PROGRAMMEREN 1

LES 4

UCLL HOGESCHOOL

OVERZICHT

- Herhaling vorige les + bespreking opdracht
- Functies: vervolg

VORIGE LES

QUIZ

Hoeveel keer zal de lijn `print("echo")` uitgevoerd worden?

```python
import random
total = random.randint(5, 10)
i = 0
while i < total:
    print("echo")
    i += 2
```

Tussen de 4 en 9 keer	Tussen de 3 en 5 keer
Tussen de 5 en 10 keer	Iets anders

QUIZ

Welke cijfers worden afgeprint wanneer onderstaande code wordt uitgevoerd?

```python
for i in range(6, 2):
  print(i)
```

6, 5, 4, 3, 2	5, 4, 3, 2
6, 5, 4, 3	Iets anders

QUIZ

Welke waarde moet `x` hebben zodat enkel de cijfers 3 en 15 afgeprint worden?

```python
for i in range(3, 18, x):
  print(i)
```

12	15
6	Iets anders

OPDRACHT

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++) { ... }
```

FUNCTIES

```python
def bereken_oppervlakte_cirkel(straal):
  return straal * straal * 3.14

opp_1 = bereken_oppervlakte_cirkel(5)
opp_2 = bereken_oppervlakte_cirkel(25)
```

FUNCTIES

Functies zijn herbruikbare stukken code.

- Een functie kan je **definiëren** en **oproepen**
- De definitie van de functie schrijf je één keer
- Je kan een functie zo vaak oproepen als nodig
- Een functie kan een **resultaat** teruggeven (Eng: *return value*)
- Een functie kan **parameters** accepteren tussen de haakjes

Specifiek voor Python: de definitie van een functie moet **vóór** de oproep staan

Opmerking: functies in programmeren hebben een andere betekenis dan in wiskunde!

FUNCTIES: WAAROM?

- Je kan stukken code hergebruiken: minder kopiëren en plakken
- Je kan je programma opsplitsen in aparte, eenvoudigere blokken
- Als je met meerdere programmeurs samenwerkt, kan je het werk opsplitsen met behulp van functies
- In combinatie met andere concepten wordt je code veel flexibeler

FUNCTIES: INDENTATIE

Functies volgen dezelfde regels rond indentaties als selecties (`if-else`) en iteraties (`for, while`).

- Beëindig de declaratie van je functie (de `def`) met een `:`
- Indenteer de volledige *body* van je functie naar rechts
- Om je functie af te sluiten, indenteer je de code nadien terug naar links

```python
print("code voor de functie")

def mijn_functie():
  print("start functie")
  x = True
  if x:
    print("een if binnen een functie staat nog meer naar rechts")

  print("einde functie")

print("code na de functie")
```

FUNCTIES: `return`

Een functie kan een resultaat teruggeven door middel van `return`:

```python
def geef_antwoord_op_alles():
  return 42


x = geef_antwoord_op_alles()
print(x)  # 42
```

Een functie stopt onmiddellijk wanneer een `return` wordt bereikt. Code nadien wordt niet uitgevoerd:

```python
def geef_aantal_lln():
  return 5
  print("Deze lijn wordt nooit uitgevoerd")
```

FUNCTIES: PARAMETERS

Een **parameter** is een soort van **plaatshouder** voor waarden binnen een functie. Ze maken een functie nog meer flexibel en herbruikbaar. Parameters worden soms ook **argumenten** genoemd.

Je kan zoveel parameters toevoegen als nodig aan de definitie van je functie:

```python
def mijn_functie(param_1, param_2, param_3):
  print("parameters zijn", param_1, param_2, param_3)
```

Wanneer je de functie oproept, worden de parameters ingevuld met concrete waarden:

```python
mijn_functie("Thomas", 5 , False)   # parameters zijn Thomas 5 False
```

FUNCTIES: PARAMETERS

Aandachtspunten:

- Geef aan elke parameter een zinvolle naam
- De volgorde van parameters is belangrijk. Als je de volgorde verandert in de definitie, moet je ze ook aanpassen op alle plaatsen waar de functie opgeroepen wordt.
- Parameters bestaan enkel **binnen** hun functie. Het heeft geen zin om ze te gebruiken buiten hun functie

FUNCTIES EN VARIABELEN

Een functie kan geen variabelen aanpassen die buiten de functie gedefinieerd zijn(*).

Een functie kan een variabele tijdelijk overschrijven. De nieuwe waarde 'overschaduwt' de oude binnen de functie *(Eng: shadowing)*.

Dit is niet universeel. Sommige talen hebben ander regels, of passen *shadowing* helemaal niet toe.

(*) Er bestaan uitzonderingen op, zie later.

QUIZ

Wat gaat er op het scherm verschijnen?

