Python Programming

Dictionaries

Kristof Michiels

Inhoud

- Wat zijn dictionaries?
- Werken met dictionaries
  - Toegang tot values
  - Een key-value-paar toevoegen
  - Een value wijzigen
  - Key-value-paren verwijderen

Inhoud

- Door een dictionary itereren met een loop
  - Door alle key-value-paren itereren
  - Itereren door alle keys (in een bepaalde volgorde)
  - Itereren door alle values
- Nesten
  - Een list van dictionaries
  - Een list in een dictionary
  - Een dictionary in een dictionary
- Dictionaries, lists en functies

Wat zijn dictionaries?

Wat zijn dictionaries?

- Net als lists laten dictionaries toe om verschillende (tot heel veel!) waarden toe te kennen aan één variabele
- In dit voor developers zeer belangrijk datatype sla je data op onder de vorm van key-value (sleutel-waarde) paren
- Waarden zijn voorzien van een unieke key, die een daarmee structuur en leesbaarheid aan je data geven
- Keys kunnen integers zijn, maar ook floats, strings, booleans, tot zelfs lists of dictionaries
- Bij getallen als key: de volgorde of startgetal zijn willekeurig
- De waarden hoeven niet uniek te zijn

Enkele voorbeelden van dictionaries

```
kunstenaars_geboortejaar = {
    "Keith Haring": 1958,
    "Jean-Michel Basquiat": 1960,
    "Andy Warhol": 1928,
}

andy_warhol = {
    "geboren": 1928,
    "gestorven": 1987,
    "stroming": "pop-art",
    "quote": "In the future everyone will be famous for fifteen minutes",
}

keith_haring = {} # een nieuwe, lege dictionary

print(type(kunstenaars_geboortejaar)) #<class 'dict'="">
```

Dictionaries

- Een dictionary-notatie bestaat uit accolades met één of meerdere key-value-paren erin
- Elk nieuw paar wordt toegevoegd aan het einde van de dictionary
- De key-value paren worden in de volgorde opgeslagen waarin ze werden toegevoegd (sinds Python 3.7)
- Het verwijderen van een key-value paar verandert niets aan de volgorde van de overblijvende paren
- Een nieuwe - lege - dictionary wordt aangemaakt en toegekend aan een variabele door gebruik te maken van accolades
- Een niet-lege dictionary kan worden gecreëerd door de key-value-waarden te scheiden door komma's
- Een dubbelpunt wordt als notatie gebruikt om elk key-value paar te scheiden

Werken met dictionaries

Toegang tot values: via de key

```
student_1 = {"favoriete_taal": "Python"}
print(student_1["favoriete_taal"])

student_2 = {"favoriete_taal": "JavaScript", "favoriete_serie": "Yellowjackets"}
print(student_2["favoriete_serie"])
```

Toegang tot de values kan ook met de get()-method. Voordeel is dat je een default waarde kan meegeven als de opgevraagde key niet zou bestaan. Indien je het tweede argument niet meegeeft, dan wordt indien de key niet bestaat de waarde None teruggegeven:

```
student_1 = {"favoriete_taal": "Python"}
print(student_1.get("favoriete_serie", "Geen favoriete serie meegegeven."))
```

Een key-value-paar toevoegen

```
student_1 = {"favoriete_taal": "Python"}
student_1["favoriete_serie"] = "Station Eleven"
student_1["favoriete_muziek"] = "Little Simz"

# of vertrekkend van een lege dictionary:

student_3 = {}
student_3["favoriete_taal"] = "Go"
student_3["favoriet_frontend_framework"] = "React"
```

Een value wijzigen

Je doet dit door te verwijzen naar de naam van de dictionary onmiddellijk gevolgd door de key tussen vierkante haakjes gevolgd door de nieuwe waarde die je eraan wenst te geven.

```
student_1 = {"favoriete_taal": "Python"}
student_1["favoriete_taal"] = "JavaScript"
```

Key-value-paren verwijderen

Met het del-trefwoord:

```
student_1 = {"favoriete_taal": "Python", "favoriete_kleur": "rood"}
del student_1["favoriete_kleur"]
```

Of met de pop()-method. De key moet steeds worden meegegeven. De value van deze key wordt teruggegeven en kan indien gewenst worden toegekend aan een value. Dit is hier te zien:

```
student_1 = {"favoriete_taal": "Python", "favoriete_kleur": "rood"}
kleur_van_student_1 = student_1.pop("favoriete_kleur")
```

