberapa item menggunakan metode extend().

```
# Appending and Extending lists in Python
odd = [1, 3, 5]
odd.append(7)

print(odd)

odd.extend([9, 11, 13])
print(odd)
```

Kita juga dapat menggunakan operator + untuk menggabungkan dua list. Ini juga disebut penggabungan.

Operator * mengulangi list untuk jumlah yang diberikan kali.

```
# Concatenating and repeating lists
odd = [1, 3, 5]

print(odd + [9, 7, 5])

print(["re"] * 3)
```

Selanjutnya, kita dapat memasukkan satu item di lokasi yang diinginkan dengan menggunakan metode insert() atau memasukkan beberapa item ke dalam list kosong.
Demonstration of list insert() method

```
odd = [1, 9]
odd.insert(1,3)

print(odd)

odd[2:2] = [5, 7]

print(odd)
```

Menghapus Elemen List

Kita dapat menghapus satu atau lebih item dari list menggunakan pernyataan del Python. Bahkan dapat menghapus daftar sepenuhnya.

```
# Deleting list items
my_list = ['p', 'r', 'o', 'b', 'l', 'e', 'm']

# delete one item
del my_list[2]
print(my_list)

# delete multiple items
del my_list[1:5]
print(my_list)

# delete the entire list
del my_list

# Error: List not defined
print(my_list)
```

Kita dapat menggunakan remove() untuk menghapus item yang diberikan atau pop() untuk menghapus item pada indeks yang diberikan.

Metode pop() menghapus dan mengembalikan item terakhir jika indeks tidak disediakan. Ini membantu kita menerapkan list sebagai sebuah stack (first in, last out data structure).

Dan, jika kita harus mengosongkan seluruh daftar, kita dapat menggunakan metode clear().

```
my_list = ['p','r','o','b','l','e','m']
my_list.remove('p')

# Output: ['r', 'o', 'b', 'l', 'e', 'm']
print(my_list)

# Output: 'o'
print(my_list.pop(1))

# Output: ['r', 'b', 'l', 'e', 'm']
print(my_list)

# Output: 'm'
print(my_list.pop())

# Output: ['r', 'b', 'l', 'e']
print(my_list)

my_list.clear()

# Output: []
print(my_list)
```

Akhirnya, kita juga dapat menghapus item dalam list dengan menetapkan list kosong ke sepotong elemen.

```
>>> my_list = ['p','r','o','b','l','e','m']
>>> my_list[2:3] = []
>>> my_list
['p', 'r', 'b', 'l', 'e', 'm']
>>> my_list[2:5] = []
>>> my_list
['p', 'r', 'm']
```

Python List Methods

Methods	Descriptions
append()	menambahkan elemen ke akhir list
extend()	menambahkan semua elemen list ke list lain
insert()	menyisipkan item pada indeks yang ditentukan
remove()	menghapus item dari list
pop()	mengembalikan dan menghapus elemen pada indeks yang diberikan
clear()	menghapus semua item dari list
index()	mengembalikan indeks item pertama yang cocok
count()	mengembalikan hitungan jumlah item yang diteruskan sebagai argumen
sort()	mengurutkan item dalam list dalam urutan naik (ascending)
reverse()	reverse urutan item dalam daftar
copy()	membuat salinan list

Tuple

Tuple di Python mirip dengan list. Perbedaan antara keduanya adalah bahwa kita tidak dapat mengubah elemen tuple setelah ditetapkan sedangkan kita dapat mengubah elemen-elemen list setelah ditetapkan.

Sifat tuple:

- Ordered
- Immutable
- Allow duplicates

Membuat Tuple

Tuple dibuat dengan menempatkan semua item (elemen) di dalam tanda kurung (), dipisahkan oleh koma.

Tuple dapat memiliki sejumlah item dan mungkin dari berbagai tipe (int, float, daftar, string, bool).

```
# Different types of tuples

# Empty tuple
my_tuple = ()
print(my_tuple)

# Tuple having integers
my_tuple = (1, 2, 3)
print(my_tuple)

# tuple with mixed data types
my_tuple = (1, "Hello", 3.4)
print(my_tuple)

# nested tuple
my_tuple = ("mouse", [8, 4, 6], (1, 2, 3))
print(my_tuple)

#Output
()
(1, 2, 3)
(1, 'Hello', 3.4)
('mouse', [8, 4, 6], (1, 2, 3))
```

Tuple juga dapat dibuat tanpa menggunakan tanda kurung. Ini dikenal sebagai tuple packing.

```
my_tuple = 3, 4.6, "dog"
print(my_tuple)

# tuple unpacking is also possible
a, b, c = my_tuple

print(a)      # 3
print(b)      # 4.6
print(c)      # dog

#Output
(3, 4.6, 'dog')
3
4.6
dog
```

Membuat tuple dengan satu elemen agak sedikit tricky.

