Inleiding Python: deel 2

L. Espeel
(bron: W. Verstraete)

vives University of Applied Sciences

1 Beslissingsstructuren
2 Lussen
3 Computerlogica
4 Functies
5 Functies met parameters
6 Lists
7 Overige collections

Beslissingsstructuren

1

Beslissingsstructuren

Vergelijking: gelijkheidsoperator

Vraag: zijn twee waarden gelijk?

Om deze vraag te stellen, gebruik je de == (gelijk aan) operator.

Vergeet dit belangrijke onderscheid niet:

= is een toewijzingsoperator, bijv. a = b wijst a toe met de waarde van b;

== is de vraag of deze waarden gelijk zijn? dus a == b vergelijkt a en b.

Het is een binaire operator met linkszijdige binding. Het heeft twee argumenten nodig en controleert of ze gelijk zijn.

Beslissingsstructuren

Vergelijking: gelijkheidsoperator

Vb:

Wat is het resultaat van de volgende vergelijking?

```
2 == 2
2 == 2.
1 == 2
```


Beslissingsstructuren

Vergelijking: gelijkheidsoperator

```
black_sheep == 2 * white_sheep

black_sheep == (2 * white_sheep)

var = 0  # Assigning 0 to var
print(var == 0)

var = 1  # Assigning 1 to var
print(var == 0)
```

Beslissingsstructuren

Vergelijking: ongelijkheid

De != (niet gelijk aan) operator vergelijkt ook de waarden van twee operanden.

Dit is het verschil: als ze gelijk zijn, is het resultaat van de vergelijking False. Als ze niet gelijk zijn, is het resultaat van de vergelijking True.

```
var = 0  # Assigning 0 to var
print(var != 0)

var = 1  # Assigning 1 to var
print(var != 0)
```


Beslissingsstructuren

Vergelijking: groter, kleiner dan

```
<, <=, >, >=
```

De operatoren (strikt en niet-strikt) zijn binaire operatoren met linkszijdige binding en hun prioriteit is groter dan == en !=

```
answer = number_of_lions >= number_of_lionesses
```

Beslissingsstructuren

Oefening: variabelen - vragen en antwoorden

Schrijf met behulp van een van de vergelijkingsoperatoren in Python een eenvoudig programma van twee regels dat de parameter n als invoer neemt, wat een geheel getal is, en False afdrukt als n kleiner is dan 100, en True als n groter is dan of gelijk is aan 100.

Maak nu geen gebruik van if-blokken (we gaan er binnenkort over praten).

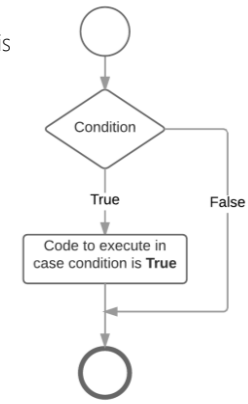
Beslissingsstructuren

Voorwaarden en voorwaardelijke uitvoering

Je weet al hoe je Python-vragen moet stellen, maar je weet nog steeds niet hoe je de antwoorden redelijk kunt gebruiken.

Je moet een mechanisme hebben waarmee je iets kunt doen als aan een voorwaarde is voldaan, en het niet kunt doen als dat niet het geval is.

```
if true_or_not:
    do_this_if_true
```



Beslissingsstructuren

Voorwaarden en voorwaardelijke uitvoering

```
if sheep_counter >= 120:
    make_a_bed()
    take_a_shower()
    sleep_and_dream()
feed_the_sheepdogs()
```

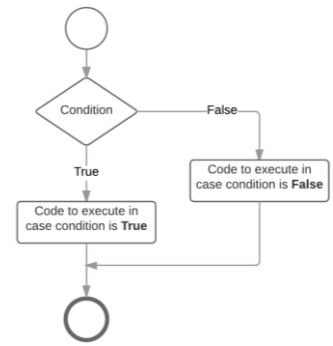
Zoals je kunt zien, worden het opmaken van een bed, het nemen van een douche en in slaap vallen en dromen allemaal voorwaardelijk uitgevoerd - wanneer sheep_counter de gewenste limiet bereikt.

Het voeren van de herdershonden wordt echter altijd gedaan (d.w.z. de functie feed_the_sheepdogs() is niet ingesprongen en behoort niet tot het if-blok, wat betekent dat deze altijd wordt uitgevoerd.)

Beslissingsstructuren

Voorwaardelijke uitvoering: de if-else-instructie

```
if true_or_false_condition:
    perform_if_condition_true
else:
    perform_if_condition_false
```



Er is dus een nieuw woord: else - dit is een keyword.

De if-else uitvoering gaat als volgt:

- als de voorwaarde True oplevert (de waarde is niet gelijk aan nul), wordt de instructie perform_if_condition_true uitgevoerd en komt er een einde aan de voorwaardelijke instructie;
- als de voorwaarde resulteert in False (deze is gelijk aan nul), wordt de instructie perform_if_condition_false uitgevoerd en wordt de voorwaardelijke instructie beëindigd.

Beslissingsstructuren

Geneste if-else-statements

```
if the_weather_is_good:
    if nice_restaurant_is_found:
        have_lunch()
    else:
        eat_a_sandwich()
else:
    if tickets_are_available:
        go_to_the_theater()
    else:
        go_shopping()
```

Hier zijn twee belangrijke punten:

- dit gebruik van de if-opdracht staat bekend als nesten; onthoud dat al het andere verwijst naar de if die op hetzelfde inspongniveau ligt; je moet dit weten om te bepalen hoe de if's en else's bij elkaar horen;
- overweeg hoe de insprong de leesbaarheid verbetert en de code gemakkelijker te begrijpen en te debuggen maakt.

