PROGRAMMEREN 1

LES 3

UCLL HOGESCHOOL

OVERZICHT

- Willekeurigheid
- Iteraties
- Computationeel denken: algoritmisch denken (uitbreiding)

VORIGE LES

QUIZ

Wat is de uitvoer van volgende code?

```python
a = True
b = False
c = True

d = False

if a and (b and (d and not d)):
  print("A")
elif not c or (b and a):
  print("B")
elif (d or c and a):
  print("C")
if b or c:
  print("D")
```

A, dan D

B, dan C

C, dan D

Enkel B

QUIZ

Wat is de uitvoer van volgende code?

```python
x = 5
y = "#!*?"
z = true
if x and y:
  if not z:
    print("optie 1")
   else:
     print("optie 2")
elif x = z:
      print("optie 3)
```

optie 1

optie 2

optie 3

Iets anders

Error/programma werkt niet

RESULTATEN ONDERZOEK

- `True` en `False` in andere talen
- Logische vergelijkingen en operatoren in andere talen
- if/elif/else en indentaties

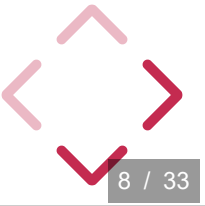
WILLEKEURIGHEID

WILLEKEURIGHEID

Willekeurigheid (*Eng: randomness*) is alomtegenwoordig in het dagelijkse leven:

- Een dobbelsteen werpen
- Het weer voorspellen
- Automatisch nieuwe werelden creëren in videospellen
- De kaarten die je krijgt in het begin van een kaartspel

Hoe kunnen we dit modelleren met computers?



`random`

Elke programmeertaal heeft gewoonlijk een functie die een willekeurige `float` genereert tussen `0.0` (inclusief) en `1.0` (exclusief).

In wiskundige notatie: het domein is [0, 1[

Specifiek voor Python:

- De functie zit in de module `random`. Je moet deze module eerst importeren met `import random`
- De functie zelf noemt ook `random` (verwarrend)

```python
import random

x = random.random()
print(x)
```

MEER `random()`

Opdracht: Pas voorgaande code aan zodat x in het domein [0, 4[ligt.

Opdracht: Pas voorgaande code aan zodat x in het domein [1, 7[ligt.

Opdracht: Pas voorgaande code aan zodat x als **geheel** getal in [2, 5] ligt. Gebruik `math.floor()` om naar beneden af te ronden.

MEER `random()`

Opdracht: Pas voorgaande code aan zodat x in het domein [0, 4[ligt.

```python
x = random.random() * 4
```

Opdracht: Pas voorgaande code aan zodat x in het domein [1, 7[ligt.

Opdracht: Pas voorgaande code aan zodat x als **geheel** getal in [2, 5] ligt. Gebruik `math.floor()` om naar beneden af te ronden.

MEER `random()`

Opdracht: Pas voorgaande code aan zodat x in het domein [0, 4[ligt.

```python
x = random.random() * 4
```

Opdracht: Pas voorgaande code aan zodat x in het domein [1, 7[ligt.

```python
x = 1 + (random.random() * 6)
```

Opdracht: Pas voorgaande code aan zodat x als **geheel** getal in [2, 5] ligt. Gebruik `math.floor()` om naar beneden af te ronden.

MEER `random()`

Opdracht: Pas voorgaande code aan zodat x in het domein [0, 4[ligt.

```python
x = random.random() * 4
```

Opdracht: Pas voorgaande code aan zodat x in het domein [1, 7[ligt.

```python
x = 1 + (random.random() * 6)
```

Opdracht: Pas voorgaande code aan zodat x als **geheel** getal in [2, 5] ligt. Gebruik `math.floor()` om naar beneden af te ronden.

```python
x = int(math.floor(2 + (random.random() * 4)))
```

MEER `random()`

Opdracht: Pas voorgaande code aan zodat x in het domein [0, 4[ligt.

```python
x = random.random() * 4
```

Opdracht: Pas voorgaande code aan zodat x in het domein [1, 7[ligt.

```python
x = 1 + (random.random() * 6)
```

Opdracht: Pas voorgaande code aan zodat x als **geheel** getal in [2, 5] ligt. Gebruik `math.floor()` om naar beneden af te ronden.

```python
x = int(math.floor(2 + (random.random() * 4)))
```

Deze transformaties werken in eender welke taal.