Memiliki satu elemen dalam tanda kurung saja tidak cukup. Kita akan membutuhkan koma tambahan untuk menunjukkan bahwa itu adalah tuple.

```
my_tuple = ("hello")
print(type(my_tuple))  # <class 'str'>

# Creating a tuple having one element
my_tuple = ("hello",)
print(type(my_tuple))  # <class 'tuple'>

# Parentheses is optional
my_tuple = "hello",
print(type(my_tuple))  # <class 'tuple'>
```

Mengakses Elemen dari Tuple

Indexing

Kita dapat menggunakan operator indeks [] untuk mengakses item dalam tuple, di mana indeks dimulai dari 0.

Jadi, tuple yang memiliki 6 elemen akan memiliki indeks dari 0 hingga 5. Mencoba mengakses indeks di luar rentang indeks tuple(6,7,... dst) akan menaikkan IndexError.

Indeks harus berupa bilangan bulat, jadi kita tidak bisa menggunakan float atau jenis lainnya, atau akan menghasilkan TypeError.

Demikian juga, nested tuple diakses menggunakan nested indexing, seperti yang ditunjukkan pada contoh di bawah ini.
Accessing tuple elements using indexing

```
my_tuple = ('p','e','r','m','i','t')

print(my_tuple[0])   # 'p'
print(my_tuple[5])   # 't'

# IndexError: list index out of range
# print(my_tuple[6])

# Index must be an integer
# TypeError: list indices must be integers, not float
# my_tuple[2.0]

# nested tuple
n_tuple = ("mouse", [8, 4, 6], (1, 2, 3))

# nested index
print(n_tuple[0][3])       # 's'
print(n_tuple[1][1])       # 4
```

Negative Indexing

Python memungkinkan pengindeksan negatif untuk urutannya. Indeks -1 mengacu pada item terakhir, -2 ke item terakhir kedua dan seterusnya.

```
# Negative indexing for accessing tuple elements
my_tuple = ('p', 'e', 'r', 'm', 'i', 't')

# Output: 't'
print(my_tuple[-1])

# Output: 'p'
print(my_tuple[-6])
```

Slicing

Kita dapat mengakses berbagai item dalam tuple dengan menggunakan titik dua operator slicing **:** .

```
# Accessing tuple elements using slicing
my_tuple = ('j','a','k','a','r','t','a')

# elements 2nd to 4th
# Output: ('a', 'k', 'a')
print(my_tuple[1:4])

# elements beginning to 2nd
# Output: ('j', 'a')
print(my_tuple[:-5])

# elements 5th to end
# Output: ('t', 'a')
print(my_tuple[4:])

# elements beginning to end
# Output: ('j', 'a', 'k', 'a', 'r', 't', 'a')
print(my_tuple[:])
```

Mengubah Tuple

Tidak seperti list, tuple tidak dapat diubah.

Ini berarti bahwa elemen tuple tidak dapat diubah setelah ditugaskan. Namun, jika elemen itu sendiri adalah tipe data yang dapat diubah seperti list, item bersarangnya dapat diubah.

Kita juga dapat menetapkan tuple ke nilai yang berbeda (penugasan kembali).

```
# Changing tuple values
my_tuple = (4, 2, 3, [6, 5])

# TypeError: 'tuple' object does not support item assignment
# my_tuple[1] = 9

# However, item of mutable element can be changed
my_tuple[3][0] = 9    # Output: (4, 2, 3, [9, 5])
print(my_tuple)

# Tuples can be reassigned
my_tuple = ('j', 'a', 'k', 'a', 'r', 't', 'a')

# Output: ('j', 'a', 'k', 'a', 'r', 't', 'a')
```

```
print(my_tuple)
```

Kita dapat menggunakan operator + untuk menggabungkan dua tuple. Ini disebut concatenation.

Kita juga dapat mengulangi elemen dalam tupel untuk beberapa kali menggunakan operator *.

Operasi + dan * menghasilkan tuple baru.

```
# Concatenation
# Output: (1, 2, 3, 4, 5, 6)
print((1, 2, 3) + (4, 5, 6))

# Repeat
# Output: ('Repeat', 'Repeat', 'Repeat')
print(("Repeat",) * 3)
```

Menghapus Tuple

Seperti dibahas di atas, kita tidak dapat mengubah elemen dalam tuple. Ini berarti bahwa kita tidak dapat menghapus atau menghapus item dari tupel.

Menghapus tuple seluruhnya, dimungkinkan menggunakan kata kunci del.

```
# Deleting tuples
my_tuple = ('j', 'a', 'k', 'a', 'r', 't', 'a')

# can't delete items
# TypeError: 'tuple' object doesn't support item deletion
# del my_tuple[3]

# Can delete an entire tuple
del my_tuple

# NameError: name 'my_tuple' is not defined
print(my_tuple)
```

```
Traceback (most recent call last):
  File "<string>", line 12, in <module>
NameError: name 'my_tuple' is not defined
```

Tuple Methods

Metode yang menambahkan item atau menghapus item tidak tersedia dengan tuple. Hanya dua metode berikut yang tersedia. Beberapa contoh metode python tuple:

```
my_tuple = ('a', 'p', 'p', 'l', 'e',)

print(my_tuple.count('p'))  # Output: 2
print(my_tuple.index('l'))  # Output: 3
```

