Python OOP

2/ Snel van start met Python

Kristof Michiels

Het doel van deze les?

- Python in een notendop. De absolute basis van Python bekijken om snel onze eerste programma's te kunnen schrijven.
- Dat geeft motivatie en zin om verder te doen. In de komende lessen behandelen we deze zaken in groter detail én we dringen dieper door in het Python universum.
- Dit helikopter-uitzicht is het begin van je reis naar het kunnen oplossen van complexe problemen met Python. Welke je afstudeerrichting ook is - AI, Cyber Security, Cloud of Internet of Things - Python heeft oplossingen

Inhoud

- Variabelen (en hun datatypes)
- Operators en expressies
- Datastructuren
- Control flow
- Functies
- Klassen en objecten
- Input van de gebruiker

Variabelen (en hun datatypes)

Variabelen

```
getal = 12
favoriete_taal = "Python"
print(mijn_getal)
```

- Zijn de basisbouwstenen van je programma's: eenheden die een naam en een waarde toegewezen krijgen
- Ze worden automatisch gemaakt wanneer ze voor het eerst worden toegewezen
- Het gelijkheidsteken noemen we een toekenningsoperator
- Als waarde zijn verschillende data types mogelijk: tekst, getallen, booleans,...
- De waarde kan gewijzigd worden tijdens looptijd van je programma

Naamgeving variabelen

```
1_getal = 12 # mag niet, geeft syntax error
getal_1 = 12 # kan wel
mijn_school = "Artesis Plantijn Hogeschool"
x = 2 # waarvoor staat x ? Probeer meer betekenis aan de naam te geven
```

- Mag niet beginnen met een getal
- Mag geen speciale karakters bevatten. Enkel underscores
- Best practice is om enkel kleine letters te gebruiken. Nooit starten met een hoofdletter (doen we bij klassen)
- Geef betekenisvolle namen aan je variabelen!

Optioneel: meerdere toewijzigen

```
voornaam, familienaam, beroep = "Kristof", "Michiels", "Docent"
```

- Je kan waarden aan meerdere variabelen toekennen in één enkele statement
- Dit maakt je programma's korter en beter leesbaar
- Je scheidt hiervoor de namen van de variabelen met komma's en doet hetzelfde met de waarden
- Python kent dan de respectievelijke waarde toe aan elke genoemde variabele

Constanten

```
STUDENTEN_PER_GROEP = 4
```

- Een constante is zoals een variabele, maar met een waarde die niet verandert tijdens de looptijd van je programma
- Python heeft geen ingebouwde ondersteuning voor constanten maar ontwikkelaars gebruiken een naam bestaande uit enkel hoofdletters om aan te geven dat het hier over een niet te veranderen constante gaat

Datatypes (ook: gegevenstypes)

```
mijn_school = "Artesis Plantijn Hogeschool"
getal = 12
print(type(mijn_school)) # str
print(type(getal)) # int
```

- Datatypes zijn de verschillende soorten data die we aan variabelen kunnen toekennen
- Op basis van die data kiest Python zelf het datatype
- De waarde wordt dan een object van die klasse. Daar hoort bepaalde functionaliteit bij
- We zagen in de voorbeelden reeds tekst en (gehele) getallen, maar Python kent er een pak meer
- De Python-functie type geeft op basis van de naam van je variabele het datatype terug

Getallen: integers en floats

```
dag = 21
temperatuur = -15
gewicht = 75.5
stand_bankrekening = -260.3
```

- Integers zijn gehele getallen. Dit zijn positieve of negatieve getallen zonder komma
- Floats zijn decimale getallen. Positieve of negatieve getallen met een komma
- Float wordt binair opgeslagen in een vast aantal bits = zwevendekommagetal m.a.w. de komma zweeft
- We zien verder bij operators en expressies nog wat je allemaal met getallen kunt doen

Tekst: strings

```
boodschap = "Dit is een string."
boodschap = 'Dit is ook een string.'
boodschap = 'Mijn vriend vroeg: "Programmeer jij ook in Python?"'
boodschap = "Ik hou van het schrijven van Python programma's"
```