Beslissingsstructuren

elif-statements

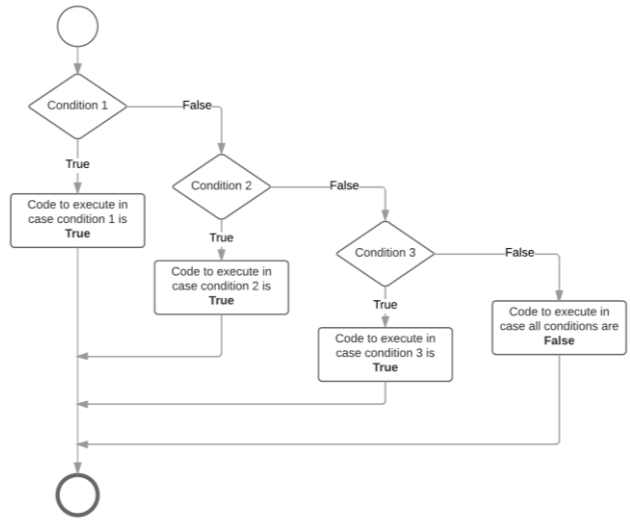
```
if the_weather_is_good:
    go_for_a_walk()
elif tickets_are_available:
    go_to_the_theater()
elif table_is_available:
    go_for_lunch()
else:
    play_chess_at_home()
```

Zoals je waarschijnlijk vermoedt, is dit een kortere vorm van else if.

- je mag else niet gebruiken zonder voorafgaande if;
- else is altijd de laatste tak van de cascade, ongeacht of je elif hebt gebruikt of niet;
- else is een optioneel onderdeel van de cascade en kan worden weggelaten;
- als er een else-tak in de cascade is, wordt slechts één van alle takken uitgevoerd;
- als er geen else branch is, is het mogelijk dat geen van de beschikbare branches wordt uitgevoerd.

Beslissingsstructuren

elif-statements



Beslissingsstructuren

elif-statements

Vb:

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

# Choose the larger number
if number1 > number2:
    larger_number = number1
else:
    larger_number = number2

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

Beslissingsstructuren

elif-statements

Vb:

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

# Choose the larger number
if number1 > number2: larger_number = number1
else: larger_number = number2

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


Beslissingsstructuren

elif-statements

Vb:

```
# Read three numbers
number1 = int(input("Enter the first number: "))
number2 = int(input("Enter the second number: "))
number3 = int(input("Enter the third number: "))

# We temporarily assume that the first number
# is the largest one.
# We will verify this soon.
largest_number = number1
```

Beslissingsstructuren

elif-statements

```
# We check if the second number is larger than the current largest_number
# and update the largest_number if needed.
if number2 > largest_number:
    largest_number = number2

# We check if the third number is larger than the current largest_number
# and update the largest_number if needed.
if number3 > largest_number:
    largest_number = number3

# Print the result
print("The largest number is:", largest_number)
```


Beslissingsstructuren

Pseudocode en inleiding tot lussen

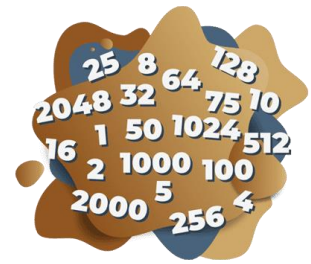
Je zou nu in staat moeten zijn om een programma te schrijven dat de grootste van vier, vijf, zes of zelfs tien getallen vindt.

Je kent het schema al, dus het vergroten van de omvang van het probleem zal niet bijzonder ingewikkeld zijn.

Maar wat gebeurt er als we je vragen een programma te schrijven dat de grootste van tweehonderd getallen vindt? Kun je je de code voorstellen?

Je hebt tweehonderd variabelen nodig. Als tweehonderd variabelen niet erg genoeg is, probeer je dan voor te stellen dat je zoekt naar de grootste van een miljoen getallen.

Stel je een code voor die 199 voorwaardelijke instructies en tweehonderd aanroepen van de functie input() bevat. Gelukkig hoef je daar niet mee om te gaan. Er is een eenvoudigere aanpak d.m.v. lussen.



Beslissingsstructuren

Oefening: vergelijkingsoperatoren en voorwaardelijke uitvoering

Spathiphyllum, beter bekend als vredeslelie of vaantjesplant, is een van de meest populaire kamerplanten binnenshuis die schadelijke gifstoffen uit de lucht filtert. Enkele van de gifstoffen die het neutraliseert, zijn benzeen, formaldehyde en ammoniak.

Stel je voor dat je computerprogramma dol is op deze planten. Telkens wanneer het een invoer ontvangt in de vorm van het woord Spathiphyllum, roept het onwillekeurig de volgende string naar de console: "Spathiphyllum is de beste plant ooit!"

Schrijf een programma dat het concept van voorwaardelijke uitvoering gebruikt, een tekenreeks als invoer neemt en:

- drukt de zin "*Ja - Spathiphyllum is de beste plant ever*!" naar het scherm als de ingevoerde tekenreeks "Spathiphyllum" is (hoofdletters)
- print "*Nee, ik wil een grote Spathiphyllum!*" als de ingevoerde tekenreeks "spathiphyllum" (kleine letters) is
- print "*Spathiphyllum! Niet [invoer]!*" anders. Opmerking: [invoer] is de tekenreeks die als invoer wordt gebruikt.

Beslissingsstructuren

Oefening: if-elif-else statement

Zoals je ongetwijfeld weet, kunnen jaren vanwege een aantal astronomische redenen schrikkeljaren of gewone jaren zijn. De eerste zijn 366 dagen lang, terwijl de laatste 365 dagen lang zijn.

Sinds de introductie van de Gregoriaanse kalender (in 1582) wordt de volgende regel gebruikt om het soort jaar te bepalen:

- als het jaartal niet deelbaar is door vier, is het een gewoon jaartal;
- anders, als het jaartal niet deelbaar is door 100, is het een schrikkeljaar;
- anders, als het jaartal niet deelbaar is door 400, is het een gewoon jaartal;
- anders is het een schrikkeljaar.

De code moet een van de twee mogelijke berichten uitvoeren, namelijk Schrikkeljaar of Gewoon jaar, afhankelijk van de ingevoerde waarde.
Het zou goed zijn om te verifiëren of het ingevoerde jaar in de Gregoriaanse jaartelling valt, en anders een waarschuwing te geven: Niet binnen de Gregoriaanse kalenderperiode.