Tuple Membership Test

Kita dapat menguji apakah suatu item ada di tuple atau tidak, menggunakan kata kunci **in**.

```
# Membership test in tuple
my_tuple = ('a', 'p', 'p', 'l', 'e',)

# In operation
print('a' in my_tuple)
print('b' in my_tuple)

# Not in operation
print('g' not in my_tuple)

# Output
True
False
True
```

List of Python Tuples

```
>>> l = [(0.25,0.13), (0.51,0.86), (0.92,0.12), (0.64,0.72)]
>>> l
[(0.25, 0.13), (0.51, 0.86), (0.92, 0.12), (0.64, 0.72)]
```

Ketika tuple disimpan dalam list, kita dapat menggunakan fungsi list .sort(). Perilaku default fungsi .sort() menggunakan operator perbandingan >. Ini mengurutkan tuple berdasarkan elemen-elemennya sedangkan elemen tuple pertama adalah kunci utama, elemen kedua adalah kunci kedua, dll.

```
>>> l = [(5,8,9), (3,9,6), (2,6,3), (5,4,9), (2,5,4)]
>>> l.sort()
>>> l
[(2,5,4),(2,6,3),(3,9,6),(5,4,9),(5,8,9)]
```

```
>>> l = [(4, -2), (-2, 6)]
>>> l.sort()
>>> l
[(-2, 6), (4, -2)]
>>> l.sort(key=sum)
>>> l
[(4, -2), (-2, 6)]
```

Tuple of Python Lists

Tuples tidak hanya dapat menyimpan tipe data primitif seperti bool, int, float, atau str. tuple dapat berisi tipe data apa pun yang diinginkan. Misalnya dapat mendeklarasikan tupel yang berisi list:

```
>>> l = ['a','b']
>>> m = ['c','d']
>>> n = ['e','f']
>>> t = (l, m, n)
>>> t
(['a', 'b'], ['c', 'd'], ['e', 'f'])
```

Untuk mengakses elemen daftar yang terkandung dalam t, Anda dapat menggunakan []-operator:

```
>>> t[0][1]
'b'
>>> t[-1][-2]
'e'

# Change the list
>>> t[1][0] = 'z'
>>> t
(['a', 'b'], ['z', 'd'], ['e', 'f'])
```

Keuntungan Tuple dibanding List

Karena tupel sangat mirip dengan list, keduanya digunakan dalam situasi yang sama. Namun, ada keuntungan tertentu dari menerapkan tuple daripada list.

- Biasanya tuple digunakan untuk tipe data heterogen (berbeda) dan list untuk tipe data homogen (serupa).
- Karena tuple tidak dapat diubah (immutable), iterasi melalui tuple lebih cepat daripada dengan list. Jadi ada sedikit peningkatan kinerja.

- Jika kita memiliki data yang tidak berubah, menerapkannya sebagai tuple akan menjamin bahwa itu tetap dilindungi dari penulisan yang tidak diinginkan.

```
import timeit

list_time =
timeit.timeit('"".join(["a","b","c","d","e","f","g","h","i","j","k","l","m"])',
number=1000000)
print("list: ", list_time)

tuple_time =
timeit.timeit('"".join(("a","b","c","d","e","f","g","h","i","j","k","l","m"))',
number=1000000)
print("tuple:", tuple_time)
```

Set

Set adalah koleksi item yang tidak berurutan (unordered). Setiap elemen set unik (tidak ada duplikat) dan keseluruhan set itu dapat diubah. Kita dapat menambah atau menghapus item darinya.

Sifat set:
unordered, *unchangeable**, and *unindexed*.
Set *items* tidak dapat diubah (unchangeable), tapi kita bisa menambah atau menghapus items.

Set juga dapat digunakan untuk melakukan operasi himpunan matematika.

Membuat Sets

Set dibuat dengan menempatkan semua item (elemen) di dalam kurung kurawal {}, dipisahkan oleh koma, atau dengan menggunakan fungsi set().

Set dapat memiliki sejumlah item dan memungkinkan dari berbagai data type (int, float, tuple, string dll.). Tetapi set tidak dapat memiliki elemen yang dapat diubah (mutable) seperti list, set atau dict sebagai elemennya.

```
# Different types of sets in Python
# set of integers
my_set = {1, 2, 3}
print(my_set)
```

```
# set of mixed data types
my_set = {1.0, "Hello", (1, 2, 3)}
print(my_set)
# set cannot have duplicates
# Output: {1, 2, 3, 4}
my_set = {1, 2, 3, 4, 3, 2}
print(my_set)

# we can make set from a list
# Output: {1, 2, 3}
my_set = set([1, 2, 3, 2])
print(my_set)

# set cannot have mutable items
# here [3, 4] is a mutable list
# this will cause an error.

my_set = {1, 2, [3, 4]}

Traceback (most recent call last):
  File "<string>", line 15, in <module>
    my_set = {1, 2, [3, 4]}
TypeError: unhashable type: 'list'
```

Membuat set kosong cukup tricky.
Kurung kurawal kosong {} tidak akan membentuk set kosong, ini akan membentuk dict kosong.
Untuk membuat set tanpa elemen apa pun, kita menggunakan fungsi set() tanpa argumen apa pun.

```
# Distinguish set and dictionary while creating empty set

# initialize a with {}
a = {}
# check data type of a
print(type(a))

# initialize a with set()
a = set()
# check data type of a
print(type(a))

#Output
```

```
<class 'dict'>
<class 'set'>
```

Mengubah Sets

Set dapat diubah. Namun, karena mereka tidak teratur, pengindeksan tidak diperlukan.

