Python 2 Cheat Sheet - CodeFever

▼ API's, HTTP & JSON
P2L2

Een **API** (Application Programming Interface) is een set van regels waarmee softwareprogramma's met elkaar kunnen communiceren.

We spreken vaak een API aan via **HTTP-requests** en gebruikten vaak **GET-requests** om data op te halen.
Het antwoord is vaak in **JSON-formaat.** Naast een antwoord, krijgen we ook een **statuscode** terug.
Enkele voorbeelden: 200 (OK), 400 (Bad Request), 404 (Not Found), 500 (Internal Server Error), ...

Voorbeeld van een HTTP-request:

```
import requests

response = requests.get(
    url=f"https://api.agify.io/",
    params={"name": "Bert"}
)
# Totale url is https://api.agify.io/?name=Bert
# Eventueel kan je ook headers meegeven, bv. "User-Agent"

if response.status_code == 200:
    data = response.json()  # Omzetten naar JSON
    leeftijd = data["age"]  # Haal de leeftijd uit de JSON
    print(f"Vermoedelijke leeftijd: {leeftijd} jaar.")
```

💡 Die eventuele parameters die meegegeven worden in de URL, worden **query parameters** genoemd.

Voorbeeld JSON:

```
{
    "name": "bert",
    "age": 73
}
```

▼ Break
P2L3

`break` wordt gebruikt om een lus te stoppen.

```
mijn_lijst = [1, 3, 5, 7, 9, 10, 11, 13, 15]
for nummer in mijn_lijst:
    if nummer % 2 == 0:
        print("Het eerste even getal is gevonden:", nummer)
        break
```

💡 Gelijkaardig: **`continue`**, dat de huidige iteratie overslaat en verdergaat met de volgende.

▼ Try & Except
P2L3

Met `try` en `except` kan je fouten opvangen in je programma.
"probeer deze code (try) tenzij daar een bepaalde error komt (except), voer dan de andere code uit".

```
try:
    aantal_personen = int(input("Hoeveel personen zijn er: "))
    resultaat = 50 / aantal_personen
    print(f"Het resultaat is: {resultaat}")
except ZeroDivisionError:
    print("Je kunt niet door nul delen!")
except ValueError:
    print("Dit is geen geldig getal.")
# Een andere error dan de twee bovenstaanden? Dan stopt het programma met een error!
```

▼ Klasses & Objecten
P2L5

Een **klasse** is een blauwdruk of recept voor een object. Een **object** is een instantie van een klasse. Het heeft unieke eigenschappen (variabelen) en methodes (functies).

Elke klasse heeft een **constructor**, die wordt aangeroepen wanneer een object wordt gemaakt:
`def __init__(self, parameters)`.
'self' is een tijdelijke naam voor het object, en wordt binnen de klasse gebruikt.

```
class Film:
    def __init__(self, titel, jaar, minuten):
        self.titel = titel
        self.jaar = jaar
        self.minuten = minuten

    def toon_leeftijd_in_jaar(self, toekomstig_jaartal):
        leeftijd = toekomstig_jaartal - self.jaar
        print(f"{self.titel} wordt {leeftijd} jaar in {toekomstig_jaartal}.")

shrek = Film("Shrek", 2001, 90)
shrek.toon_leeftijd_in_jaar(2031)  # Shrek wordt 30 jaar in 2031.
print(shrek.titel)  # Shrek
```

▼ Async & Await
P2L8

Async en **await** worden gebruikt om asynchrone code te schrijven in Python.
Asynchrone code is code die niet in een vaste volgorde wordt uitgevoerd.

async wordt gebruikt om een functie aan te duiden als asynchroon.
await wordt gebruikt om te wachten op een asynchrone taak.

```
import asyncio

async def taak(duur):
    print(f"Start taak voor {duur} seconden")
    await asyncio.sleep(duur)
    print(f"Einde taak na {duur} seconden")

async def mijn_asynchrone_functie():
    taak1 = taak(2)
    taak2 = taak(3)
    await asyncio.gather(taak1, taak2)  # Wacht tot beide taken klaar zijn

asyncio.run(mijn_asynchrone_functie())
```

▼ Decorators
P2L8

Een **decorator** is een functie die een andere functie aanpast of uitbreidt.
Ze worden aangegeven met het **@**-symbool boven een functie-definitie.

```
# Voorbeeld van een decorator bij Discordbots
@bot.event
async def on_ready():
    print('Ready!')
```

▼ IDE (Integrated Development Environment)
P1L1

Een IDE is software die programmeurs helpt bij het schrijven, testen en debuggen van code.
Bijvoorbeeld: PyCharm.