Tip: gebruik de operatoren != en %.

Bekijk je uiteindelijke code of je het nog korter zou kunnen schrijven in 1 if-statement (zie straks bij Computerlogica)...

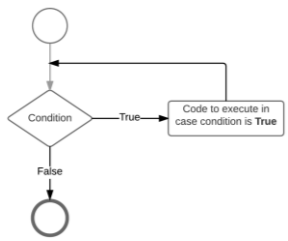
Lussen

2

Lussen

while

```
while conditional_expression:
    instruction_one
    instruction_two
    instruction_three
    :
    :
    instruction_n
```



- als je meer dan één instructie binnen één while-lus wilt uitvoeren, moet je (zoals bij if) alle instructies op dezelfde manier laten inspringen;
- een instructie of reeks instructies die binnen de while-lus wordt uitgevoerd, wordt de body van de lus genoemd;
- als de voorwaarde False is (gelijk aan nul) zodra deze voor de eerste keer wordt getest, wordt de body niet één keer uitgevoerd;
- de body zou in staat moeten zijn om de waarde van de voorwaarde te veranderen, want als de voorwaarde in het begin waar is, kan de body continu uitvoeren.

Lussen

Oneindige lus

```
while True:
    print("I'm stuck inside a loop.")
```

Deze lus zal oneindig "I'm stuck inside a loop." afdrukken op het scherm.

Gebruik crtl-c om eruit te breken in IDLE

Lussen

while

```
# Store the current largest number here.
largest_number = -999999999

# Input the first value.
number = int(input("Enter a number or type -1 to stop: "))

# If the number is not equal to -1, continue.
while number != -1:
    # Is number larger than largest_number?
    if number > largest_number:
        # Yes, update largest_number.
        largest_number = number
    # Input the next number.
    number = int(input("Enter a number or type -1 to stop: "))

# Print the largest number.
print("The largest number is:", largest_number)
```

Analyseer het programma zorgvuldig.
Kijk waar de lus begint (regel 8).
Lokaliseer de body van de lus en ontdek hoe de body wordt verlaten.

Lussen

Teller om de lus te verlaten

```
counter = 5
while counter != 0:
    print("Inside the loop.", counter)
    counter -= 1
print("Outside the loop.", counter)
```

```
counter = 5
while counter:
    print("Inside the loop.", counter)
    counter -= 1
print("Outside the loop.", counter)
```

Voel je niet verplicht om je programma's altijd zo kort en compact mogelijk te coderen. Leesbaarheid kan een belangrijkere factor zijn. Houd je code gereed voor een nieuwe programmeur.

Lussen

Oefening: raad het geheime nummer

Een junior goochelaar heeft een geheim nummer gekozen. Hij heeft het verborgen in een variabele met de naam secret_number. Hij wil dat iedereen die zijn programma uitvoert, het spel "Raad het geheime nummer" speelt en raadt welk nummer hij voor hen heeft gekozen. Degenen die hier niet in slagen, zullen voor altijd vastzitten in een eindeloze lus!

Jouw taak? Dit programma te schrijven. De code :

- zal de gebruiker vragen om een geheel getal in te voeren;
- zal een while-lus gebruiken;
- zal controleren of het nummer dat door de gebruiker is ingevoerd hetzelfde is als het nummer dat door de goochelaar is gekozen.
  - Als het door de gebruiker gekozen nummer anders is dan het geheime nummer van de goochelaar, krijgt de gebruiker het bericht "Ha ha! Je zit vast in mijn lus!" en wordt gevraagd om opnieuw een nummer in te voeren.
  - Als het nummer dat door de gebruiker is ingevoerd overeenkomt met het nummer dat door de goochelaar is gekozen, moet het nummer op het scherm worden afgedrukt en moet de goochelaar de volgende woorden zeggen: "Goed gedaan, Dreuzel! Je bent nu vrij."

De goochelaar rekent op je! Stel hem niet teleur.

Lussen

for

Een ander soort lus die in Python beschikbaar is, komt voort uit de observatie dat het soms belangrijker is om het aantal iteraties van de lus te tellen dan om de voorwaarden te controleren.

```
for i in range(100):
    # do_something()
    pass
```

De functie range() (dit is een zeer speciale functie) is verantwoordelijk voor het genereren van alle gewenste waarden van de controlevariabele; in ons voorbeeld creëert de functie (we kunnen zelfs zeggen dat deze de lus voedt met) volgende waarden uit de volgende set: 0, 1, 2 .. 97, 98, 99; opmerking: in dit geval begint de functie range() zijn werk vanaf 0 en voltooit het één stap (één geheel getal) voor de waarde van zijn argument;

Lussen

for

```
for i in range(2, 8):
    print("The value of i is currently", i)
```

In dit geval bepaalt het eerste argument de initiële (eerste) waarde van de controlevariabele.

Het laatste argument toont de eerste waarde waaraan de controlevariabele niet zal worden toegewezen.

Opmerking: de functie range() accepteert alleen gehele getallen als argumenten en genereert reeksen van gehele getallen.

Lussen

Oefening: via for-lus het woord mississippily tellen

Weet jij wat Mississippi is? Het is de naam van een van de staten en rivieren in de Verenigde Staten. De rivier de Mississippi is ongeveer 2.340 mijl lang, waardoor het de op een na langste rivier in de Verenigde Staten is (de langste is de rivier de Missouri). Het is zo lang dat een enkele druppel water 90 dagen nodig heeft om de hele lengte af te leggen!

Het woord Mississippi wordt ook voor een iets ander doel gebruikt: *mississippily tellen*.

Het wordt gebruikt om seconden te tellen. Het idee erachter is dat het toevoegen van het woord Mississippi aan een getal bij het hardop tellen van seconden ervoor zorgt dat ze dichter bij de kloktijd klinken, en daarom duurt "één Mississippi, twee Mississippi, drie Mississippi" ongeveer drie seconden! Het wordt vaak gebruikt door kinderen die verstoppertje spelen om er zeker van te zijn dat de zoeker eerlijk telt.