Omdat het laatste geval zo frequent voorkomt, heeft Python hier een bestaande functie voor: `random.randint(a, b)`

`random()`?

`random` in de meeste talen is eigenlijk maar pseudorandom (schijnwillekeurigheid). Achter de schermen volgt de computers strikte regels voor het aanmaken van 'willekeurige' getallen.

Een versimpeld algoritme:

1. Ga op zoek naar één cijfer om mee te starten. Dit getal is je **seed**
2. Transformeer je seed naar een getal binnen [0, 1[. De transformatie gebeurt volgens bepaalde standaard formules. Dit is je eerste willekeurig getal
3. Voor elk volgend willekeurig getal neem je het **vorige** willekeurige getal als startpunt

`random()` EN SEEDS

Als je weet wat de seed is, kan je alle volgende getallen voorspellen. Vandaar dat men spreekt over schijnwillekeurigheid (pseudorandomness).

Voorbeelden van seeds:

- De klok van je apparaat tot op een nanoseconde nauwkeurig
- Het aantal beelden (frames) tussen het opstarten van een spel en het indrukken van een knop
- Een foto van een reeks lavalampen, omgezet naar een cijfervoorstelling (Link)
- ...

ITERATIES

CODE HERHALEN

INTRODUCTIE

Bekijk volgend stuk code om tot 3 te tellen:

```python
print(1)
print(2)
print(3)
```

- Breidt deze code uit zodat ze tot 5 telt

INTRODUCTIE

Bekijk volgend stuk code om tot 3 te tellen:

```python
print(1)
print(2)
print(3)
```

- Breidt deze code uit zodat ze tot 5 telt
- Wat als je tot 100, 10 000, ... moet tellen?

WAAROM ITERATIES?

Heel vaak willen we stukken code **herhalen**:

- Het aantal leerlingen op een school tellen
- Een muntstuk 10 keer opwerpen en het resultaat noteren
- 60 keer per seconden het scherm van een videospel verversen
- Voor elke leerling in een klas het gemiddelde van hun taken en toetsen berekenen

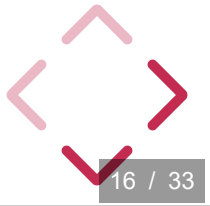
Gewoon code kopiëren en plakken is niet voldoende:

- Niemand heeft zin om hetzelfde stuk code 100 keer kopiëren en plakken
- Wat als de opdracht veranderd? Bv. slechts 50 keer iets uitvoeren in plaats van 200 keer
- Soms weten we niet op voorhand hoe vaak we iets moeten herhalen

EERSTE SOORT: ONEINDIG

```python
while True:
  print("Bezig met uitvoeren...")
```

- `while` is een speciaal woord in Python dat een herhaling aangeeft
- `True` geeft aan dat de code heel de tijd herhaald wordt
- Op het einde van de `while` staat een `:`
- Alles binnen de herhaling is naar rechts geïndenteerd. Er kunnen **meerdere** lijnen binnen de herhaling staan



EERSTE SOORT: ONEINDIG

```python
while True:
  print("Bezig met uitvoeren...")
```

Eens deze code start, kan ze niet uit zichzelf stoppen! Ze blijft (theoretisch) oneindig lang doorgaan.

Om een actief programma te stoppen, zijn er een paar manieren:

- Druk de toetsencombinatie `Ctrl + C` in in de console/terminal
- Zoek een stop-knop (niet altijd beschikbaar)

Oneindige lussen worden soms gebruikt in videospellen en andere programma's die continu werk moeten verrichten.

Omdat ze zo moeilijk te stoppen zijn, **vermijd je beter oneindige herhalingen.**

TWEEDE SOORT: CONDITIONEEL

```python
import random

kans_op_regen = random.random()

while kans_op_regen > 0.5:
  print("Ik blijf nog een dag binnen")

  # Nieuwe dag, nieuwe kans
  kans_op_regen = random.random()

# weinig kans op regen
print("Tijd om buiten te komen")
```

TWEEDE SOORT: CONDITIONEEL

```python
while kans_op_regen > 0.5:
```

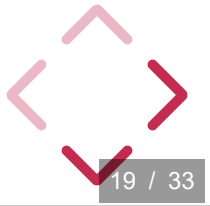
Lees dit als: *zolang* `kans_op_regen` *groter is dan* `0.5`*, voer onderstaande code uit:*

Algemeen: de code in de herhaling wordt uitgevoerd zolang de conditie `True` is.