▼ Input en Output
P1L1

```
naam = input("Naam? ")
print("Hallo, " + naam)
```

▼ Strings Samenvoegen
P1L1 & P1L7

Met '+' teken:

```
aantal_lessen = 10
les = "Python"
print("Ik heb " + str(aantal_lessen) + " lessen " + les + " gevolgd.")
```

Met komma's in print:

```
aantal_lessen = 10
les = "Python"
print("Ik heb", aantal_lessen, "lessen", les, "gevolgd.")
```

Met f-strings:

```
aantal_lessen = 10
les = "Python"
print(f"Ik heb {aantal_lessen} lessen {les} gevolgd.")
```

▼ Variabelen en Types
P1L2

Type	Wat is het?	Voorbeeld
String	Een stuk tekst	"Taart"
Integer (= int)	Een geheel getal	10
Float	Een kommagetal	5.01
Boolean	True of False	False

Types omzetten: `int()`, `str()`, `float()`

▼ Operators
P1L2

Operator	Naam	Voorbeeld
+	Optellen	5 + 2 → 7
-	Aftrekken	5 - 2 → 3
*	Vermenigvuldigen	5 * 2 → 10
/	Delen	5 / 2 → 2.5
//	Gehele deling (= floor division)	5 // 2 → 2
**	Machtsverheffing	5 ** 2 → 25
%	Modulus (= rest bij deling)	5 % 2 → 1

▼ Functies
P1L2

```
def begroet_vriend(bericht, naam="vriend"):
    begroeting = bericht + ", " + naam + "!\nWat leuk om je te zien."
    print(begroeting)

begroet_vriend("Gegroet")
begroet_vriend("Eeeuuy", "Babs Blijbek")

# Uitkomst:
# Gegroet, vriend!
# Wat leuk om je te zien.
# Eeeuuy, Babs Blijbek!
# Wat leuk om je te zien.
```

💡 "\n" is een speciaal karakter dat een nieuwe regel aangeeft in de tekst.

▼ Selecties: if - elif - else
P1L3

```
getal = int(input("Voer een getal in: "))

if getal > 0:
    print("Het ingevoerde getal is positief.")
elif getal < 0:
    print("Het ingevoerde getal is negatief.")
else:
    print("Het ingevoerde getal is nul.")
```

▼ Lussen (Iteraties)
P1L3 & P1L5

For-lussen:

```
for i in range(10):
    # deze lus wordt 10 keer uitgevoerd, waarbij i telkens 0, 1, 2, ..., 9 is

for item in lijst:
    # code die uitgevoerd wordt voor elk element in de lijst
```

While-lussen:

```
while voorwaarde:
    # zolang de voorwaarde (=conditie) True (waar) is, bv. while teller < 5

while True:
    # Oneindige uitvoering, zie ook "break".
```

Geneste lussen:

```
for i in range(5):       # 0 t/m 4
    for j in range(10):  # 0 t/m 9
        print(i, j)
# Uitkomst: 0 0, 0 1, 0 2, ..., 4 8, 4 9
```

▼ Break
P1L5

`break` wordt gebruikt om een lus te stoppen.

```
mijn_lijst = [1, 3, 5, 7, 9, 10, 11, 13, 15]
for nummer in mijn_lijst:
    if nummer % 2 == 0:
        print("Het eerste even getal is gevonden:", nummer)
        break
```

💡 Gelijkaardig: **`continue`**, dat de huidige iteratie overslaat en verdergaat met de volgende.

▼ Modules
P1L3 & P1L6

Modules bevatten Python-code (bv. functies) die je kunt importeren in je eigen programma.

