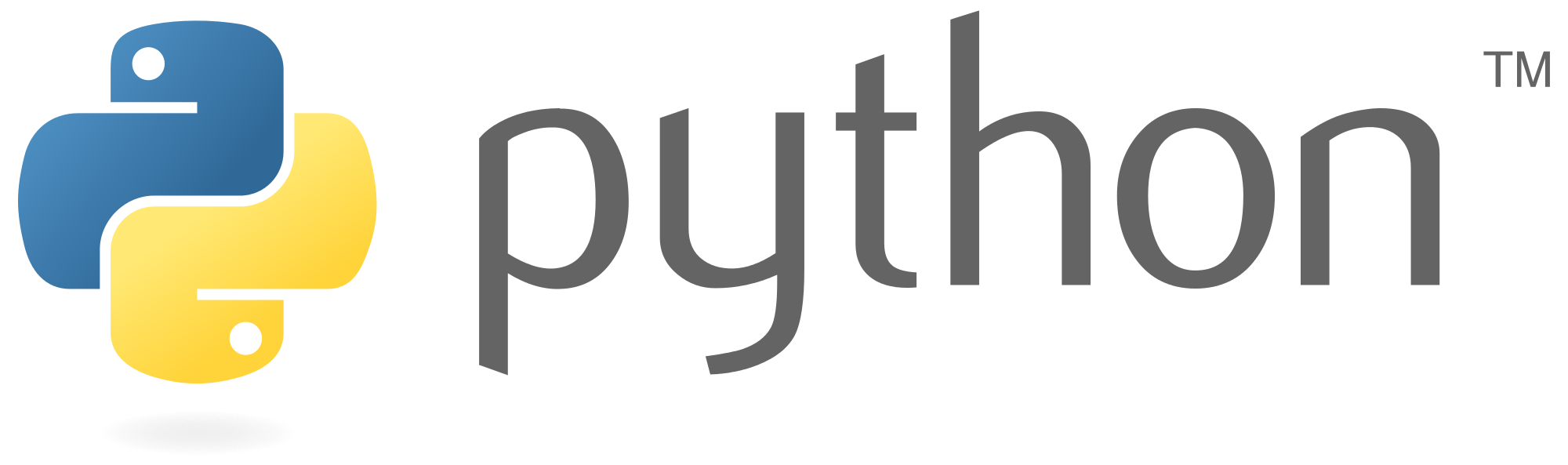
Python [Afbeeldingsresultaat voor python](https://www.google.nl/url?sa=i&rct=j&q=&esrc=s&source=images&cd=&cad=rja&uact=8&ved=2ahUKEwiB5pHhzKPcAhXIZ1AKHVf0BugQjRx6BAgBEAU&url=https://www.at-automation.nl/python-programmeertaal/&psig=AOvVaw3IrQ4sZ2dt4DCSqGOlZuy3&ust=1531829768337304)



Python is een zeer sterke en vaak voorkomende programmeertaal.

Als je je bijvoorbeeld wilt verdiepen in het gebruik van drones is deze taal zeer geschikt.

Verder is Python een programmeertaal waar je het programmeren goed onder de knie krijgt, mocht je later met een specifieke taal aan de slag gaan heb je heel veel basisprincipes van het programmeren door.

Deze cursus heeft niet alleen tot doel om wat opgaven/opdrachten te maken maar veel meer om je te laten denken als een programmeur, en hoe een computer een programma uitvoert.

Een voorbeeldje : Stel je hebt een stapel kaarten en je moet deze van hoog naar laag

sorteren.

Voor ons is dat vrij duidelijk maar een computer snapt er niets van

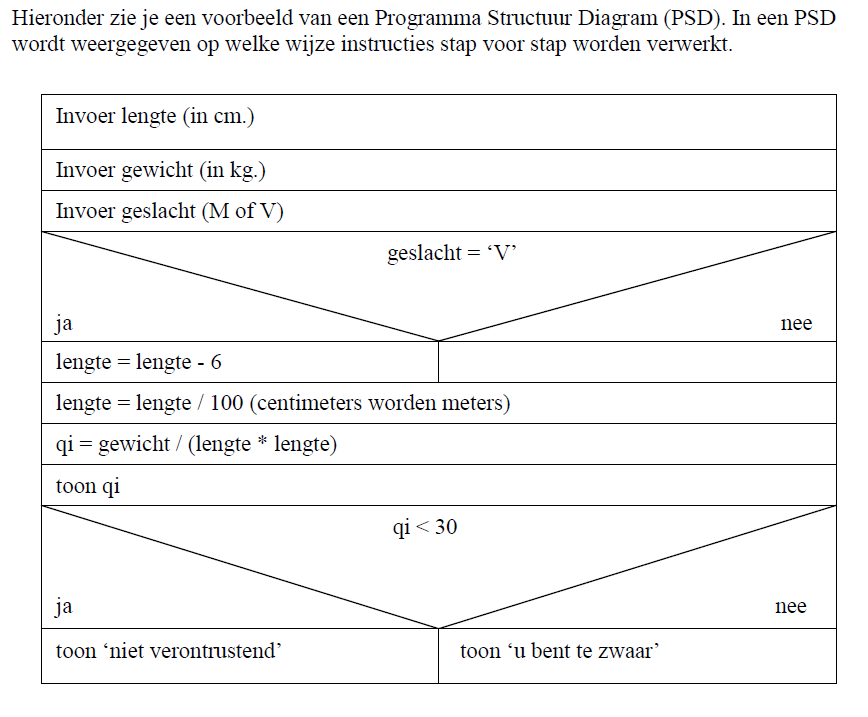
Je moet een programma schrijven wat heel duidelijk maakt welke

stappen er moeten gebeuren om deze taak uit te voeren.