- Een string bestaat uit een reeks tekst-karakters
- Alles binnen aanhalingstekens wordt beschouwd als een string
- Je mag gebruik maken van enkele of dubbele aanhalingstekens
- Deze flexibiliteit laat toe om aanhalings- en afkappingstekens te gebruiken in je strings
- Wees standvastig in je keuzes

Tekst: strings

```
zin = "\"Deze piano\'s klinken fantastisch!\" zegt de muziekliefhebber enthousiast."
```

- Wat doen wanneer je een mix hebt van enkele en dubbele aanhalingstekens binnen je string? Je kan de aanhalingstekens binnen de string escapen met een backslash (\)
- Waarom de naam string? Een reeks van tekens. Elk karakter krijgt individuele geheugenruimte en Python reigt ze aan elkaar.

Acties op strings: hoofdlettergebruik wijzigen

```
naam = "Guido van Rossum"
print(naam.title()) # Elk woord laten beginnen met een hoofdletter: Guido Van Rossum
print(naam.upper()) # Alles in hoofdletters: GUIDO VAN ROSSUM
print(naam.lower()) # Alles in kleine letters: guido van rossum
```

- Met een string-object komt extra functionaliteit: functies (methods) die we op onze waarde kunnen toepassen
- De dot-notatie (".") vertelt Python in het eerste vb. om de title-method uit te voeren op de variabele naam
- Elke method wordt gevolgd door haakjes, omdat methods vaak bijkomende informatie nodig hebben om hun werk te kunnen doen. Hier is dit evenwel niet het geval
- De lower()-method wordt vaak gebruikt om data op te slaan die door een gebruiker werd ingegeven

f-strings: variabelen gebruiken in een string

```
voornaam = "Guido"
familienaam = "van Rossum"
volledige_naam = f"{voornaam} {familienaam}"
boodschap = f"Bedankt voor Python, {volledige_naam.title()}!"
print(boodschap)
```

- Om variabelen te gebruiken binnen strings plaats je de letter f onmiddellijk voor het eerste aanhalingsteken
- Je omringt de variabelen die je in de string gebruikt met accolades
- Python vervangt elke variabele door de waarde
- De f in f-strings staat voor "format"

Strings: witruimte toevoegen

```
print("\tHelderheid")
print("Pythonisch programmeren:\nHelder\nGeloofwaardig\nEfficiënt")
print("Pythonisch programmeren:\n\tHelder\n\tGeloofwaardig\n\tEfficiënt")
```

- Met witruimte doelen we op niet-afdrukbare tekens als spaties, tabs en einde-lijnsymbolen
- Je gebruikt ze om je output op een beter leesbare manier te organiseren
- Een tab-insprong toevoegen aan je tekst doe je met "\t"
- Een nieuwe regel voeg je toe met "\n"
- Combinaties zijn mogelijk: "\n\t" zorgt ervoor dat Python op een nieuwe regel begint, met een tabinsprong

Strings: witruimte verwijderen

```
boodschap = " Ik hou van Python "
print(boodschap.rstrip())
print(boodschap.lstrip())
print(boodschap.strip())
```

- Python maakt het eenvoudig om (eventueel) aanwezige witruimte te verwijderen
- Er kan een onderscheid worden gemaakt tussen witruimte links, rechts of aan beide zijden van een string
- Handig als je twee strings met elkaar wil vergelijken

"Waar" of "Niet waar": booleans

```
licht_is_aan = False
ik_hou_van_python = True

if licht_is_aan:
    print("Het licht is aan!")
else:
    print("Het licht is uit!")
```

- Booleans zijn de waarden True en False (let op de hoofdletters T en F)
- We verkrijgen ze meestal als het resultaat van een vergelijkende expressie (zie volgend hoofdstukje)
- Ze zijn op die manier bijzonder handig in het bepalen van de control flow van een toepassing

Operators en expressies

Operators en expressies

- Een operator is een code-element dat een bewerking uitvoert op een of meer code-elementen die waarden bevatten
- Een expressie is een reeks waarde-elementen in combinatie met operators, wat een nieuwe waarde oplevert. De operators handelen op de waarde-elementen door berekeningen, vergelijkingen of andere bewerkingen uit te voeren.
- Er bestaan verschillende soorten operators en dus ook expressies

Wiskundige operators

```
print(1 + 1)
print((-2 + 3) * -1) # -1
print(4 * 5)
print(5 ** 2)
print(20 / 5) # 4.0 een float
print(13 % 4) # 1
```