```python
x = 5

def pas_x_aan():
    x = 8

print(x)
```

5	8
Niets	Het programma bevat een fout

QUIZ

Vergeet niet om functies op te roepen! Enkel definiëren heeft geen effect op de rest van de code

Nieuwe poging: wat gaat er op het scherm verschijnen?

```python
x = 5


def pas_x_aan():
    x = 8
    print(x)   # Een extra print()

pas_x_aan()   # Nu roepen we de functie effectief op
print(x)
```

Eerst 5, dan 8	Eerst 8, dan 8
Eerst 8, dan 5	Eerst 5, dan 5

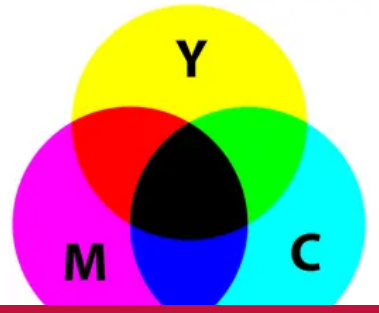
FUNCTIES: OPDRACHTEN

Schrijf een functie `is_even()` die één parameter accepteert: een cijfer. De functie geeft `True` terug als het cijfer even is, anders geeft ze `False` terug.

```python
print(is_even(5))   # False
print(is_even(8))   # True
```

Schrijf een functie `meng_kleuren()` die twee kleuren als parameters accepteert. Mogelijke waarden zijn cyaan (`"c"`), magenta (`"m"`) en geel (`"y"`). De functie geeft terug welke nieuwe kleur je bekomt als je de twee kleuren mengt. Naast voorgaande drie kleuren kan je ook rood (`"r"`), groen (`"g"`) en blauw (`"b"`) bekomen.

```python
print(meng_kleuren("c", "m"))   # "b
print(meng_kleuren("m", "y"))   # "r
print(meng_kleuren("c", "c"))   # "c
```



OPDRACHTEN

OPDRACHTEN OP DODONA

Functies worden behandeld in reeks 4 op Dodona.

Deze reeks bevat ook al oefeningen op datastructuren (voornamelijk lijsten). Dit wordt later behandeld.

Volgende opdrachten kan je oplossen zonder lijsten:

- MacArthurs wiskundetruc
- Big Mac index
- Tandwielen

OPDRACHT TEGEN VOLGENDE LES

Onderzoek functies in Kotlin en 2 andere programmeertalen (uitgezonder Python). Meer bepaald:

- Hoe declareer je een functie?
- Hoe voeg je parameters toe aan je functiedeclaratie?
- Hoe geef je een resultaat terug van een functie?
- Is de taal strikt wat betreft types in functies? Moet je bv. aangeven wat het type van een parameter is?
  - Indien ja: hoe noteer je de types in de functiedeclaratie?

Tijdsinschatting: 30m - 1u

OPDRACHT TEGEN VOLGENDE LES

Onderzoek functies in Kotlin en 2 andere programmeertalen (uitgezonder Python). Meer bepaald:

- Hoe declareer je een functie?
- Hoe voeg je parameters toe aan je functiedeclaratie?
- Hoe geef je een resultaat terug van een functie?
- Is de taal strikt wat betreft types in functies? Moet je bv. aangeven wat het type van een parameter is?
  - Indien ja: hoe noteer je de types in de functiedeclaratie?

Tijdsinschatting: 30m - 1u

OPDRACHT TEGEN VOLGENDE LES

Onderzoek functies in Kotlin en 2 andere programmeertalen (uitgezonder Python). Meer bepaald:

- Hoe declareer je een functie?
- Hoe voeg je parameters toe aan je functiedeclaratie?
- Hoe geef je een resultaat terug van een functie?
- Is de taal strikt wat betreft types in functies? Moet je bv. aangeven wat het type van een parameter is?
  - Indien ja: hoe noteer je de types in de functiedeclaratie?

Tijdsinschatting: 30m - 1u