Je taak is hier heel eenvoudig: schrijf een programma dat een for-lus gebruikt om "mississippily" tot vijf te tellen. Nadat het programma tot vijf heeft geteld, moet het laatste bericht op het scherm worden afgedrukt: "Klaar of niet, hier kom ik!"

Lussen

break statements

Af te raden!

```
largest_number = -99999999
counter = 0

while True:
    number = int(input("Enter a number or type -1 to end the program: "))
    if number == -1:
        break
    counter += 1
    if number > largest_number:
        largest_number = number

if counter != 0:
    print("The largest number is", largest_number)
else:
    print("You haven't entered any number.")
```

Lussen

continue statements

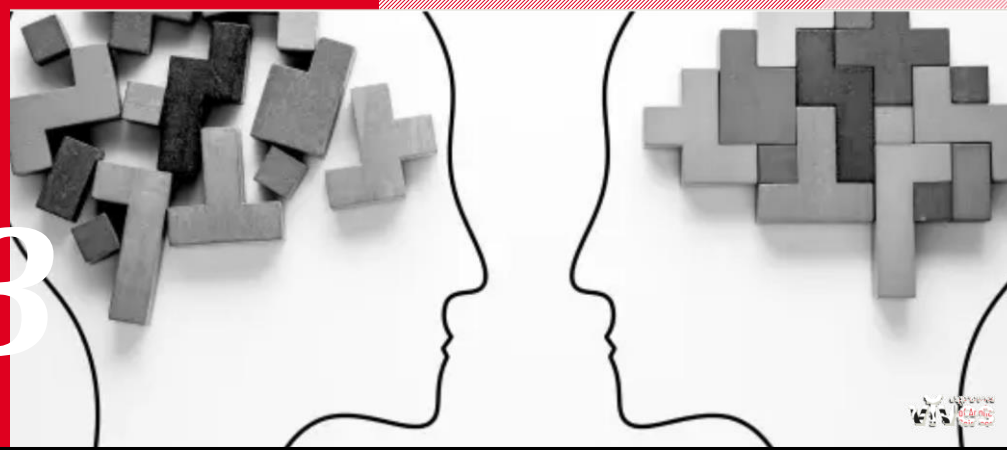
Af te raden!

```
largest_number = -99999999
counter = 0

number = int(input("Enter a number or type -1 to end program: "))

while number != -1:
    if number == -1:
        continue
    counter += 1

    if number > largest_number:
        largest_number = number
    number = int(input("Enter a number or type -1 to end the program: "))

if counter:
    print("The largest number is", largest_number)
else:
    print("You haven't entered any number.")
```


vives University of Applied Sciences

Computerlogica

3



Computerlogica

De AND-operator

```
counter > 0 and value == 100
```

Voorwaarde A	Voorwaarde B	A en B
False	False	False
False	True	False
True	False	False
True	True	True

Computerlogica

De OR-operator

```
counter > 0 or value == 100
```

Voorwaarde A	Voorwaarde B	A of B
False	False	False
False	True	True
True	False	True
True	True	True

Computerlogica

De NOT-operator

```
not value == 100
```

Voorwaarde A	A
False	True
True	False

Computerlogica

Combineren en inverteren van vergelijkingen

Vergelijkingen kan je met deze operatoren combineren:

```
not test == 0 or value == 1 and frequency == 100
```

Opmerking: deze haakjes zijn hier overbodig, maar maakt het wel leesbaarder:

```
not test == 0 or (value == 1 and frequency == 100)
```

Volgorde van uitvoeren
(van links naar rechts in Python):
1. Haakjes
2. not-operator
3. Vergelijkingsoperatoren
4. and-operator
5. or-operator

Computerlogica

Combineren en inverteren van vergelijkingen

Voor een andere werking dan de standaardwerking, moeten er haakjes staan!

- Als het "or"-deel eerst moet uitgevoerd worden en daarna "and":

```
(not test == 0 or value == 1) and frequency == 100
```

- Als het resultaat van de vergelijkingen pas op het einde geïnverteerd moet worden:

```
not(test == 0 or value == 1 and frequency == 100)
```


Computerlogica

Bitoperatoren

Er zijn vier operatoren waarmee je enkele gegevensbits kunt manipuleren. Ze worden bitsgewijze operatoren genoemd.

```
& (ampersand) – bitwise conjunction
| (bar) – bitwise disjunction
~ (tilde) – bitwise negation
^ (caret) – bitwise exclusive or (xor)
```

Computerlogica

Logische versus bit operatoren

```
i = 15
j = 22
log = i and j
print(log)
```

log-variabele is True

Uitvoer: 22

```
i = 15
j = 22
bit = i & j
print(bit)
```

bit-variabele is 00110

Uitvoer: 6

Computerlogica

Omgaan met specifieke bits

```
flag_register = 0x1234                          # 0b1001000110100
the_mask = 8                                    # 0b0000000001000

print(bin(flag_register))                       # 0b1001000110100
if flag_register & the_mask:
    print("set")
else:
    print("not set")

flag_register = flag_register | the_mask        # 0b1001000111100
print(bin(flag_register))

flag_register = flag_register ^ the_mask        # 0b1001000110100
print(bin(flag_register))
```

Computerlogica

Binaire shift

Python biedt nog een andere bewerking met betrekking tot enkele bits: verschuiven. Dit wordt alleen toegepast op gehele waarden en je mag geen floats gebruiken als argumenten hiervoor.

Deze handeling pas je al heel vaak en vrij onbewust toe. Hoe vermenigvuldig je een getal met tien? Kijk eens:

12345 × 10 = 123450

Zoals je kunt zien, is vermenigvuldigen met tien in feite een verschuiving van alle cijfers naar links en het resulterende gat opvullen met nul.

Delen door tien? Kijk eens:

12340 ÷ 10 = 1234

Delen door tien is niets anders dan de cijfers naar rechts verschuiven.