Het is belangrijk dat de conditie kan **veranderen**. Als de conditie nooit verandert, krijg je een oneindige herhaling.

```python
# weinig kans op regen
print("Tijd om buiten te komen")
```

Deze code wordt pas uitgevoerd **na** de lus. Op dat moment is de conditie `False`.



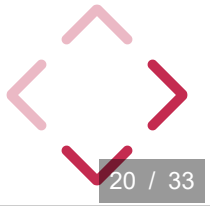
QUIZ

Hoeveel keer zal `"hallo"` in de uitvoer verschijnen?

```python
a = True
b = a and True

while b:
  print("hallo")
  a = False
```

0	1
meer dan 1 keer	Error/programma werkt niet



QUIZ

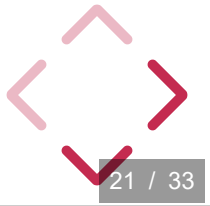
Hoeveel keer zal `"hallo"` in de uitvoer verschijnen?

Opgelet: de code is niet hetzelfde als in de vorige vraag.

```python
a = False
b = a and True

while b:
  print("hallo")
  b = a
```

0	1
meer dan 1 keer	Error/programma werkt niet



INTERMEZZO: VARIABELEN AANPASSEN

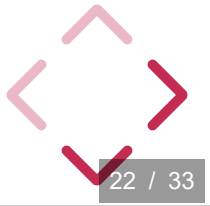
Jullie weten al dat variabelen aanpasbaar kunnen zijn. Variabelen kunnen ook aangepast worden op basis van hun **huidige waarde**.

Een veelgebruikte techniek in programmeren is een variabele maken die een teller voorstelt. De volgende waarde wordt berekend op basis van de vorige:

```python
i = 0
i = i + 1  # i is nu 1
i = i + 5  # i is nu 6
i = i * 2  # i is nu 12
i += 1  # verkorte notatie voor i = i + 1

# De volgende lijn is een veelgebruikte afkorting voor i = i + 1
# Python ondersteunt deze notatie NIET, sommige andere talen wel
# i++
```

- Tellervariabelen starten gewoonlijk op `0`
- Typische namen voor tellervariabelen zijn `i`, `j`, `teller`, `counter`, ...



HOE WERKT EEN HERHALING EXACT?

ALGORITME

```python
i = 0

while i < 5:
    print("nieuwe iteratie gestart"
    # ...
    i += random.random()

    # ...
    print("einde van iteratie")

print("lus voorbij")
```

1. De conditie wordt gecontroleerd
2. Als de conditie `False` is, ga naar de laatste stap
3. Anders wordt de code binnen de herhaling uitgevoerd
4. In deze code kan de conditie eventueel aangepast worden
5. **De code wordt altijd afgewerkt**, zelfs als de conditie halverwege zou veranderen naar `False`
6. Op het einde van de herhaling: ga

DERDE SOORT: EINDIG

```python
i = 0

while i < 3:
  print("En nog eens")
  i += 1
```

DERDE SOORT: EINDIG

```python
i = 0

while i < 3:
  print("En nog eens")
  i += 1
```

Bij een eindige herhaling weet je steeds op voorhand(*) hoe vaak de lus uitgevoerd zal worden.

Eindige lussen maken bijna altijd gebruik van tellervariabelen. Vermijd het gebruik van kommagetallen met tellervariabelen, aangezien de afrondingsfouten voor verrassingen kunnen zorgen.

