Start to Program: Python, Les 12

Modules

Python heeft een hele lijst ingebouwde code. Denk aan print(), int(), len(). Zoals je vorige week gezien hebt zijn dit reeds bestaande functies. Deze zijn beperkt, maar kunnen worden uitgebreid. Dit doe je door middel van modules.

Modules zijn Python bestanden met voorgeprogrammeerde code. Deze modules kunnen dan bestaan uit functies, variabelen,... Je kan dus ook zelf modules schrijven en deze toevoegen. Er bestaan ook een aantal ingebouwde modules in Python. Dit om het programma lichter te maken. Zo moeten niet alle Python standaard functies telkens worden ingeladen.

Modules importeren

Je kan voor je een module wil importeren eerst controleren of deze wel bestaat.

Hiervoor ga je rechtstreeks in de commandprompt of de terminal op zoek. Start daar Python en probeer je de module te importeren.

Als deze bestaat gaat de commandprompt gewoon verder. Anders zal je een foutmelding krijgen.

Probeer dit met een module die je weet dat die bestaat: random.

Nadien met een externe module: matplotlib.

> In de commandprompt geef je *python* in.
>
> import random
>
> import matplotlib
>
> sluit Python met quit() of ctrl+d

Je merkt dat matplotlib niet gekend is. Deze zal je eerst moeten installeren voor je er gebruik van kan maken. Daarover later meer.

Je kan één of meerdere gekende modules importeren zonder extra installatie.

Dit doe je als volgt:

```
import random
import math
```

Vanaf nu kan je in de rest van de file de bijhorende functies gebruiken van de random module en van de math module.

De random module heb je al eens gebruikt om willekeurige nummers te genereren.

De math module heeft uitgebreide wiskundige berekeningen. Zoals het afronden van een getal naar boven of beneden, wiskundige constantes zoals PI en berekenen van de sinus.

Voor je deze in de code kan gebruiken ga je eerst de module nog eens aanroepen. Zo weet het programma dat er gezocht moet worden in een andere module.

```
print(random.randint(1,25))
print(math.pi)
```

Je kan ook een enkele functie uit een module aanroepen.

Zo hoef je de module niet aan te spreken verder in je code.

```
from random import randint

print(randint(1,25))
```

Het sterretje, * , is hierbij een universele selector. Zo haal je alle functies van de module binnen. Dit is echter af te raden. Het omzeilt het doel van specifieke functies binnen te halen. Gebruik het dus niet om minder te moeten typen.

```
from random import *
```

Een betere manier om minder te typen is een alias geven aan de module. Zo blijf je specifiek werken en overzicht bewaren welke functies juist uit een module afkomstig zijn.

De meeste standaard Python modules zijn kort van naam, de naam is ook duidelijk.

Als je echter extra modules gaat installeren, zoals matplotlib, kan het handing zijn om deze een alias te geven.

```
import math as m
print(m.pi)
```

Nu kan je de math module overal met m aanspreken in je project.

OPDRACHT:

Ga op zoek naar de module turtle. Deze module bevat functies om met een visueel robotje, turtle, figuren te tekenen.

Maak een nieuw bestand aan turtle_test.py. Probeer hierin het voorbeeld van de Python documentation site.

Overloop de code en zoek de bijhorende functie op in de lijst.

Wat doen ze juist? Begrijp je ze?

Maak zelf nog 2 orginele andere figuren.

Een lijst van gekende modules voor Python vind je hier:
https://docs.python.org/3/py-modindex.html#cap-r

Ga op zoek naar de module die tijd toevoegt.

Print de volgende elementen.

1. huidige datum en tijd.
2. Het huidige jaar.
3. De huidige maand.
4. De nummer van de week.
5. De dag van de week, naam en nummer.
6. De dag van het jaar.
7. De dag van de maand.

Modules schrijven

Je kan alle zelfgeschreven Python bestanden gebruiken als een module. Deze moeten echter wel bereikbaar zijn voor je hoofdprogramma.

Dit wil zeggen dat het in eenzelfde projectmap zit als de uit te voeren module.

Neem het theorie en oefeningen bestand van vorige les er even bij.

Zet deze in een nieuwe map.

In deze map komt ook een nieuw bestand: hoofd.py

importeer nu het py bestand van vorige les.

Voer het bestand hoofd.py uit.

Wat merk je op?

Hoe komt dit?

Wat zou je beter opsplitsen?

Stel dat je al je modules bijhoud in een bepaalde map en niet je projectmap moet je een pad naar die map leggen.

Een keer dat het pad is vastgelegd kan je de modules van die map importeren.

```
import sys
sys.path.append('pad/naar/module/map')
import module_naam
```

Als laatste optie kan je in de commandprompt Python starten. Geef de volgende commando's in.

```
import sys
print(sys.path)
```

Dit geeft het installatie pad van Python weer. Neem het pad naar de map packages. Hierin kan je dan modules rechtstreeks in de Python installatie toevoegen.

Nu maakt deze deel uit van de Python bibliotheek en kan je ze ook rechtstreeks aanroepen in elk Python programma.

Uitbreiding lussen: break, continue en pass

Tot nu toe liet je alle lussen hun volledig beloop gaan. Je kan dit verloop echter beïnvloeden met enkele statements.

Break

Als er aan een bepaalde voorwaarde voldaan is met de lus beëindigen.

Hiervoor kan je het break statement gebruiken.

```
getal = 0
for getal in range(10):
        getal = getal + 1
        if getal == 5:
                break
        print(getal)

print('lus gestopt')
```

Continue

De lus zal stoppen op de voorwaarde van de if, maar nadien wel verder gaan.

```
getal = 0
for getal in range(10):
        getal = getal + 1
        if getal == 5:
                continue
        print(getal)

print('lus gestopt')
```

Je kan deze gebruiken om bepaalde uitkomsten te filteren zonder de lus te onderbreken.

Pass

De voorwaarde in de if zal worden overgeslagen.

```
getal = 0
for getal in range(10):
        getal = getal + 1
        if getal == 5:
                pass
        print(getal)

print('lus gestopt')
```

Je kan deze gebruiken als je een stuk van je code wil testen zonder een bepaalde voorwaarde, maar je wilt wel de code laten staan.

Door pass toe te voegen kan je sneller werken en het overzicht in de structuur behouden.

Gebruik dit om te debuggen.