Importeer de volledige module

```
import time
time.sleep(2)
```

Importeer specifieke functies

```
from time import sleep
sleep(2)
```

Importeer alles uit de module

```
from time import *
sleep(2)
```

▼ Arrays (Lijsten)
P1L4

Lijsten worden gemaakt met vierkante haken, bv. `getallen = [0, 2, 4, 6, 8]`
Een element of een deel van de lijst selecteren kan met **slicing**:

```
woorden = ["appel", "banaan", "citroen", "druif", "kiwi", "mango"]
print(woorden)      # ["appel", "banaan", "citroen", "druif", "kiwi", "mango"]
print(woorden[:3])  # ["appel", "banaan", "citroen"]
print(woorden[2:4]) # ["citroen", "druif"]
print(woorden[2:])  # ["citroen", "druif", "kiwi", "mango"]
```

Methode	Beschrijving
lijst.append(element)	Voegt een element toe achteraan de lijst.
lijst.count(element)	Geeft terug hoe vaak dat element voorkomt in de lijst.
lijst.extend(andere_lijst)	Zet alle elementen van een lijst achter de huidige lijst.
lijst.index(element)	Geeft de index van het eerste element met een bepaalde waarde.
lijst.insert(positie, element)	Zet een element op een bepaalde positie.
lijst.pop(positie)	Verwijdert een element op een bepaalde positie, en geeft het ook terug.
lijst.pop()	Verwijdert het laatste element, en geeft het ook terug.
lijst.remove(element)	Verwijdert een element met een bepaalde waarde.
del lijst[index_nummer]	Verwijdert het element op de opgegeven index. Bv. del lijst[0] → verwijdert het eerste element.
lijst.reverse()	Draait de lijst om.
lijst.sort()	Sorteert de lijst.
len(lijst)	Geeft de lengte van de lijst terug (= aantal elementen in de lijst)
lijst[index_nummer]	Geeft het element op een bepaalde index terug. Bv. lijst[0] → eerste element.

▼ Tuples
P1L7

Een tuple is overanderlijke lijst van data. Ze worden gemaakt met ronde haken.

```
weekdagen = ("maandag", "dinsdag", "woensdag", "donderdag", "vrijdag")
```

Sommige functies van lijsten werken ook op tuples, zoals `len()` en `index()`.

▼ Dictionaries
P1L7

Een dictionary is een collectie van data die bestaat uit key-value paren. De sleutels zijn uniek.

```
woordenboek = {
    "huis": "Een gebouw waar mensen in wonen.",
    "één": 1,
    "CodeFever": "De leukste hobby!"
}
```

Voorbeeld	Beschrijving
woorden["peer"] = "fruit"	Key-value paar toevoegen, of bestaande waarde wijzigen
del woorden["appel"]	Iets verwijderen uit de dictionary
print(woorden["CodeFever"])	Een element opvragen en printen
if "banaan" in woorden:	Controleren of een key aanwezig is in de dictionary

▼ Random
P1L8

Met de module random kan je willekeurige dingen maken in je programma.

```
import random
kleuren = ("rood", "blauw", "geel", "groen")
print(random.choice(kleuren))             # Willekeurige kleur

dobbelsteen_worp = random.randint(1, 6)   # Int tussen 1 en 6
punten = random.uniform(0, 10)            # Float tussen 0 en 10
```

▼ Bestanden lezen
P1L8

```
bestand = open("files/tekstbestand.txt", "r")
for regel in bestand:
    print(regel)
```

▼ RGB
P1L9

RGB staat voor Rood, Groen, Blauw. Enkele voorbeelden:

```
rood = (255, 0, 0)
groen = (0, 255, 0)
blauw = (0, 0, 255)
oranje = (255, 150, 0)
```

💡 Dit zijn telkens tuples met de drie getallen in. Elk getal ligt tussen 0 en 255.

▼ Pygame
P1L9 & P1L10

Pygame is een bibliotheek die je kan gebruiken om games te maken in Python. De event loop is een oneindige lus die luistert naar gebeurtenissen en acties uitvoert.

```
import pygame, sys
from pygame.locals import QUIT

pygame.init()
scherm = pygame.display.set_mode((800, 600))
pygame.display.set_caption("Mijn eerste game")

while True:
    for event in pygame.event.get():
        if event.type == QUIT:
            pygame.quit()
            sys.exit()
        pygame.display.update()
```

▼ Scoping
P1L10

Scoping gaat over de zichtbaarheid van variabelen in je programma.
Lokale scope: variabelen die je binnen een functie definieert en enkel daar beschikbaar zijn.
Globale scope: variabelen die je buiten je functies definieert en "overal" beschikbaar zijn.

```
def functie():
    x = 10
    print(x)

functie()
# Dit zal een foutmelding geven
print(x)
```

💡 Als je een globale variabele wilt **veranderen** in een functie, moet je **`global variabele`** gebruiken.