(*) *op voorhand* betekent hier *vlak voor de lus start*. Het mag nog onzeker zijn op het moment dat het programma begint

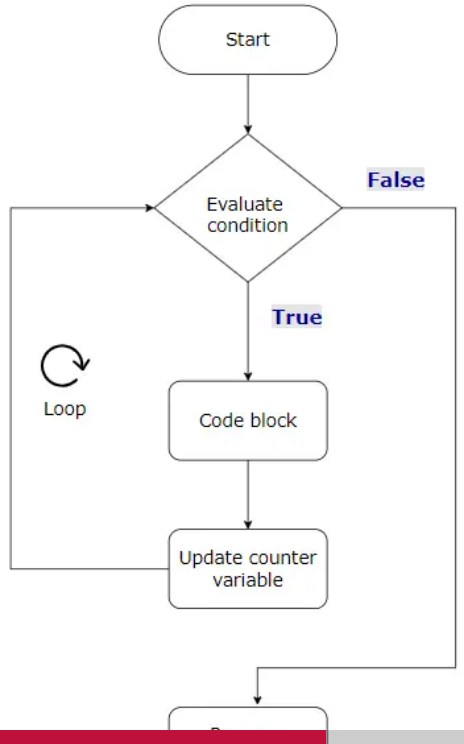
DE `for`-LUS

Veel programmeertalen hebben een versimpelde notatie voor eindige herhalingen: de `for`-lus.

```python
for i in range(3):
  print("En nog eens")
```

- Er wordt automatisch een tellervariabele aangemaakt (de naam is vrij te kiezen)
- Op het einde van de lus wordt de tellervariabele automatisch verhoogt met 1

De `for`-lus wordt veel gebruikt. De notatie verschilt wel sterk van taal tot taal.



DE `range()` VAN EEN `for`-LUS IN PYTHON

```python
for i in range(3):
```

Waardes van `i`:

- `0` → start lus
- `1`
- `2`
- `3` → stop lus, voer NIET uit

```python
for i in range(2, 5):
```

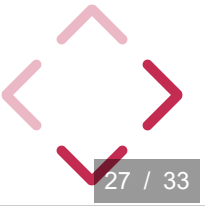
Waardes van `i`:

- `2` → start lus
- `3`
- `4`
- `5` → stop lus, voer NIET uit

```python
for i in range(10, 4, -2):
```

Waardes van `i`:

ALGEMEEN: `range(stop, start, step)`



OPDRACHTEN

OPDRACHT 1

Schrijf een programma dat telt van 0 (nul) tot en met 9. Gebruik een `while`-lus.

Pas nadien je code aan zodat het programma telt van 1 tot en met 10.

OPDRACHT 2

Maak de vorige opdracht opnieuw, maar gebruik nu een `for`-lus.

COMPUTATIONEEL DENKEN

DEEL 2

ALGORITMISCH DENKEN: UITBREIDING

DEELCOMPETENTIES CD

Iconen: CoDe-platform KU Leuven
Link naar de slides waarop deze leerstof gebaseerd is

Abstractie	Onnodige details weglaten
Veralgemening	Principes veralgemenen zodat ze breder toepasbaar zijn
Decompositie	Opdeling in kleinere, eenvoudigere problemen
Algoritmisch denken	Opstellen van stappenplan voor het oplossen van een probleem
Evaluatie	Kritisch bekijken van oplossing

ALGORITMISCH DENKEN

> *"Het opstellen van een reeks instructies die stap voor stap uitgevoerd moeten worden."*

Voorbeelden uit deze les:

- Een algoritme om herhalingen te beschrijven
- Een stroomdiagram van een for-lus

Hoe kan je afleiden of herhalingen nodig zijn in een algoritme?

- In de probleembeschrijving: woorden zoals 'herhaal', 'opnieuw', 'doe iets totdat', ...
- In een stappenplan: zinnen zoals 'keer terug naar stap X'
- Visueel: letterlijk een lus in een stroomdiagram

OPDRACHT TEGEN VOLGENDE LES

OPDRACHT TEGEN VOLGENDE LES

Onderzoek lussen in Kotlin en JavaScript. Meer bepaald:

- Hoe schrijf je een `while` en een `for` in beide talen?
- De `for`-lus is berucht om zijn nogal complexe syntax in sommige talen, zoals JavaScript. Zorg dat je zeker volgende statement in JavaScript begrijpt en kan aanpassen:

```javascript
for (let i = 0; i < 7; i++) { ... }
```

Tip: hou de slide over `range()` in Python bij de hand tijdens je zoektocht.

Tijdsinschatting: 30m - 1u