Een programma maken/schrijven kun je zelden uit je hoofd, er zijn te veel stappen die gedaan moeten worden en het is onmogelijk om dit uit je hoofd te doen.

Het beste middel is om een zgn PSD ( Programma Structuur Diagram )

Op de volgende pagina staat een voorbeeldje



Kijk eens of je begrijpt wat deze PSD moet uitvoeren

Een PSD bestaat uit 3 verschillende onderdelen nl : Rechthoeken

Driehoeken

Teksten

Om een programma goed te laten funtioneren en zeker om het duidelijk te maken moet je een ***algoritme*** bedenken.

Een algoritme bestaat uit de volgende stappen : Probleemstelling

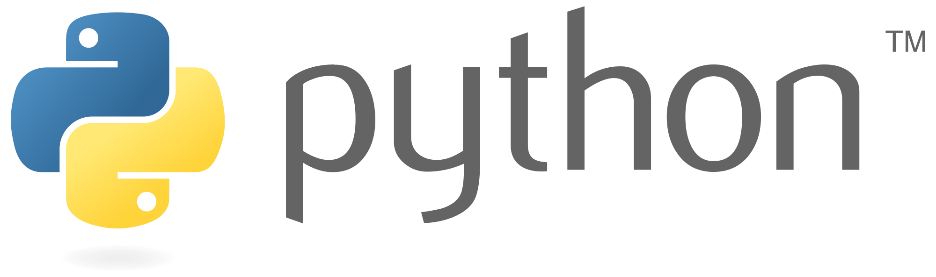
Analyse

Schematische voorstelling ( PSD )

Het schrijven van de broncode

Compileren en testen

Het inleveren van de opg 1 t/m 23 levert 25 punten op

[](https://www.google.nl/url?sa=i&rct=j&q=&esrc=s&source=images&cd=&cad=rja&uact=8&ved=2ahUKEwiB5pHhzKPcAhXIZ1AKHVf0BugQjRx6BAgBEAU&url=https://www.at-automation.nl/python-programmeertaal/&psig=AOvVaw3IrQ4sZ2dt4DCSqGOlZuy3&ust=1531829768337304)

Als je Python opstart krijg je in principe 2 schermen , een zgn IDLE en een editor

In de editor kun je programma’s schrijven , opslaan , uitvoeren enz

( vorig jaar heb je bij HTML ook gewerkt met een editor nl Notepad ++ )

Je zou deze ook kunnen gebruiken.

In de IDLE kun je 1 commando geven en deze wordt dan direct uitgevoerd.

Bijvoorbeeld : **print** (7/4 ) en druk op enter dan krijg je als resultaat 1.75 op het scherm.

De opdracht ( of straks het programma ) wordt uitgevoerd in de zgn “Python shell”

En nu beginnen :

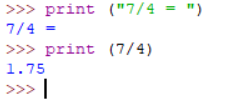
Maak een map aan op je USB: schijf met de naam *mijnpython* en sla al je pythonbestanden hier op

Opg 1 voer in de IDLE de volgende 2 opdrachten uit : print ( “7/4 = “) en print ( 7/4 )

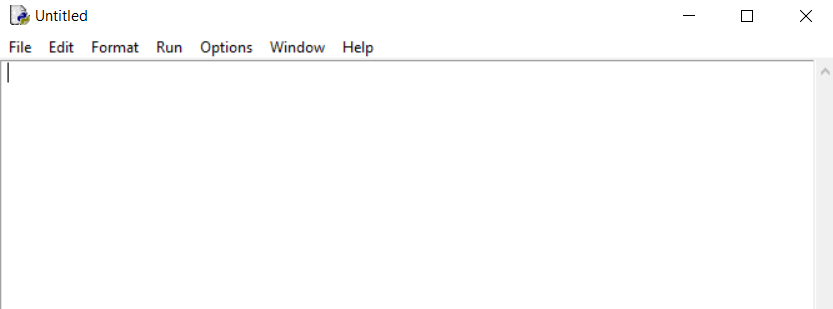
Kijk uit met kopieren want de editor neemt spaties enz niet mee en krijg je

foutmeldingen

Dit zou het resultaat moeten zijn



Opg 2 Ga nu naar File en kies New File



Je bent nu in de editor en typ het volgende in :

print ( "7/4 = " , 7/4 )

Ga dan naar File en kies Save As

Ga vervolgens naar je net gemaakte map en sla dit bestand op als ***opg 2***

Om het programma uit te voeren kies je Run en dan Run Module F5

Je programma wordt nu uitgevoerd en op het scherm verschijnt :



De komma in opdracht 2 zorgt ervoor dat je in de uitvoer op je scherm dingen naast elkaar kunt zetten.