Kita tidak dapat mengakses atau mengubah elemen set menggunakan pengindeksan atau slicing.

Kita dapat menambahkan satu elemen menggunakan metode add(), dan beberapa elemen menggunakan metode update(). Metode update() dapat menerima tuple, list, string atau set lain sebagai argumennya dan tidak akan terjadi duplikat.

```
# initialize my_set
my_set = {1, 3}
print(my_set)

# my_set[0]
# if you uncomment the above line
# you will get an error
# TypeError: 'set' object does not support indexing

# add an element
# Output: {1, 2, 3}
my_set.add(2)
print(my_set)

# add multiple elements
# Output: {1, 2, 3, 4}
my_set.update([2, 3, 4])
print(my_set)

# add list and set
# Output: {1, 2, 3, 4, 5, 6, 8}
my_set.update([4, 5], {1, 6, 8})
print(my_set)
```

```
{1, 3}
{1, 2, 3}
{1, 2, 3, 4}
```

```
{1, 2, 3, 4, 5, 6, 8}
```

Menghapus elemen dari satu set

Item tertentu dapat dihapus dari set menggunakan metode discard() dan remove().

Satu-satunya perbedaan antara keduanya adalah bahwa fungsi discard() membuat set tidak berubah jika elemen tidak ada di dalam set. Di sisi lain, fungsi remove() akan menimbulkan error dalam kondisi seperti itu (jika elemen tidak ada dalam set). Contoh berikut akan mengilustrasikan hal ini.

```
# Difference between discard() and remove()

# initialize my_set
my_set = {1, 3, 4, 5, 6}
print(my_set)

# discard an element
# Output: {1, 3, 5, 6}
my_set.discard(4)
print(my_set)

# remove an element
# Output: {1, 3, 5}
my_set.remove(6)
print(my_set)

# discard an element
# not present in my_set
# Output: {1, 3, 5}
my_set.discard(2)
print(my_set)

# remove an element
# not present in my_set
# you will get an error.
# Output: KeyError

my_set.remove(2)
```

```
{1, 3, 4, 5, 6}
{1, 3, 5, 6}
```

```
{1, 3, 5}
{1, 3, 5}
Traceback (most recent call last):
  File "<string>", line 28, in <module>
KeyError: 2
```

Demikian pula, kita dapat menghapus dan mengembalikan item menggunakan metode pop().

Karena set adalah tipe data yang tidak berurutan, tidak ada cara untuk menentukan item mana yang akan muncul. Ini benar-benar acak.

Kita juga dapat menghapus semua item dari satu set menggunakan metode clear().

```
# initialize my_set
# Output: set of unique elements
my_set = set("HelloWorld")
print(my_set)

# pop an element
# Output: random element
print(my_set.pop())

# pop another element
my_set.pop()
print(my_set)

# clear my_set
# Output: set()
my_set.clear()
print(my_set)

#Output
{'H', 'l', 'r', 'W', 'o', 'd', 'e'}
H
{'r', 'W', 'o', 'd', 'e'}
set()
```

Python Set Operations

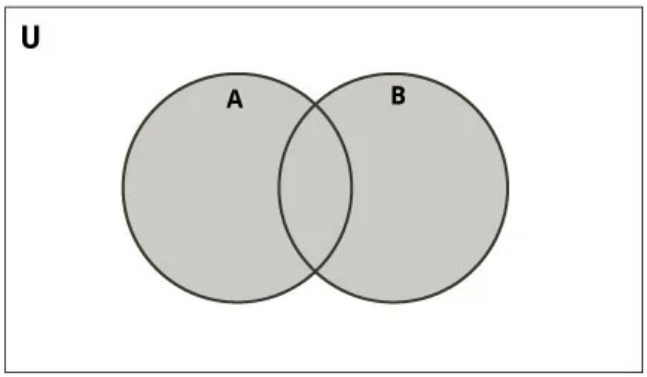
Set dapat digunakan untuk melakukan operasi himpunan matematika seperti union, intersection, difference and symmetric difference.

```
>>> A = {1, 2, 3, 4, 5}
```

```
>>> B = {4, 5, 6, 7, 8}
```

Set Union

Union A dan B adalah gabungan semua elemen dari kedua set.
Union dilakukan menggunakan | Operator. Hal yang sama dapat dilakukan dengan menggunakan metode union().