Computerlogica

Binaire shift

Dezelfde soort bewerking wordt uitgevoerd door de computer, maar met één verschil: aangezien twee het grondtal is voor binaire getallen (niet 10), komt het verschuiven van een waarde één bit naar links dus overeen met vermenigvuldigen met twee; respectievelijk, één bit naar rechts verschuiven is als delen door twee (merk op dat het meest rechtse bit verloren gaat).

De shift-operatoren in Python zijn een paar digraphs: << en >>, die duidelijk aangeven in welke richting de shift zal werken.

```
var = 17
var_right = var >> 1
var_left = var << 2
print(bin(var), bin(var_left), bin(var_right))
```

Functies

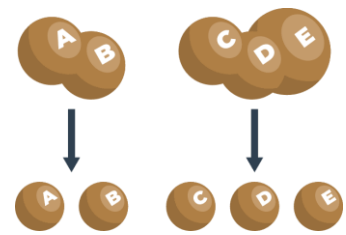
Functies

Waarom functies?

Functies vereenvoudigen het werk van het programma aanzienlijk, omdat elk stuk code afzonderlijk kan worden gecodeerd en afzonderlijk kan worden getest.

Als een stuk code zo groot wordt dat lezen en onderschatten een probleem kan veroorzaken, overweeg dan om het op te splitsen in afzonderlijke, kleinere problemen, en elk van hen in de vorm van een afzonderlijke functie te implementeren.

Deze decompositie (ontleding) gaat door totdat je een reeks korte functies krijgt, die gemakkelijk te begrijpen en te testen zijn.



Functies

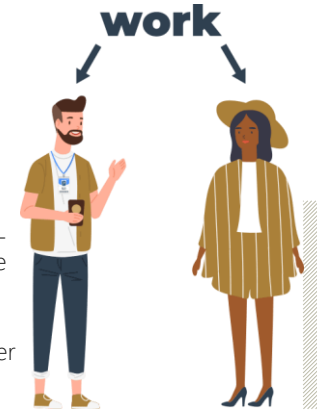
Decompositie

Het komt vaak voor dat het probleem zo groot en complex is dat het niet aan één ontwikkelaar kan worden toegewezen en er een team van ontwikkelaars aan moet werken. Het probleem moet worden opgesplitst tussen verschillende ontwikkelaars op een manier die hun efficiënte en naadloze samenwerking verzekert.

Het lijkt ondenkbaar dat meer dan één programmeur tegelijkertijd hetzelfde stuk code schrijft, dus de taak moet over alle teamleden worden verdeeld.

Dit soort decompositie heeft een ander doel dan het eerder beschreven doel - het gaat niet alleen om het delen van het werk, maar ook om het delen van de verantwoordelijkheid onder veel ontwikkelaars.

Elk van hen schrijft een duidelijk gedefinieerde en beschreven set functies, die, wanneer ze in de module worden gecombineerd (we vertellen je hier later meer over), het uiteindelijke product opleveren.



Functies

Een eerste functie

```
def function_name():
    function_body
```

- Het begint altijd met het sleutelwoord def (van define)
- net na def komt de naam van de functie (de regels voor het benoemen van functies zijn precies hetzelfde als voor het benoemen van variabelen)
- na de functienaam is er plaats voor een paar haakjes (ze bevatten hier niets, maar dat zal binnenkort veranderen)
- de regel moet eindigen met een dubbele punt;
- de regel direct na def begint de hoofdtekst van de functie - een paar (minstens één) noodzakelijkerwijs geneste instructies, die elke keer dat de functie wordt aangeroepen zullen worden uitgevoerd; opmerking: de functie eindigt waar het nesten eindigt, dus je moet voorzichtig zijn.

Functies

Een eerste functie

```
def message():
    print("Enter a value: ")

print("We start here.")
message()
print("We end here.")
```


Functies

Regels

Je mag geen functie aanroepen die niet bekend is op het moment van aanroepen.

```
print("We start here.")
message()
print("We end here.")


def message():
    print("Enter a value: ")
```

Je mag geen functie en variabele met dezelfde naam hebben.

```
def message():
    print("Enter a value: ")

message = 1
```

Functies met parameters

5

Functies met parameters

Wat is een functie?

```
def message(number):
    print("Enter a number:", number)
```

Een parameter is eigenlijk een variabele, maar er zijn twee belangrijke factoren die parameters anders en speciaal maken:

- parameters bestaan alleen binnen functies waarin ze zijn gedefinieerd, en de enige plaats waar de parameter kan worden gedefinieerd is een spatie tussen een paar haakjes in het def statement;
- het toekennen van een waarde aan de parameter gebeurt op het moment dat de functie wordt aangeroepen, door het overeenkomstige argument op te geven.

Functies met parameters

Variabelen en functies

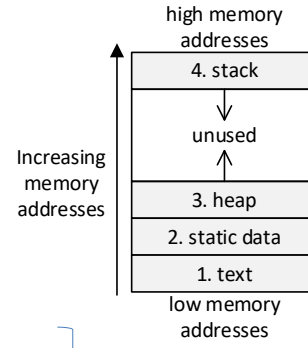
```
number = 1234
def message(number):
    print("Enter a number:", number)

message(1) # R1

# R0
```

Het is toegestaan en mogelijk om een variabele dezelfde naam te geven als de parameter van een functie.

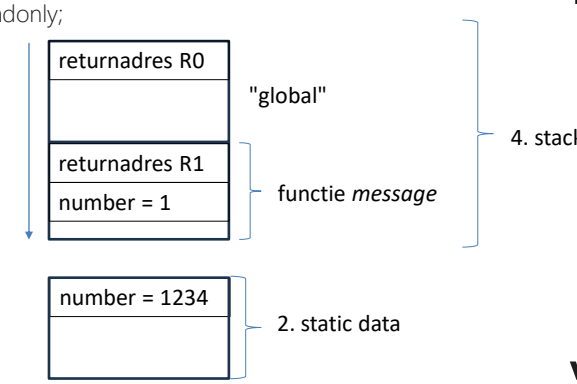
Functies met parameters

Functies met meerdere argumenten

Hoe worden variabelen allemaal bijgehouden in RAM-geheugen?
→ In *4. stack*: lokale variabelen per functie
→ In *2. static data*: globale variabelen
(*1. text:* bevat je uitvoerende code, is readonly;
3. heap: hier niet van toepassing)



Voor het vorig voorbeeld wordt dit bij het uitvoeren van functie message:



Functies met parameters

Keyword argument passing

```
def introduction(first_name, last_name):
    print("Hello, my name is", first_name, last_name)

introduction(first_name = "James", last_name = "Bond")
introduction(last_name = "Skywalker", first_name = "Luke")
```

Python biedt een andere conventie voor het doorgeven van argumenten, waarbij de betekenis van het argument wordt gedicteerd door de naam, niet door de positie - dit wordt het doorgeven van keyword argument genoemd.