- Standaardregels van rekenkundige voorrang zijn van toepassing
- Meest voor de hand liggende type. Je kent ze uit de wiskunde
- Optellen, aftrekken, vermenigvuldigen (*), delen (/), machtsverheffing (**)
- Een speciale operator is de modulus-operator: de rest bij deling door (%). We gebruiken deze vaak bij het schrijven van onze programma's

Wiskundige operators

```
print(9 / 3) # 3.0
print(13 + 2.0) # 15.0
print(4 * 5.0) # 20.0
print(2.0 ** 3) # 8.0
totaal_aantal_aardbewoners = 7_902_193_151
print(totaal_aantal_aardbewoners) # 7902193151
print(9 // 4) # 2
```

- Wanneer je twee getallen deelt is het resultaat altijd een float
- Python gebruikt standaard een float als resultaat voor elke bewerking die een float bevat
- Bij grote getallen kun je cijfers groeperen met underscores om ze leesbaarder te maken
- Wil je enkel het gehele gedeelte terugkrijgen als geheel getal, gebruik dan de //-operator

Wiskundige operators met strings

```
"string1" + "string2"
"string1" * 4
"1" + 4 # error
```

- We kunnen strings aaneen plakken door ze "bij elkaar op te tellen". In het Engels: concatenation.
- We kunnen strings met elkaar "vermenigvuldigen"
- Een string + een getal geven een foutmelding

Vergelijkende operators

```
print(4 == 5) # False
print(4 < 5) # True
print(5 >= 2) # True
print(5 <= 5) # True
print(5 > 2) # True
```

- Het resultaat van een vergelijkende expressie met deze operators is een boolean
- We kunnen deze resultaten gebruiken in beslissingstructuren en loops (zie verder)
- Willen we twee waarden met elkaar vergelijken, dan gebruiken we een dubbel gelijkheidsteken

Logische operators

```
print(True and True) # True
print(True and False) # False
print(True or False) # True
print(False or False) # False
print(not True)
print(3 < 5 and 6 > 2) # True
```

- Het betreft hier de Engelse termen and, or en not
- Ze laten toe om gecombineerde expressies - die resulteren in een boolean - te evalueren
- De not-operator keert de boolean om: True wordt False en omgekeerd

Identiteitsoperators

```
string_1 = "Aap"
string_2 = "Noot"
string_3 = string_1
print(string_1 is string_2) # False
print(string_1 is not string_2) # True
print(string_1 is string_3) # True
```

- Identiteitsoperators (identity operators) worden gebruikt om objecten met elkaar te vergelijken
- Niet om te zien of ze gelijkwaardig zijn, maar of ze hetzelfde object zijn, met dezelfde geheugenlocatie
- Is: als de expressie True teruggeeft zijn x en y hetzelfde object
- Is not: als de expressie True teruggeeft zijn x en y niet hetzelfde object

Lidmaatschap operators

```
print(1 in [1,2,3,4,5]) # True
print(10 not in [1,2,3,4,5]) # True
print("Python" in "Mijn favoriete programmeertaal is Python") # True
```

- Lidmaatschap operators (membership operators) worden gebruikt om te evalueren of een reeks aanwezig is in een object
- In: is de reeks aanwezig in het object
- Not in: is de reeks niet aanwezig in het object

Datastructuren

Datastructuren

- De datatypes die we tot nog toe hebben gezien vertegenwoordigen telkens één enkele waarde, zoals 3.11, "Python" of True
- We beschikken in Python ook over datatypes die het vasthouden van een reeks waarden in één variabele mogelijk maken
- We spreken hier over datastructuren en de meest gebruikte zijn de vier die hier kort zullen worden besproken: lists, sets, tuples en dictionaries
- Ze zijn supernuttig bij het schrijven van programma's en worden dan ook heel vaak gebruikt door developers
- Ze worden hier heel kort overlopen maar no worries: ze krijgen later hun eigen les ;-)

Lists

```
mijn_list = [3.11, "Python", False]
print(len(mijn_list)) # 3
print(mijn_list[1]) # "Python"
```