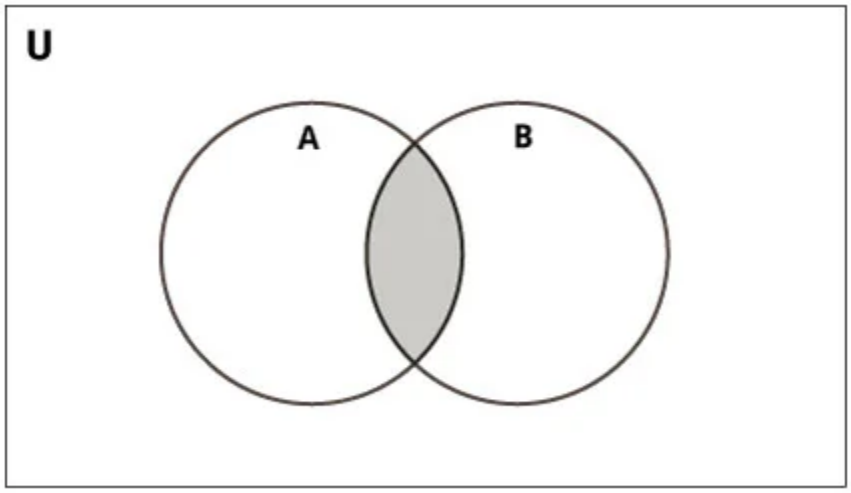

```
# Set union method
# initialize A and B
A = {1, 2, 3, 4, 5}
B = {4, 5, 6, 7, 8}

# use | operator
# Output: {1, 2, 3, 4, 5, 6, 7, 8}
print(A | B)

#Output
{1, 2, 3, 4, 5, 6, 7, 8}
```

```
# use union function
>>> A.union(B)
{1, 2, 3, 4, 5, 6, 7, 8}

# use union function on B
>>> B.union(A)
{1, 2, 3, 4, 5, 6, 7, 8}
```

Set Intersection

Intersection A dan B adalah seperangkat elemen yang memiliki kesamaan di kedua set.

Intersection dilakukan menggunakan & operator. Hal yang sama dapat dilakukan dengan menggunakan metode intersection().



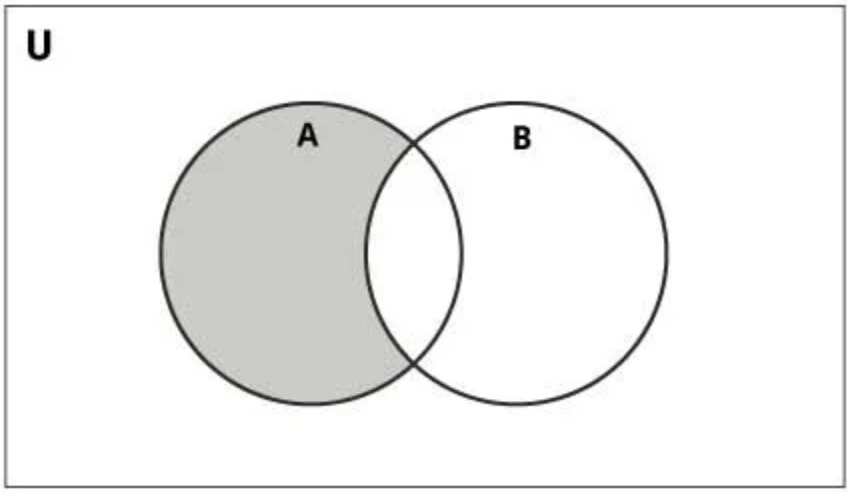

```
# Intersection of sets
# initialize A and B
A = {1, 2, 3, 4, 5}
B = {4, 5, 6, 7, 8}

# use & operator
# Output: {4, 5}
print(A & B)

#Output
{4, 5}
```

```
# use intersection function on A
>>> A.intersection(B)
{4, 5}

# use intersection function on B
>>> B.intersection(A)
{4, 5}
```

Set Difference

Perbedaan himpunan B dari himpunan A(A - B) adalah sekumpulan elemen yang hanya ada di A tetapi tidak dalam B. Demikian pula, B - A adalah satu set elemen dalam B tetapi tidak dalam A.

Perbedaan dilakukan menggunakan - operator. Hal yang sama dapat dilakukan dengan menggunakan metode difference().



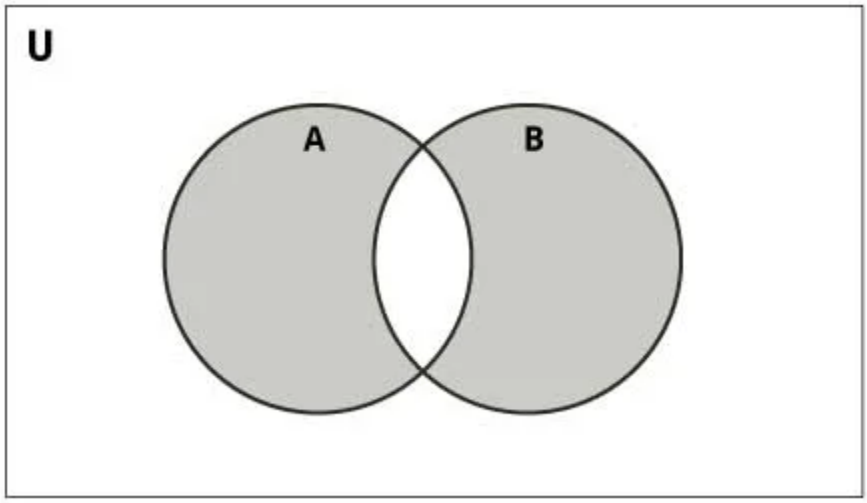

```
# Difference of two sets
# initialize A and B
A = {1, 2, 3, 4, 5}
B = {4, 5, 6, 7, 8}

# use - operator on A
# Output: {1, 2, 3}
print(A - B)

#Output
{1, 2, 3}
```

```
# use difference function on A
>>> A.difference(B)
{1, 2, 3}

# use - operator on B
>>> B - A
{8, 6, 7}

# use difference function on B
>>> B.difference(A)
{8, 6, 7}
```

Set Symmetric Difference

Perbedaan Simetris A dan B adalah sekelompok elemen dalam A dan B tetapi tidak di keduanya (tidak termasuk intersection).

