

# Python Programming

## Werken met lists

**Kristof Michiels**

# ! Disclaimer

- Dit is een alfa-versie van de cursus die zal gebruikt worden voor het vak Python Programming in semester 2
- Deze informatie is dus mogelijk nog onvolledig en kan nog onzorgvuldigheden of zelfs fouten bevatten

# Lesinhoud

# Lesinhoud

- Wat zijn lists?
- Elementen wijzigen, toevoegen en verwijderen
- Een list ordenen
- Een list doorlopen
- Werken met delen van een list
- Tuples

Wat zijn lists?

# Wat zijn lists?

```
python_data_types = ["str", "int", "float", "complex", "list", "tuple", "range", \
"dict", "set", "frozenset", "bool", "bytes", "bytearray", "memoryview"]
boodschap = f"Een bekend data-type is de {python_data_types[0]}."
print(boodschap)
```

- Een *list* is één van de ingebouwde data-types in Python
- Zoals de naam het al zegt wordt het gebruikt om verzamelingen met data op te slaan
- Het voordeel van een *list* is dat data gestructureerd kan worden opgeslagen, in volgorde, gesorteerd, enz...
- Een *list* wordt aangemaakt door middel van deze [ ] vierkante haakjes en de individuele waarden zijn gescheiden door komma's
- Als naam voor een *list* wordt vaak een meervoud gekozen

# Toegang tot elementen in een *list*

```
weekdagen = ["maandag", "dinsdag", "woensdag", "donderdag", "vrijdag"]
print(weekdagen) #['maandag', 'dinsdag', 'woensdag', 'donderdag', 'vrijdag']
print(weekdagen[0])
print(weekdagen[1])
boodschap = f"Bijna weekend, het is vandaag {boodschap[4]}."
print(boodschap)
```

- Vraag je Python een *list* te printen dan krijg je de volledige list terug, inclusief de vierkante haakjes
- *Lists* zijn geordende collecties, dus kan je via de positie toegang krijgen tot de elementen
- Je gebruikt hiervoor de naam van de *list*, gevolgd door het index-getal van het gewenste element geplaatst binnen vierkante haakjes
- Het index-getal van het eerste element in de lijst is 0, tweede element is 1 enz...

# Toegang tot elementen in een *list*

```
weekdagen = ["maandag", "dinsdag", "woensdag", "donderdag", "vrijdag"]  
boodschap = f"Bijna weekend, het is vandaag {weekdagen[-1]}."  
print(boodschap)
```

- Het laatste element krijg je door het gebruik van het index-getal -1, het voorlaatste element met -2 enz...



# Elementen wijzigen, toevoegen en verwijderen

Lists worden vaak dynamisch gebruikt. We moeten dus elementen kunnen wijzigen, toevoegen en verwijderen in run-time

# Elementen wijzigen in een list

```
lievelingskleuren = ["oranje", "groen", "rood"]  
lievelingskleuren[0] = "geel"
```

- We kunnen elementen wijzigen door de naam van de lijst op te roepen, gevolgd door de index van het te wijzigen element
- Als we er vervolgens een waarde aan toekennen dan wordt dit de nieuwe waarde van dit element

# Elementen toevoegen met de append()-method

```
lievelingskleuren = []  
lievelingskleuren.append("oranje")  
lievelingskleuren.append("groen")  
lievelingskleuren.append("rood")  
print(lievelingskleuren) #['oranje', 'groen', 'rood']
```

- Er zijn verschillende manieren om nieuwe data toe te voegen aan een bestaande list
- Met de append()-method voeg je elementen toe aan het einde van een list
- Deze method maakt het eenvoudig om dynamisch te vertrekken vanaf een lege list. Met elke append()-call voeg je een nieuw element toe aan de lijst

# Elementen toevoegen met de insert()-method

```
lievelingskleuren = ["oranje", "groen", "rood"]  
lievelingskleuren.insert(0, "geel")  
print(lievelingskleuren) #['geel', 'oranje', 'groen', 'rood']
```

- De insert()-method laat toe elementen toe te voegen op elke positie in een list
- Je doet dit door de index en de waarde van het nieuwe element mee te geven
- In het bovenstaande voorbeeld voegen we een element toe bij positie 0. Alle huidige elementen schuiven hierdoor een plaats door naar rechts

# Elementen verwijderen met het del statement

```
lievelingskleuren = ["oranje", "groen", "rood"]  
del lievelingskleuren[0] #['groen', 'rood']  
print(lievelingskleuren)  
del lievelingskleuren[1]  
print(lievelingskleuren) #['groen']
```

Ken je de positie van een element dan kan je het verwijderen met het del statement.

