08

FUNCTIES

WAT LEREN WE?

- Het nut van functies
- Functies creëren
- Parameters en argumenten
- Waardes uit functies retourneren middels return
- Functie benamingen
- ~~Commentaar in functies~~
- Variabele scope en levensduur
- Lokale en globale variabelen
- Het gebruik van functies om grip te krijgen op complexiteit
- ~~Modules~~
- ~~Gebruik van een main() functie~~
- ~~Anonieme functies~~
- `ord()` en `chr()`

PROBLEEM

- Wat doet dit programma?

```
from math import sqrt, floor

aantal_gevraagd = int(input('aantal gevraagd: '))

som, aantal, n = 0, 0, 2


while aantal < aantal_gevraagd:
    i = 2
    while i <= floor(sqrt(n)) and n % i != 0:
        i += 1
    if i == floor(sqrt(n)) + 1:
        som, aantal = som + n, aantal + 1
    n += 1
print(som)
```

PROBLEEM

- Wat doet dit programma?

```
from math import sqrt, floor

aantal_gevraagd = int(input('aantal gevraagd: '))

som, aantal, n = 0, 0, 2


while aantal < aantal_gevraagd:
    if is_priem(n):
        som, aantal = som + n, aantal + 1
    n += 1
print(som)
```

NUT VAN EEN FUNCTIE

- **Encapsulatie**: Het "inpakken" van een nuttig stuk code op zo'n manier dat het gebruikt kan worden zonder kennis van de specifieke werking van de code.
- **Generalisatie**: Het geschikt maken van een stuk code voor diverse situaties door gebruik te maken van parameters.
- **Beheersbaarheid**: Het verdelen van een complex programma in gemakkelijk te bevatten delen.
- **Onderhoudbaarheid**: Het gebruik maken van betekenisvolle functienamen en logische opdelingen om een programma beter leesbaar en begrijpbaar te maken.
- **Herbruikbaarheid**: Het faciliteren van de overdraagbaarheid van code tussen programma's.

SYNTAX

- Syntax

```
def <functienaam>(<parameter_lijst>):
    <acties>
```

- Functienaan: `isEven` of `is_even`

- Voorbeelden:

```
def welkom(naam):
    print('Welkom terug ' + naam)
```

```
def discriminant(a, b, c):
    return (b ** 2) - (4 * a * c)
```

OPLOSSING

- Functie:

```
def is_priem(n):
    i = 2
    while i <= floor(sqrt(n)) and n % i != 0:
        i += 1
    return i == floor(sqrt(n)) + 1
```

PROBLEEM

- Wat doet dit programma?

```
from math import sqrt, floor


def is_priem(n):
    i = 2
    while i <= floor(sqrt(n)) and n % i != 0:
        i += 1
    return i == floor(sqrt(n)) + 1


# Hoofdprogramma ###
aantal_gevraagd = int(input('aantal gevraagd: '))
som, aantal, n = 0, 0, 2

while aantal < aantal_gevraagd:
    if is_priem(n):
        som, aantal = som + n, aantal + 1
    n += 1
print(som)
```

RETURN

- Return als laatste regel van een functie

```
def pythagoras(a, b):
    if a <= 0 or b <= 0:
        return -1
    return sqrt(a*a + b*b)
```

```
def pythagoras(a, b):
    if a <= 0 or b <= 0:
        return
    return sqrt(a*a + b*b)
```

- Voordelen: leesbaarheid, debugging

```
def pythagoras(a, b):
    c = -1
    if a > 0 and b > 0:
        c = sqrt(a*a + b*b)
    return c
```

- Vermijd **return** zonder expressie!

RETURN

- Meerdere waarden teruggeven

```
def wortels(a, b, discriminant):

    w1, w2 = None, None

    if discriminant >= 0:

        w1 = (-b + sqrt(discriminant)) / (2 * a)

        w2 = (-b - sqrt(discriminant)) / (2 * a)

    return w1, w2


wortel1, wortel2 = wortels(2,4,8)

print(wortel1, wortel2)
```

FUNCTIES DIE FUNCTIES GEBRUIKEN

```
from math import sqrt


def discriminant(a, b, c):
    return (b ** 2) - (4 * a * c)


def bereken_wortels(a, b, c):
    w1, w2 = None, None
    d = discriminant(a, b, c)
    if d >= 0:
        w1 = (-b + sqrt(d)) / (2 * a)
        w2 = (-b - sqrt(d)) / (2 * a)
    return w1, w2


wortel1, wortel2 = bereken_wortels(2, 4, -1)
print(wortel1, wortel2)
```

BEREIK EN LEVENSDUUR VAN VARIABELEN

- Scope = bereik

```
from random import randint
def gooi_muntstuk():
    rg = randint(0, 2)
    if rg == 0:
        muntstuk = 'kop'
    else:
        muntstuk = 'munt'
    return muntstuk

print(gooi_muntstuk())
print(rg, muntstuk)
```

GLOBALE EN LOKALE VARIABELEN

```
appel = 'appel'
banaan = 'banaan'
kers = 'kers'


def print_fruit():
    appel = 'olifant'
    banaan = 'aap'
    kers =  'giraf'
    print(appel, banaan, kers)


print_fruit()
print(appel, banaan, kers)
```

```
appel banaan kers
olifant aap giraf
```

ORD EN CHAR

- ASCII-tabel

```
>>> ord('A')
65
>>> ord('z')
122
>>> chr(65 + 25)
'Z'
>>> chr(122 - 25)
'a'
```