Perbedaan simetris dilakukan menggunakan operator ^. Hal yang sama dapat dilakukan dengan menggunakan metode symmetric_difference().




```
# Symmetric difference of two sets
# initialize A and B
A = {1, 2, 3, 4, 5}
B = {4, 5, 6, 7, 8}

# use ^ operator
# Output: {1, 2, 3, 6, 7, 8}
print(A ^ B)

#Output
{1, 2, 3, 6, 7, 8}
```

```
# use symmetric_difference function on A
>>> A.symmetric_difference(B)
{1, 2, 3, 6, 7, 8}

# use symmetric_difference function on B
>>> B.symmetric_difference(A)
{1, 2, 3, 6, 7, 8}
```

Dictionary

Dictionary Python adalah kumpulan item yang tidak berurutan. Setiap item dictionary memiliki pasangan key/value.

Dictionary dioptimalkan untuk mengambil nilai saat key diketahui.

Membuat Python Dictionary

Membuat dictionary semudah menempatkan item di dalam kurung kurawal {} yang dipisahkan oleh koma.

Item memiliki key dan value yang sesuai yang dinyatakan sebagai pasangan (key: value).

Meskipun nilainya dapat dari tipe data apa pun dan dapat diulang, key harus dari tipe yang tidak dapat diubah (string, number atau tuple) dan harus unik. Kita juga dapat membuat dictionary dengan fungsi dict()

```
# empty dictionary
my_dict = {}

# dictionary with integer keys
my_dict = {1: 'apple', 2: 'ball'}

# dictionary with mixed keys
my_dict = {'name': 'John', 1: [2, 4, 3]}

# using dict()
my_dict = dict({1:'apple', 2:'ball'})

# from sequence having each item as a pair
my_dict = dict([(1,'apple'), (2,'ball')])
```

Python Dictionary from Lists

```
>>> keys = ['is_cat', 'name', 'age_years', 'weight_kgs']
>>> values = [True, "Mr. Fluffers", 7, 6.1]
>>> d = dict(zip(keys, values))
>>> d
{'is_cat': True, 'name': 'Mr. Fluffers', 'age_years': 7, 'weight_kgs': 6.1}
```

Untuk mencocokkan key dan value dari kedua, fungsi zip() digunakan dan diteruskan ke fungsi dict().
Jika kita tidak ingin mengatur value saat membuat dict baru, tapi kita sudah tahu key-nya, maka menggunakan fungsi .fromkeys()

```
>>> keys = ['is_cat', 'name', 'age_years', 'weight_kgs']
>>> dict.fromkeys(keys)
{'is_cat': None, 'name': None, 'age_years': None, 'weight_kgs': None}
```

```
>>> keys = ['is_cat', 'name', 'age_years', 'weight_kgs']
>>> dict.fromkeys(keys, 0)
{'is_cat': 0, 'name': 0, 'age_years': 0, 'weight_kgs': 0}
```

Mengakses Elemen dalam Dictionary

Jika pengindeksan digunakan dengan tipe data lain untuk mengakses value, dictionary menggunakan key. Ket dapat digunakan baik di dalam tanda kurung siku [] atau dengan metode get().

Jika kita menggunakan tanda kurung siku [], KeyError dimunculkan jika key tidak ditemukan dalam dictionary. Di sisi lain, metode get() mengembalikan None jika key tidak ditemukan.

```
# get vs [] for retrieving elements
my_dict = {'name': 'Jack', 'age': 26}

# Output: Jack
print(my_dict['name'])

# Output: 26
print(my_dict.get('age'))

# Trying to access keys which doesn't exist throws error
# Output None
print(my_dict.get('address'))

# KeyError
print(my_dict['address'])

#Output
Jack
26
None
```

```
Traceback (most recent call last):
  File "<string>", line 15, in <module>
    print(my_dict['address'])
KeyError: 'address'
```

Fungsi .items() memungkinkan kita mengakses pasangan key value sebagai tuple dalam list:

```
>>> d = {'a': 1, 'b': 2, 'c': 3}
>>> l = list(d.items())
[('a', 1), ('b', 2), ('c', 3)]
>>> l[0]
('a', 1)
```

Python juga menyediakan fungsi khusus untuk menambah atau memperbarui pasangan key value dengan fungsi .update(). Fungsi .update() mengambil dict lain sebagai argumen atau list 2-tuple yang mewakili pasangan key value:

```
>>> d = {'a': 1, 'b': 2, 'c': 3}
>>> d.update({'a':0})
>>> d
{'a': 0, 'b': 2, 'c': 3}
>>> d.update({'d':4})
>>> d
{'a': 0, 'b': 2, 'c': 3, 'd': 4}
>>> d.update([('e',5)])
>>> d
{'a': 0, 'b': 2, 'c': 3, 'd': 4, 'e': 5}
```

Menambah dan Mengubah Elemen di dalam Dict

Dict dapat diubah/mutable. Kita dapat menambahkan item baru atau mengubah nilai item yang ada menggunakan.