# Elementen verwijderen met de pop()-method

```
lievelingskleuren = ["oranje", "groen", "rood"]  
verwijderde_kleur = lievelingskleuren.pop()  
print(lievelingskleuren) #['oranje', 'groen']  
print(verwijderde_kleur) #rood  
andere_verwijderde_kleur = lievelingskleuren.pop(0)  
print(andere_verwijderde_kleur) #oranje
```

- De pop()-method verwijdert standaard het laatste item in een list
- Geef je tussen de haakjes de index van een element mee dan verwijder je een specifiek element
- Deze method laat toe met dit element te werken na de verwijdering. Heb je geen plannen om dit te doen, kies dan het del statement

# Een item verwijderen op waarde met de remove()-method

```
lievelingskleuren = ["oranje", "groen", "rood", "paars"]  
lievelingskleuren.remove("groen")  
print(lievelingskleuren) #['oranje', 'rood', 'paars']
```

- Ken je de positie van een element niet, maar enkel de waarde? Dan kan je gebruik maken van de remove()-method
- De remove()-method verwijdert enkel het eerste element in de list met de betreffende waarde
- Als de waarde vaker kan voorkomen, dan is een loop noodzakelijk

# Een list ordenen

Python geeft ons verschillende mogelijkheden om list-waarden in een bepaalde volgorde te presenteren



# Een list ordenen met de sort()-method

```
eighties_popmuziek = ["Prince", "Abba", "Sade", "Madonna"]  
eighties_popmuziek.sort()  
print(eighties_popmuziek) #['Abba', 'Madonna', 'Prince', 'Sade']  
eighties_popmuziek.sort(reverse=True)  
print(eighties_popmuziek) #['Sade', 'Prince', 'Madonna', 'Abba']
```

- De sort()-method maakt alfabetisch sorteren eenvoudig
- Let wel: de volgorde van de list wijzigt permanent, dus de oorspronkelijke volgorde gaat verloren
- We kunnen ook in omgekeerde alfabetische volgorde sorteren met het argument reverse=True

# Een list ordenen met de sorted()-functie

```
eighties_popmuziek = ["Prince", "Abba", "Sade", "Madonna"]  
print(sorted(eighties_popmuziek)) #['Abba', 'Madonna', 'Prince', 'Sade']  
print(eighties_popmuziek) #['Prince', 'Abba', 'Sade', 'Madonna']
```

- Om een lijst te sorteren zonder de oorspronkelijke volgorde verloren te laten gaan kan je de sorted() functie gebruiken
- Ook deze functie accepteert een reverse=True argument om omgekeerd alfabetisch te ordenen

# Een list "omdraaien" met de reverse()-method

```
eighties_popmuziek = ["Prince", "Abba", "Sade", "Madonna"]  
eighties_popmuziek.reverse()  
print(eighties_popmuziek)
```

- Het omdraaien van de oorspronkelijke volgorde van een list doe je door gebruik van de reverse()-method
- De oorspronkelijke volgorde gaat verloren, maar je kan deze ongedaan maken door de reverse()-method nogmaals toe te passen

# De lengte van een list bepalen met de len()-functie

```
eighties_popmuziek = ["Prince", "Abba", "Sade", "Madonna"]  
print(len(eighties_popmuziek)) #4
```

Gebruik de len()-functie om het aantal elementen in een list te bepalen

**Een list doorlopen**

# Een list doorlopen

```
lievelingskleuren = ["oranje", "groen", "rood", "paars"]  
for kleur in lievelingskleuren:  
    print(f"Nog een mooie kleur: {kleur.title()}!")
```

- Vaak willen we een list doorlopen om op de elementen een bepaalde taak (of taken) te verrichten
- We kunnen hiervoor een for-loop gebruiken
- Gebruik van een enkelvoud en meervoud-naamgeving ("for item in lijst\_van\_items") maakt je code beter leesbaar

# Numerieke lists maken met de range()-functie

```
for waarde in range(1, 10):  
    print(waarde)
```

```
getallen = list(range(1, 10))  
print(getallen) #[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

- Lists zijn bijzonder geschikt om reeksen getallen op te slaan
- De range()-functie maakt het gemakkelijk om een reeks getallen te genereren
- Het bovenste voorbeeld zal de getallen van 1 tot en met 9 afdrukken
- Je kan het resultaat van range() onmiddellijk capteren in een list door gebruik te maken van een list()-functie

# Numerieke lists maken met range()