Functies met parameters

Geparametriseerde functies

```
def introduction(first_name, last_name="Smith"):
     print("Hello, my name is", first_name, last_name)

introduction("James", "Doe")
introduction("Henry")
introduction(first_name="William")
```

Het komt soms voor dat de waarden van een bepaalde parameter vaker worden gebruikt dan andere. Bij dergelijke argumenten kan rekening worden gehouden met hun standaard (vooraf gedefinieerde) waarden wanneer de overeenkomstige argumenten zijn weggelaten.

Functies met parameters

De return-instructie

Alle eerder gepresenteerde functies hebben een bepaald effect - ze produceren wat tekst en sturen deze naar de console.

Natuurlijk kunnen functies - net als hun wiskundige broers en zussen - resultaten hebben.

Om functies een waarde te laten retourneren (maar niet alleen voor dit doel) gebruik je de return-instructie.

Dit woord geeft jou een volledig beeld van zijn mogelijkheden. Opmerking: het is een Python-sleutelwoord.

De retourinstructie heeft twee verschillende varianten - laten we ze afzonderlijk bekijken.

Functies met parameters

return zonder expressie of waarde

```
def happy_new_year(wishes = True):
    print("Three...")
    print("Two...")
    print("One...")
    if not wishes:
        return

    print("Happy New Year!")
```

Een False argument zal het gedrag van de functie wijzigen – de retourinstructie zal ervoor zorgen dat deze net voor de wensen wordt beëindigd.

Functies met parameters

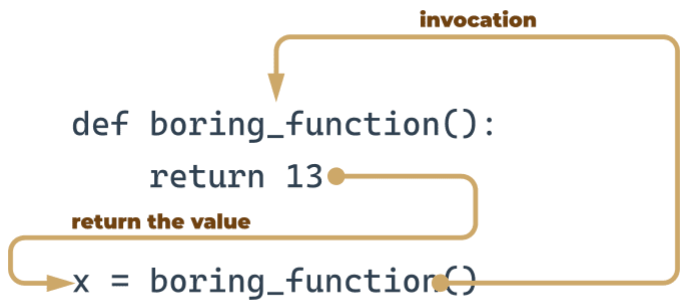
return met expressie of waarde

```
def boring_function():
    return 123

x = boring_function()

print("The boring_function has returned its result. It's:", x)
```

De functie evalueert de waarde van de uitdrukking en retourneert deze (vandaar nogmaals de naam) als het resultaat van de functie.

Functies met parameters

return met expressie of waarde



Functies met parameters

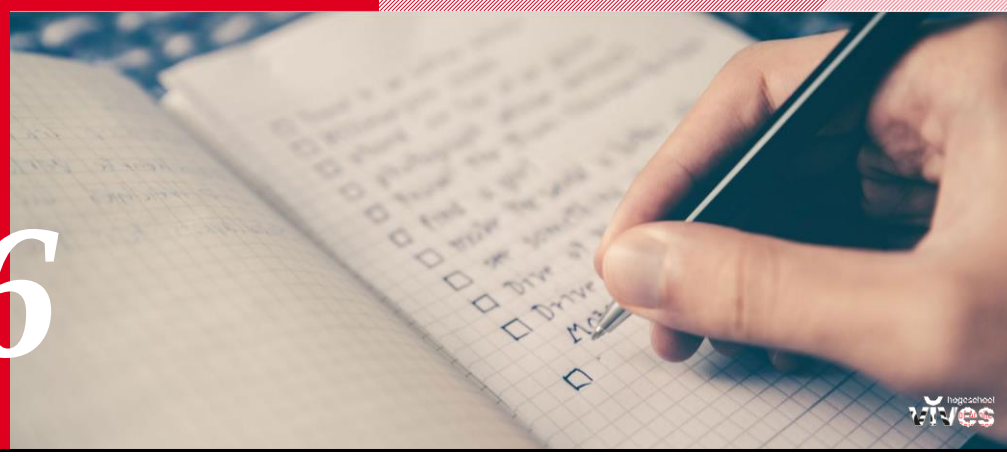
Oefening: schrikkeljaar bepalen d.m.v. je eigen functies te schrijven

Jouw taak is het schrijven en testen van een functie die één argument (een jaar) nodig heeft en True retourneert als het jaartal een schrikkeljaar is, of False anders.

Functies kunnen ook zichzelf oproepen (recursie), zie codevoorbeeld faculteit berekenen op Toledo.

Lists

6



Collections

Lists

Een list is een veranderlijk reekstype. Het is een verzameling van element, elk element is een scalar.

```
list1=["hello",1,4,8,"good"]
print(list1)
```

Lists gebruiken vierkante haakjes.

Opmerking: elk list-element kan van een ander datatype zijn (float, geheel getal of een ander soort gegevens).

Collections

Lists

Hoe verander je de waarde van een gekozen element in de lijst?

Laten we een nieuwe waarde van 111 toekennen aan het eerste element in de lijst. We doen het op deze manier:

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


numbers[0] = 111
print("New list contents: ", numbers)  # Current list contents.
```

Collections

Lists

Elk van de elementen van de lijst is afzonderlijk toegankelijk. Het kan bijvoorbeeld worden afgedrukt:

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

numbers[0] = 111
print("\nPrevious list contents:", numbers)  # Printing previous list contents.

numbers[1] = numbers[4]  # Copying value of the fifth element to the second.
print("Previous list contents:", numbers)  # Printing previous list contents.

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


Collections

Lists

De del() functie

```
del numbers[1]
print(len(numbers))
print(numbers)
```

Collections

Lists

Het ziet er misschien vreemd uit, maar negatieve indices zijn legaal en kunnen erg handig zijn. Een element met een index gelijk aan -1 is het laatste in de lijst.