Jika key sudah ada, maka nilai yang ada akan diperbarui. Jika key tidak ada, pasangan baru (key: value) akan ditambahkan ke dict.

```
# Changing and adding Dictionary Elements
my_dict = {'name': 'Jack', 'age': 26}

# update value
my_dict['age'] = 27

#Output: {'age': 27, 'name': 'Jack'}
```

```
print(my_dict)

# add item
my_dict['address'] = 'Downtown'

# Output: {'address': 'Downtown', 'age': 27, 'name': 'Jack'}
print(my_dict)

#Output
{'name': 'Jack', 'age': 27}
{'name': 'Jack', 'age': 27, 'address': 'Downtown'}
```

Menghapus Elemen dalam Dict

Kita dapat menghapus item tertentu dalam dict dengan menggunakan metode pop(). Metode ini menghapus item dengan key yang disediakan dan mengembalikan nilainya.

Metode popitem() dapat digunakan untuk menghapus dan mengembalikan pasangan item (key, value) dari dict. Semua item dapat dihapus sekaligus, menggunakan metode clear().

Kita juga dapat menggunakan kata kunci del untuk menghapus item individual atau seluruh dict itu sendiri.

```
# Removing elements from a dictionary

# create a dictionary
squares = {1: 1, 2: 4, 3: 9, 4: 16, 5: 25}

# remove a particular item, returns its value
# Output: 16
print(squares.pop(4))

# Output: {1: 1, 2: 4, 3: 9, 5: 25}
print(squares)

# remove an arbitrary item, return (key,value)
# Output: (5, 25)
print(squares.popitem())

# Output: {1: 1, 2: 4, 3: 9}
print(squares)
```

```
# remove all items
squares.clear()

# Output: {}
print(squares)

# delete the dictionary itself
del squares

# Throws Error
print(squares)
```

```
16
{1: 1, 2: 4, 3: 9, 5: 25}
(5, 25)
{1: 1, 2: 4, 3: 9}
{}
Traceback (most recent call last):
  File "<string>", line 30, in <module>
    print(squares)
NameError: name 'squares' is not defined
```

Fungsi .popitem() tidak memiliki parameter dan menghapus pasangan nilai kunci secara acak dari dict

```
>>> d = {'a': 1, 'b': 2, 'c': 3}
>>> d.popitem()
('c', 3)
>>> d
{'a': 1, 'b': 2}
```

Menghapus semua dict

```
>>> d = {'a': 1, 'b': 2, 'c': 3}
>>> d.clear()
>>> d
{}
```

Merging Python Dictionaries

Ketika ada dua dict dan ingin digabungkan menjadi satu gunakan operator pipe (|). Jika dua dict yang harus digabungkan tidak memiliki key yang sama, kita mendapatkan dict yang berisi

semua pasangan key value dari kedua dict. Namun, ketika key ada di kedua dict, nilai dict kedua digabungkan ke dalam yang baru:

```
>>> d = {'a': 1, 'b': 2}
>>> e = {'c': 3, 'd': 4}
>>> d | e
{'a': 1, 'b': 2, 'c': 3, 'd': 4}
>>> f = {'b': -1, 'c': 3}
>>> d | f
{'a': 1, 'b': -1, 'c': 3}
>>> f | d
{'b': 2, 'c': 3, 'a': 1}
```

Nested Python Dictionaries

Misalnya, kita punya beberapa hewan peliharaan dan ingin menyimpan informasi tentang mereka secara terstruktur. Dalam hal ini, kita dapat menggunakan nested dict Python dan menggunakan operator [] nested untuk mengakses dan memperbarui data:

```
>>> pets = {
'Mr. Fluffers': {'species': 'cat', 'age_years': 7, 'weight_kgs': 6.1},
'Rambo': {'species': 'dog', 'age_years': 4, 'weight_kgs': 31 }
}

>>> pets['Mr. Fluffers']['species']
'cat'

>>> pets['Rambo']['age_years'] = 5

>>> pets
{'Mr. Fluffers': {'species': 'cat', 'age_years': 7, 'weight_kgs': 6.1},
'Rambo': {'species': 'dog', 'age_years': 5, 'weight_kgs': 31}}
```

Mendapatkan Keys & Values

Fungsi .values() mengembalikan semua nilai dalam dict:

```
>>> d = {'a': 1, 'b': 2, 'c': 3}
>>> d.values()
dict_values([1, 2, 3])
```

Fungsi .keys() yang mengembalikan semua kunci dalam dict:

```
>>> d = {'a': 1, 'b': 2, 'c': 3}
>>> d.keys()
dict_keys(['a', 'b', 'c'])
```