De len()-functie

- Deze functie, die we al zagen bij lists, kan ook gebruikt worden om te weten te komen hoeveel key-value-paren een bepaalde dictionary bevat
- De dictionary wordt als enige argument meegegeven aan de functie
- Als resultaat komt het aantal key-value-paren terug. Bij een lege dictionary komt de waarde 0 terug

```
kunstenaars_geboortejaar = {
    "Keith Haring": 1958,
    "Jean-Michel Basquiat": 1960,
    "Andy Warhol": 1928,
}
print(len(kunstenaars_geboortejaar))
```

Door een dictionary itereren met een loop

Door een dictionary itereren met een loop

- Soms bevat een dictionary een groot aantal key-value-paren
- Bij groot mag je gerust aan 100000-en of zelfs nog veel meer denken
- Vaak ga je doorheen die collectie willen itereren en dat doe je met een loop

Door alle key-value-paren itereren

```
favoriete_talen = {
    "Kristof": "Python",
    "Tim": "C#",
    "Tom": "JavaScript",
}

for key, value in favoriete_talen.items():
    print(f"{key} heeft {value} als favoriete programmeertaal")
```

De in-operator

Wordt gebruikt om te bepalen of een bepaalde key (of value) aanwezig is. De operator evalueert in True als de key effectief aanwezig is, anders evalueert ze in False. Het resultaat kan overal gebruikt worden waar een Booleaanse waarde verwacht wordt (bvb in de voorwaarde van een if-statement of loop).

```
if "Keith Haring" in kunstenaars_geboortejaar.keys():
    print("Over Keith Haring hebben we al informatie toegevoegd.")

if 1967 not in kunstenaars_geboortejaar.values():
    print("We hebben nog geen kunstenaar geboren in 1967.")
```

Itereren door alle keys

```
favoriete_talen = {
    "Kristof": "Python",
    "Tim": "C#",
    "Tom": "JavaScript",
}

for key in favoriete_talen.keys():
    print(f"{key} heeft een favoriete programmeertaal")

#loopen door de keys is ook het default gedrag bij loopen door een dictionary
# dus hetzelfde als schrijven

for key in favoriete_talen:
    print(f"{key} heeft een favoriete programmeertaal")
```

Itereren door alle keys in een bepaalde volgorde

De natuurlijke volgorde van de elementen in een dictionary wordt bepaald door de volgorde waarin ze zijn toegevoegd. Wil je de keys evenwel sorteren dan kan je daar de sorted()-functie voor gebruiken.

```
favoriete_letters = {
    "o": "O is een otter, die zwemt in het meer",
    "k": "K is een koopman, die koffie verzond",
    "a": "A is een aapje, dat eet uit zijn poot",
}

for key, value in sorted(favoriete_letters.items()):
    print(value)
```

Itereren door alle values

```
favoriete_talen = {
    "Kristof": "Python",
    "Tim": "C#",
    "Tom": "JavaScript",
    "Hassan": "Python",
}

for favoriete_taal in favoriete_talen.values():
    print(favoriete_taal)
```

Herhalingen vermijden door gebruik van een set

```
favoriete_talen = {
    "Kristof": "Python",
    "Tim": "C#",
    "Tom": "JavaScript",
    "Hassan": "Python",
}

for favoriete_taal in set(favoriete_talen.values()):
    print(favoriete_taal)

# een set maken:

kleuren = {"blauw", "zwart", "geel", "oranje"}
```

Herhalingen vermijden door gebruik van een set

- Een set is een verzameling van elementen waarbij elk element uniek moet zijn. De set()-functie identificeert de unieke elementen in een lijst en bouwt een set uit die elementen
- Sets lijken wat op dictionaries omdat ze verpakt zitten in accolades. Ze bevatten evenwel geen key-value-paren maar enkelvoudige waarden

Nesten

Nesten

- Lists en dictionaries zijn zeer veelzijdige datatypes waarmee je veel richtingen uitkunt
- Zo kan je verschillende dictionaries onderbrengen in een list, of een list gebruiken als value gebruiken in een dictionary
- Je kan zelfs dictionaries gebruiken als values in een dictionary
- Dit noemen we nesten

Een list van dictionaries

```
kristof = {"favoriete_taal": "Python", "favoriete_muziek": "The War on Drugs"}
tim = {"favoriete_taal": "C#", "favoriete_muziek": "Lana Del Rey"}
tom = {"favoriete_taal": "JavaScript", "favoriete_muziek": "Admiral Freebee"}

lectoren = [kristof, tim, tom]
```

Uiteraard zou dit ook als één enkel statement kunnen geschreven worden.