```
numbers = [111, 7, 2, 1]
print(numbers[-1])
```


Collections

Oefening: lists

Er was eens een hoed. De hoed bevatte geen konijn, maar een lijst van vijf cijfers: 1, 2, 3, 4 en 5.

Jouw taak is:

- schrijf een coderegel die de gebruiker vraagt het middelste getal in de lijst te vervangen door een geheel getal dat door de gebruiker is ingevoerd (stap 1),
- schrijf een coderegel die het laatste element uit de lijst verwijdert (stap 2),
- schrijf een coderegel die de lengte van de bestaande lijst afdrukt (stap 3).

Klaar voor deze uitdaging?

Collections

Lists

Element toevoegen:

```
list.append(value)
```

Element invoegen:

```
list.insert(location, value)
```


Collections

Lists

Overlopen:

```
my_list = [10, 1, 8, 3, 5]
total = 0

for i in range(len(my_list)):
    total += my_list[i]

print(total)
```

Collections

Oefening: lists

The Beatles waren een van de meest populaire muziekgroepen van de jaren zestig en de best verkochte band in de geschiedenis. Sommige mensen beschouwen ze als de meest invloedrijke act van het rocktijdperk. Ze werden inderdaad opgenomen in Time Magazine's compilatie van de 100 meest invloedrijke mensen van de 20e eeuw.

De band onderging vele bezettingswisselingen, met als hoogtepunt in 1962 de bezetting van John Lennon, Paul McCartney, George Harrison en Richard Starkey (beter bekend als Ringo Starr).

Schrijf een programma dat deze veranderingen weerspiegelt en je laat oefenen met het concept van lijsten. Jouw taak is:

- stap 1: maak een lege lijst met de naam beatles;
- stap 2: gebruik de functie append() om de volgende leden van de band aan de lijst toe te voegen: John Lennon, Paul McCartney en George Harrison;
- stap 3: gebruik de for-lus en de append()-functie om de gebruiker te vragen de volgende leden van de band aan de lijst toe te voegen: *Stu Sutcliffe* en *Pete Best*;
- stap 4: gebruik de del-instructie om *Stu Sutcliffe* en *Pete Best* van de lijst te verwijderen;
- stap 5: gebruik de insert()-functie om *Ringo Starr* aan het begin van de lijst toe te voegen.

Collections

Lists

Elementen wissen

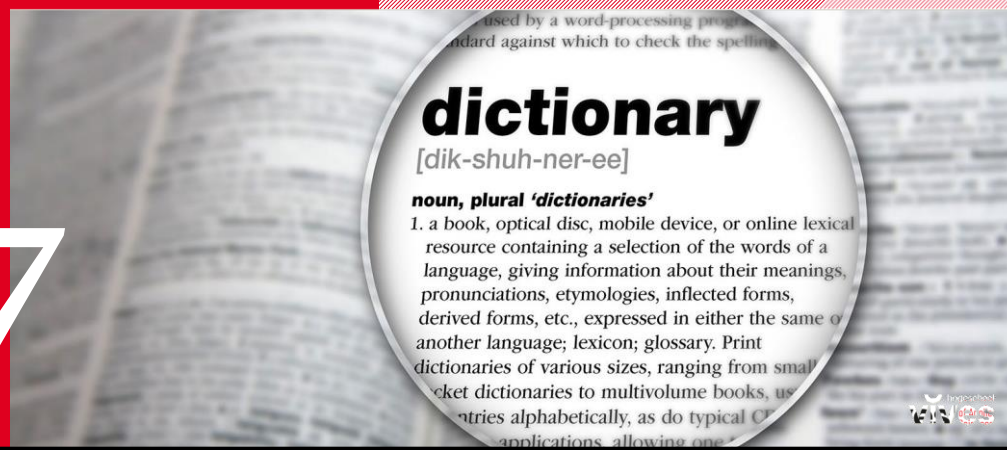
De lengte van een lijst kan variëren tijdens de uitvoering. Nieuwe elementen kunnen aan de lijst worden toegevoegd, andere kunnen eruit worden verwijderd. Dit betekent dat de lijst een zeer dynamisch geheel is.

Als je de huidige lengte van de lijst wilt controleren, kan je een functie met de naam len() gebruiken (de naam komt van lengte).

Overige collections

Tuples & Dictionaries

7



Collections

Collections

Python Collections worden gebruikt om gegevens op te slaan.

Lists	Tuples	Sets	Dictionaries
Een list is een verzameling geordende gegevens.	Een tuple is een geordende verzameling van gegevens.	Een set is een ongeordende verzameling.	Een dictionary is een ongeordende verzameling gegevens die gegevens opslaat in sleutel-waardeparen.

Collections

Tuples

Een tuple is een onveranderlijk reekstype. Het kan zich gedragen als een lijst, maar het kan niet worden gewijzigd.

```
tuple_1 = (1, 2, 4, 8)
tuple_2 = 1., .5, .25, .125
my_tuple = (1, 2, True, "a string", (3, 4), [5, 6], None)
empty_tuple = ()


print(tuple_1)
print(tuple_2)
```

Tupels gebruiken ronde haakjes, hoewel het ook mogelijk is om een tupel te maken met alleen een reeks waarden gescheiden door komma's.

Opmerking: elk tuple-element kan van een ander datatype zijn (float, geheel getal of een ander soort gegevens).

Tuples and dictionaries

Een tuple gebruiken

```
my_tuple = (1, 10, 100, 1000)

print(my_tuple[0])
print(my_tuple[-1])
print(my_tuple[1:])
print(my_tuple[:-2])

for elem in my_tuple:
    print(elem)
```

Als je de elementen van een tuple wilt ophalen om ze te kunnen lezen, kan je dezelfde conventies gebruiken die je gewend bent bij het gebruik van lijsten.

Tuples and dictionaries

Een tuple gebruiken

Wissen:

```
my_tuple = 1, 2, 3,
del my_tuple
print(my_tuple) # NameError: name 'my_tuple' is not defined
```

Als je de elementen van een tuple wilt ophalen om ze te kunnen lezen, kan je dezelfde conventies gebruiken die je gewend bent bij het gebruik van lijsten.

Tuples and dictionaries

Een tuple gebruiken

Wat kunnen tuples nog meer voor je doen?