Date Data Type

Python memiliki modul bernama datetime untuk bekerja dengan tanggal dan waktu.

```
import datetime

datetime_object = datetime.datetime.now()
print(datetime_object)
#Output
2018-12-19 09:26:03.478039
```

```
import datetime

date_object = datetime.date.today()
print(date_object)
#Output
2018-12-19
```

Objek tanggal untuk mewakili tanggal

```
import datetime

d = datetime.date(2019, 4, 13)
print(d)
#Output
2019-04-13
```

Dapatkan tanggal dari timestamp

Kita juga dapat membuat objek tanggal dari timestamp. **timestamp** Unix adalah jumlah detik antara tanggal tertentu dan 1 Januari 1970 di UTC. Kita dapat mengonversi **timestamp** ke tanggal menggunakan metode fromtimestamp().

```
from datetime import date

timestamp = date.fromtimestamp(1326244364)
print("Date =", timestamp)
```

```
Date = 2012-01-11
```

Mendapatkan tanggal, tahun, tanggal

```
from datetime import date

# date object of today's date
today = date.today()

print("Current year:", today.year)
print("Current month:", today.month)
print("Current day:", today.day)
```

Perbedaan antara dua tanggal dan waktu

```
from datetime import datetime, date

t1 = date(year = 2018, month = 7, day = 12)
t2 = date(year = 2017, month = 12, day = 23)
t3 = t1 - t2
print("t3 =", t3)

t4 = datetime(year = 2018, month = 7, day = 12, hour = 7, minute = 9,
second = 33)
t5 = datetime(year = 2019, month = 6, day = 10, hour = 5, minute = 55,
second = 13)
t6 = t4 - t5
print("t6 =", t6)

print("type of t3 =", type(t3))
print("type of t6 =", type(t6))
#Output
t3 = 201 days, 0:00:00
t6 = -333 days, 1:14:20
type of t3 = <class 'datetime.timedelta'>
type of t6 = <class 'datetime.timedelta'>
```

Perbedaan antara dua tanggal dan waktu menggunakan timedelta

```
from datetime import timedelta

t1 = timedelta(weeks = 2, days = 5, hours = 1, seconds = 33)
t2 = timedelta(days = 4, hours = 11, minutes = 4, seconds = 54)
```

```
t3 = t1 - t2

print("t3 =", t3)
```

Python format datetime

Cara tanggal dan waktu diwakili mungkin berbeda di tempat, organisasi, dll yang berbeda. Lebih umum menggunakan mm/dd/yyyy di USA, sedangkan dd/mm/yyyy lebih umum di UK.

Python memiliki metode strftime() dan strptime() untuk menangani ini.

Python strftime() - datetime objek ke string

Metode strftime() didefinisikan di dalam class tanggal, tanggal waktu, dan waktu. Metode ini akan membuat string yang diformat dari objek tanggal, datetime, atau waktu tertentu.

```
from datetime import datetime

# current date and time
now = datetime.now()

t = now.strftime("%H:%M:%S")
print("time:", t)

s1 = now.strftime("%m/%d/%Y, %H:%M:%S")
# mm/dd/YY H:M:S format
print("s1:", s1)

s2 = now.strftime("%d/%m/%Y, %H:%M:%S")
# dd/mm/YY H:M:S format
print("s2:", s2)

#Output
time: 04:34:52
s1: 12/26/2018, 04:34:52
s2: 26/12/2018, 04:34:52
```

```
%Y - year [0001,..., 2018, 2019,..., 9999]
%m - month [01, 02, ..., 11, 12]
%d - day [01, 02, ..., 30, 31]
```

```
%H - hour [00, 01, ..., 22, 23
%M - minute [00, 01, ..., 58, 59]
%S - second [00, 01, ..., 58, 59]
```

Python strptime() - string ke datetime

Metode strptime() akan membuat objek datetime dari string tertentu (mewakili tanggal dan waktu).

```
from datetime import datetime

date_string = "21 June, 2018"
print("date_string =", date_string)

date_object = datetime.strptime(date_string, "%d %B, %Y")
print("date_object =", date_object)

#Output
date_string = 21 June, 2018
date_object = 2018-06-21 00:00:00
```

Timezone

Misalkan, kita sedang mengerjakan sebuah proyek dan perlu menampilkan tanggal dan waktu berdasarkan zona waktu mereka.

```
from datetime import datetime
import pytz

local = datetime.now()
print("Local:", local.strftime("%m/%d/%Y, %H:%M:%S"))


tz_NY = pytz.timezone('America/New_York')
datetime_NY = datetime.now(tz_NY)
print("NY:", datetime_NY.strftime("%m/%d/%Y, %H:%M:%S"))

tz_London = pytz.timezone('Europe/London')
datetime_London = datetime.now(tz_London)
print("London:", datetime_London.strftime("%m/%d/%Y, %H:%M:%S"))
```

```
#Output
Local time: 2018-12-20 13:10:44.260462
America/New_York time: 2018-12-20 13:10:44.260462
Europe/London time: 2018-12-20 13:10:44.260462
```