- Lists worden geschreven met vierkante haakjes en bevatten een aantal door komma's gescheiden waarden
- Deze waarden kunnen alle datatypes zijn die we tot nu toe hebben gezien
- Je kan dus een list van strings hebben, een list met strings en integers, ... tot en met lists die lists als elementen hebben
- List objecten hebben een handige length-functie (len) die het aantal elementen in de list teruggeeft
- De volgorde van elementen in een lijst is belangrijk want we roepen deze op via het indexgetal

Sets

```
mijn_set = {"C#", "JavaScript", "C++"}
print(len(mijn_set)) # 3
```

- Sets zijn verwant met lists, met dat verschil dat alle elementen uniek moeten zijn
- Je beschrijft een set met accolades
- Je kan ook bij set objecten de length functie gebruiken
- De volgorde binnen een set is niet van belang

Tuples

```
mijn_tuple = ("IoT", "AI", "CSC")
print(len(mijn_tuple)) # 3
```

- Tuples worden geschreven met haakjes en lijken sterk op lists
- De volgorde van elementen in een lijst is belangrijk en ook hier beschikken we over de length-functie
- Het verschil met lists is dat eens een tuple is gecreëerd je geen nieuwe elementen meer kan toevoegen
- Waarom gebruiken programmeurs indien geschikt tuples i.p.v. lists: geheugenefficiëntie

Dictionaries

```
mijn_dict = {
    "programmeertaal": "Python",
    "jaar_van_ontstaan": 1991,
    "huidige_versie": 3.11
}
print(mijn_dict["programmeertaal"])
```

- Dictionaries kan je vergelijken met een woordenboek. Je kan een woord opzoeken en je krijgt een definitie
- We gebruiken accolades aan de buitenzijde en binnenin spreken we over key-value-paren
- De sleutels binnen een dictionary moeten uniek zijn. De volgorde van de elementen is niet van belang
- Dictionaries zijn een beetje als sets: unieke waarden, volgorde niet van belang, accolades als notatie

Control Flow

Control Flow

- Met control flow-elementen neem je de volgorde waarin de instructies van je programma wordt uitgevoerd in handen
- De elementen die je hier tot je beschikking hebt zijn beslissingsstructuren, loops en het aanroepen van functies

If/else beslissingsstructuren

```
voorwaarde = True # elke expressie die een True of False teruggeeft
if voorwaarde:
    print("De voorwaarde was waar!")
    print("Ik behoor ook tot de if-structuur")
print("Ik behoor niet meer tot de if-structuur")
```

```
if voorwaarde:
    print("De voorwaarde was waar")
else:
    print("De voorwaarde was niet waar")
```

Beslissingsstructuren laten toe code uit te voeren afhankelijk v.h. resultaat van de voorwaarde: True of False

If/else beslissingsstructuren

```
voorwaarde_1 = True
voorwaarde_2 = False
if voorwaarde_1:
    print("De eerste voorwaarde is waar")
    if voorwaarde_2:
        print("Voorwaarde één én twee zijn beiden waar")
else:
    print("De eerste voorwaarde is niet waar")
```

If-structuren kunnen in elkaar verweven (=genest) worden, zoveel als nodig

For loops

```
gemeten_temperaturen = [14,12,15,9]
for meting in gemeten_temperaturen:
    print(meting)
```

- For loops doorlopen wat we in Python iterables noemen. Een list is bijvoorbeeld zo'n iterable
- Binnen een for loop kan je dus voor elk element in een lijst een bepaalde actie beschrijven
- Wanneer de voorwaarden tijdens het doorlopen van de iterable moeten kunnen wijzigen gebruik je beter een while-loop
- Is niet dezelfde als de in-operator die we eerder tegenkwamen

While loops

```
teller = 1
while teller < 10:
    print(teller)
    teller = teller + 1
```

- While loops gebruik je om één of meerdere statements een aantal keer te laten uitvoeren, zo lang de voorwaarde die je aan de loop meegeeft True is. Wordt de voorwaarde False, dan stopt de loop
- Je moet zorgen dat in de statements binnen de loop een mechanisme aanwezig is dat de voorwaarde na verloop van tijd doet veranderen in False. Doe je dit niet dan beland je in wat we noemen een infinite loop. De loop gaat dan continue door en je programma blijft vastzitten