```
my_tuple = (1, 10, 100)

t1 = my_tuple + (1000, 10000)  #the + operator can join tuples together
t2 = my_tuple * 3   #the * operator can multiply tuples, just like lists;

print(len(t2))#the len() function accepts tuples, and returns the number of elements contained inside;
print(t1)
print(t2)
print(10 in my_tuple)   #the in and not in operatoren work in the same way as in
lists. print(-10 not in my_tuple)
```

Tuples and dictionaries

Een tuple gebruiken

Je kunt een tuple elementen doorlopen (Voorbeeld 1),
controleren of een specifiek element (niet) aanwezig is in een tuple (Voorbeeld 2).

```
# Example 1
 tuple_1 = (1, 2, 3)
for elem in tuple_1:
     print(elem)

# Example 2
tuple_2 = (1, 2, 3, 4)
print(5 in tuple_2)
print(5 not in tuple_2)
```


Collections

Sets

Een set is een veranderlijk reekstype. Een set kan geen dubbele waarden bevatten en is ongeordend.

```
set1={1,2,3,4,5}
print(set1)
set2={3,7,1,6,1}
print(set2)
```

Sets gebruiken accolades.

Opmerking: elk set-element kan van een ander datatype zijn (float, geheel getal of een ander soort gegevens).

Collections

Dictionaries

Een dictionary is een veranderlijk reekstype.

```
dictionary = {"cat": "chat", "dog": "chien", "horse": "cheval"}
phone_numbers = {'boss': 5551234567, 'Suzy': 22657854310}
empty_dictionary = {}
print(dictionary)
```

De lijst met paren wordt omgeven door accolades, terwijl de paren zelf worden gescheiden door komma's en de sleutels en waarden door dubbele punten.

Opmerking: elk dictionary-element (zowel key als value) kan van een ander datatype zijn (float, geheel getal of een ander soort gegevens).

Collections

Dictionaries

In de wereld van Python heet het woord dat je zoekt een sleutel (key). Het woord dat je uit het woordenboek haalt, wordt een waarde (value) genoemd.

Dit betekent dat een woordenboek een verzameling sleutel-waardeparen is. Opmerking:

- elke sleutel moet uniek zijn
- een sleutel kan elk onveranderlijk type object zijn: het kan een getal zijn (geheel getal of float), of zelfs een string, maar geen lijst;
- een dictionary is geen list – een list bevat een reeks genummerde waarden, terwijl een woordenboek waardenparen bevat;
- de functie len() werkt ook voor woordenboeken – het geeft het aantal sleutel-waarde-elementen in het woordenboek terug;
- een dictionary is een one-way tool – als je een Engels-Frans woordenboek hebt, kun je zoeken naar Franse equivalenten van Engelse termen, maar niet andersom.

Collections

Dictionaries

Als je een van de waarden wilt ophalen, moet je een geldige sleutelwaarde opgeven:

```
print(dictionary['cat'])
print(phone_numbers['Suzy'])
```


Collections

Dictionaries

Zoeken in een dictionary:

```
dictionary = {"cat": "chat", "dog": "chien", "horse": "cheval"}
words = ['cat', 'lion', 'horse']

for word in words:
    if word in dictionary:
        print(word, "->", dictionary[word])
    else:
        print(word, "is not in dictionary")
```

Collections

Dictionary-functies

De keys()-functie:

```
dictionary = {"cat": "chat", "dog": "chien", "horse": "cheval"}

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

De functie retourneert een itereerbaar object dat bestaat uit alle sleutels die in het woordenboek zijn verzameld. Als je een groep sleutels hebt, heb je op een gemakkelijke en handige manier toegang tot het hele woordenboek.

Collections

Dictionary-functies

De items()-functie:

```
dictionary = {"cat": "chat", "dog": "chien", "horse": "cheval"}

for english, french in dictionary.items():
    print(english, "->", french)
```

Laten we nu eens kijken naar een dictionary-functie met de naam items(). De functie retourneert tuples (dit is het eerste voorbeeld waarin tuples meer zijn dan alleen een voorbeeld van zichzelf) waarbij elke tuple een sleutel-waardepaar is.

Collections

Dictionary-functies

Waarden wijzigen:

```
dictionary = {"cat": "chat", "dog": "chien", "horse": "cheval"}

dictionary['cat'] = 'minou'
print(dictionary)
```

Waarden wissen:

```
phonebook = {} # an empty dictionary

phonebook["Adam"] = 3456783958 # create/add a key-value pair
print(phonebook) # outputs: {'Adam': 3456783958}

del phonebook["Adam"]
print(phonebook) # outputs:
```


Collections

Dictionary-functies

De values()-functie:

```
dictionary = {"cat": "chat", "dog": "chien", "horse": "cheval"}


for french in dictionary.values():
    print(french)
```

Collections

Dictionary-functies

Waarde invoegen:

```
pol_eng_dictionary = {"kwiat": "flower"}

pol_eng_dictionary.update({"gleba": "soil"})
print(pol_eng_dictionary) # outputs: {'kwiat': 'flower', 'gleba': 'soil'}
```

Laatste element wissen:

```
pol_eng_dictionary.popitem()
print(pol_eng_dictionary) # outputs: {'kwiat': 'flower'}
```


Collections

Dictionary-functies

Waarde wissen:

```
del pol_eng_dictionary["zamek"] # remove an item
```

Alles wissen:

```
pol_eng_dictionary.clear() # removes all the items
```

Kopiëren:

```
copy_dictionary = pol_eng_dictionary.copy()
```