```
even_getallen = list(range(2, 15, 2))  
print(even_getallen) #[2, 4, 6, 8, 10, 12, 14]
```

```
kwadraten = []  
for waarde in range(1,10):  
    kwadraten.append(waarde**2)  
print(kwadraten) #[1, 4, 9, 16, 25, 36, 49, 64, 81]
```

- We kunnen de range()-functie ook een derde argument meegeven
- Python gebruikt die waarde als een stapgrootte bij het genereren van getallen
- Met de range()-functie kan bijna elke denkbare set van getallen worden gecreëerd



# Handige functies bij het werken met numerieke lists

```
getallenreeks = [0, 0, 1, 0, 2, 0, 2, 2, 1, 6, 0, 5]  
min(getallenreeks) #0  
max(getallenreeks) #9  
sum(digits)
```

Met behulp van deze functies kan je heel eenvoudig het minimum, maximum of de som van een numerieke list vinden

# Verkorte notatie

```
kwadraten = [waarde**2 for waarde in range(1, 10)]
```

- In bovenstaande code combineer je de for loop en de creatie van nieuwe elementen in één en dezelfde regel code, en voeg je deze automatisch toe aan een lijst
- We noemen dit list comprehensions

## Werken met delen van een list

# Een list "slicen"

```
lievelingskleuren = ["oranje", "groen", "rood", "paars"]
print(lievelingskleuren[0:3]) #['oranje', 'groen', 'rood']
print(lievelingskleuren[1:4]) #['groen', 'rood', 'paars']
print(lievelingskleuren[:4]) #['oranje', 'groen', 'rood', 'paars']
print(lievelingskleuren[2:]) #['rood', 'paars']
print(lievelingskleuren[-3:]) #['groen', 'rood', 'paars']
```

- In Python verwijzen we naar een specifieke groep elementen in een list met de term "slice"
- Een slice maak je door de index van het eerste en het laatste element waarmee je wil werken mee te geven
- Net zoals bij range() stopt Python één element voor de tweede index die je hebt meegegeven
- Laat je de eerste index in een slice weg dan wordt automatisch gestart bij het eerste element van de list

# Een list "slicen"

- Op gelijkaardige wijze wordt tot het einde van de list gegaan als je de tweede index weglaat
- Een negatieve index verwijst naar een element op zoveel posities van het einde van een list
- Een derde waarde geeft mee hoeveel elementen telkens mogen worden overgeslagen

# Een slice van een list doorlopen

```
lievelingskleuren = ["oranje", "groen", "rood", "paars"]  
for kleur in lievelingskleuren[:3]:  
    print(kleur) # oranje groen rood
```

- Met een for-loop kan je een slice doorlopen

# Een kopie maken van een list

```
lievelingskleuren = ["oranje", "groen", "rood", "paars"]
lievelingskleuren_kopie = lievelingskleuren[:]
lievelingskleuren.append("geel")
lievelingskleuren_kopie.append("magenta")
print(lievelingskleuren) #['oranje', 'groen', 'rood', 'paars', 'geel']
print(lievelingskleuren_kopie) #['oranje', 'groen', 'rood', 'paars', 'magenta']
```

Om een list te kopiëren maak je een slice die de oorspronkelijke list bevat door beide indexen weg te laten ([:])

# Tuples



# Tuples

```
mijn_tuple = (100, 30, 60)
print(mijn_tuple[0]) # 100
print(mijn_tuple[1]) # 30
```

- Een tuple is een list met het verschil dat een tuple onveranderlijk (of immutable) is
- Gebruik tuples wanneer je een reeks van waarden wil opslaan die niet veranderen tijdens de levenscyclus van het programma
- Ziet er exact uit als een list, maar wordt weergegeven met ronde haakjes
- Definieer je een tuple met één enkel element dan moet je een komma gebruiken na het eerste en enige element: `mijn_tuple = (5,)`

# Een tuple overschrijven

```
mijn_tuple = (100, 30, 60)
mijn_tuple = (50, 30)
for getal in mijn_tuple:
    print(getal)
```

- Een tuple wijzigen gaat niet
- Je kan evenwel een nieuwe waarde toekennen aan de variabele die de tuple vertegenwoordigt
- Je kan een tuple doorlopen net zoals elke andere list

# Python Programming - les 2 -

[kristof.michiels01@ap.be](mailto:kristof.michiels01@ap.be)