Functies

Functies

```
print("Print is een Python functie")

def vermenigvuldig_met_zes(waarde):
    return 6 * waarde

print(vermenigvuldig_met_zes(4))
```

- We hebben al wat functies gezien zoals print() of len(). Deze functie worden voorzien door de Python interpreter
- Maar we kunnen ook onze eigen functies schrijven
- Dit zijn handige verzamelingen statements die kunnen uitgevoerd worden telkens de functie wordt aangeroepen

Functies

```
def vermenigvuldig_waardes(waarde_1, waarde_2):
    return waarde_1 * waarde_2

print(vermenigvuldig_waardes(3, 5)) # 15

def voeg_element_toe_aan_lijst(mijn_lijst, element):
    mijn_lijst.append(element)

voeg_element_toe_aan_lijst([1,2,3],4)
```

- Voor functies gebruiken we het keyword def. Een functie-declaratie eindigt op een ":"
- De naamgeving voor functies is dezelfde als die voor variabelen
- Binnen de haakjes na de functienaam kan je geen, één of meerdere argumenten meegeven

Functies en het None type

```
print(print("Hallo wereld")) # None
type(None) # NoneType
```

- We gebruiken variabelennamen als parameter en deze namen kan je binnen de functie gaan gebruiken. Ze vertegenwoordigen daar de waarden die al argument worden meegegeven telkens de functie wordt aangeroepen
- Functies kunnen een waarde teruggeven (met return), maar dat is niet verplicht
- Een functie zonder return statement geeft None terug
- None is een bijzonder Python keyword zoals null of undefined in andere programmeertalen. Het vertegenwoordigt de afwezigheid van een bepaalde waarde

None

- None is een speciaal basisgegevenstype dat een enkel speciaal gegevensobject definieert met de naam None
- Wordt gebruikt om een lege waarde weer te geven. Vaak gebruikt als een tijdelijke aanduiding om een punt in een gegevensstructuur aan te geven waar uiteindelijk betekenisvolle gegevens zullen worden gevonden
- Je kan eenvoudig testen op de aanwezigheid van None, omdat er slechts één instantie van None is in het hele Python-systeem (alle verwijzingen naar None verwijzen naar hetzelfde object) en None is alleen gelijk aan zichzelf

```
var = None
if var is None:
    print("None")
else:
    print("Niet None")
```

Klassen en objecten

Klassen en objecten

```
class Kat():
    def __init__(self, naam):
        self.naam = naam
        self.poten = 4

    def spreek(self):
        print(f"{self.naam} zegt 'Miauw'!")

een_kat_object = Kat("Felix")
een_andere_kat_object = Kat("Minnie")
een_kat_object.spreek() # Felix zegt 'Miauw'!
```

Klassen en objecten zijn de ingrediënten van Objectgeoriënteerd programmeren (= de OOP in ons vak)

Klassen en objecten

- OOP is een werkwijze om de structuur logisch en overzichtelijk te houden
- Gebruik van klassen maakt het mogelijk om honderden variabelen en functies overzichtelijk in georganiseerde en van heldere naamgeving voorziene structuren onder te brengen
- Een klasse noemen we de blauwdruk/het plan
- Het object noemen we de instance of een afdruk van die blauwdruk
- De init-functie noemen we de initialisatiefunctie. Ze wordt aangeroepen telkens een nieuw object gecreëerd wordt
- self verwijst naar de specifieke instance
- De naam van een klasse laten we starten met een hoofdletter

Input van de gebruiker

Input van de gebruiker

- Je kan de functie input() gebruiken om invoer van de gebruiker te krijgen
- Gebruik de promptstring die u aan de gebruiker wilt tonen als invoerparameter:

```
naam = input("Naam? ")
print(naam)
leeftijd = int(input("Leeftijd? "))
print(leeftijd)
```

Input van de gebruiker

- Nadeel is dat de invoer binnenkomt als een string, dus als je het als een getal wilt gebruiken, moet je de functie int() of float() gebruiken om de string om te zetten naar een getal

```
basis = float(input("Basis in cm? "))
hoogte = float(input("Hoogte in cm? "))
oppervlakte = basis * hoogte
print(f"De oppervlakte van de rechthoek is {oppervlakte}cm2")
```

Python OOP - Snel van start met Python

kristof.michiels01@ap.be