Een list in een dictionary

```
kristof = {
    "favoriete_taal": "Python",
    "favoriete_muziek": "The War on Drugs",
    "favoriete_series": ["Yellowjackets", "Station Eleven", "Righteous Gemstones"],
}

for serie in kristof["favoriete_series"]:
    print(serie)
```

Een list in een dictionary

```
lotto_trekkingen = {
    "2022-02-19": [6,15,18,30,39,41,2],
    "2022-02-16": [2,3,14,18,31,39,45],
    "2022-02-12": [2,3,19,30,37,40,15],
}

for trekking, resultaten in lotto_trekkingen.items():
    print(f"{trekking}: ")
    for resultaat in resultaten:
        print(f"\t{resultaat}")
```

Een dictionary in een dictionary

```
lectoren = {
    "Kristof": {
        "favoriete_taal": "Python",
        "favoriete_muziek": "The War on Drugs",
        "favoriete_serie": "Better Call Saul",
    },
    "Tim": {
        "favoriete_taal": "C#",
        "favoriete_muziek": "Lana Del Rey",
        "favoriete_serie": "Station Eleven",
    },
    "Tom": {
        "favoriete_taal": "JavaScript",
        "favoriete_muziek": "Admiral Freebee",
        "favoriete_serie": "The Righteous Gemstones",
    },
}
```

Dictionaries, lists en functies

Dictionaries, lists en functies

- Dictionaries (en lists) kunnen net als andere datatypes als argumenten aan functies worden meegegeven, of als returnwaarde worden teruggegeven
- Dit laat ons toe om meerdere of een variërend aantal waarden mee te geven of te ontvangen

Een willekeurig aantal argumenten meegeven

- Python laat toe dat functies een willekeurig aantal argumenten kunnen ontvangen
- De asterisk in de parameternaam draagt Python op om een tuple aan te maken en alle waarden die meegegeven worden in deze tuple op te slaan

```
def geef_favoriete_series(*series):
    """Deze functie drukt een reeks favoriete tv-series af """
    for serie in series:
        print(serie)

geef_favoriete_series("Yellowjackets")
geef_favoriete_series("Yellowjackets", "Station Eleven", "The Righteous Gemstones")
```

Positionele en willekeurige argumenten combineren

- Positionele en willekeurige argumenten kunnen zonder problemen worden gecombineerd
- De parameter die een willekeurig aantal argumenten accepteert moet steeds als laatste in de functie-definitie worden geplaatst

```
def geef_favoriete_series(naam_kijker, *series):
    """Deze functie drukt een reeks favoriete tv-series af """
    print(naam_kijker)
    for serie in series:
        print(serie)

geef_favoriete_series("Kristof", "Yellowjackets")
geef_favoriete_series("Tim", "Yellowjackets", "Station Eleven", "Ted Lasso")
```

Willekeurige trefwoordargumenten gebruiken

- In het vorige voorbeeld accepteerde de functie een willekeurig aantal series als argument
- in bepaalde situaties wil je een functie evenwel in staat stellen een willekeurig aantal verschillende argumenten te accepteren
- In dat geval kan je je functie een ongelimiteerd aantal key-value-paren laten aannemen
- De dubbele asterisk voor het argument draagt Python op een lege dictionary aan te maken en er de eventuele binnenkomende key-value paren in onder te brengen

Willekeurige trefwoordargumenten gebruiken

```
def verzamel_gebruikersinformatie(voornaam, familienaam, **extra_informatie):
    """Een dictionary bouwen met alle informatie die we over de gebruiker kennen"""
    extra_informatie["voornaam"] = voornaam
    extra_informatie["naam"] = familienaam
    return extra_informatie

kristof_info = verzamel_gebruikersinformatie("Kristof", "Michiels",
    favoriete_taal="Python", favoriete_muziek="Little Simz",
    favoriete_serie="Station Eleven")

print(kristof_info)
```

Python Programming - les 5 -

kristof.michiels01@ap.be