Dus in ons geval eerst de tekst 7/4 ( tussen aanhalingstekens ) en dan het antwoord van 7/4

Er zijn verschillende soorten **data types** ( resultaten die op het scherm komen) nl :

Strings Een stuk tekst omsloten door aanhalingstekens

Bv. “ hier zijn 2 mango’s “

Integers Gehele positieve of negatieve getallen of nul

Bv. 3 of -5 of 0 of 123456

Floats Gebroken getallen of getallen met een komma ( in Python een punt)

Bv. 3.1415 of 18.0

Met deze 3 data types kun je verschillende expressies ( berekeningen ) uitvoeren

Een aantal van deze expressies zijn : (+) optelling

(-) aftrekking

(\*) vermenigvuldiging

(/) deling

(//) integer deling ( deling die naar beneden

afrond op een heel getal)

(\*\*) machtsverheffing

(%) modulo ( is de rest bij een deling )

Tot slot nog even dit : Soms moet je een datatype veranderen in een ander datatype.

Dat kun je doen met **type casting** functies

***Int()*** geeft als uitkomst een geheel getal naar beneden afgerond

***float()*** geeft als uitkomst een komma getal

***str()*** geeft als uitkomst een tekst

En dan willen we vaak een commentaarregel in een programma toevoegen zodat de programmeur weet wat er in het volgende stukje code gaat gebeuren

Het is dus een stuk tekst wat niet door het programma wordt uitgevoerd

Hiervoor gebruik je het # teken

Bv . # als dit in je code staat doet het programma er niets mee

Opg 3. Schrijf een programmaatje voor het volgende en save het onder de naam opg3

Een DVD kost € 24,95 in een webshop. Maar de webshop zelf krijgt 40% korting

bij hun inkoop . Het verschepen van DVD’s kost €3, voor het de eerste DVD en €0,75

voor ieder volgende DVD.

Hoeveel winst maak de webshop als hij 60 DVD’s verkoopt ?

( antwoord € 551,55 )

Opg 4. Schrijf een programmaatje voor het volgende probleem en save het onder de naam

opg4

Het in nu 14.30uur en je stelt een alarm in dat af moet gaan over 318 uur

Hoe laat gaat het alarm dan af.

( gebruik de modulo operator )

( antwoord 20.30 uur maar op je scherm zie je 20.3 , hier komen we later op terug)

VARIABELEN

In opg 3 heb je een programma geschreven waar je de winst voor 60 DVD’s moest uitrekenen. Je kunt je voorstellen dat je dat ook voor 50 of 800 of 3021 moet doen

Om dan iedere keer je programma te wijzigen gaat veel te veel tijd kosten en iemand die niet weet hoe je moet programmeren zit helemaal in de knoei.

Nu komt de variabele goed te pas.

Een variabele is een stukje geheugen in de computer die je zelf een naam geeft. Daar kun je iets opslaan. Vergelijk het maar met een kast met een enorme hoeveelheid laden.

Ieder lade heeft een naam en daar kan iets in zitten

De naam van een variabele mag je zelf bedenken maar niet alles mag

Een variabele mag 1. Alleen bestaan uit letters , cijfers en “underscores” ( \_ )

2. Moet altijd beginnen met een letter of underscore

3. Mag geen gereserveerd woord zijn ( zie hieronder )

False class finally is return

None continue for lambda try

True def from nonlocal while

and del global not with

as elif if or yield

assert else import pass

break except in raise

( Kijk goed naar de hoofdletters want Python maakt verschil tussen hoofd

en kleine letters )

2 voorbeelden :

***x = 5***

***print (x )***

Hier wordt eerst een variabele met de naam x gecreerd en in deze variabele wordt het getal 5 geplaatst

Verder zet print ( x ) het getal 5 op het scherm en niet de letter x !!

***x=2***

***y=3***

***print ( “x = “ , x )***

***print ( “y = “ , y )***

***print ( “x + y = “ , x + y )***

Bij dit programma wordt de waarde 2 in x en de waarde 3 in y geplaatst

Op het scherm verschijn : x = 2

y = 3

x + y = 5

*Gebruik altijd namen voor je variabelen die zinnig zijn*. ( zie hieronder )

***a = 3.14159265***

***b = 7.5***

***c = 8.25***

***d = a\*b\*b\*c/3***

***print (d)***

Als je dit ziet weet je hoogstwaarschijnlijk dat er iets uitgerekend wordt maar je weet niet wat

Je kunt het ook zo maken en dan is het een stuk duidelijker

***pi = 3.14159265***

***straal = 7.5***

***hoogte = 8.25***

***volume\_van\_de\_kegel = pi \* straal \* straal \* hoogte / 3***

***print ( volume\_van\_de\_kegel )***

Dit is wat beter niet waar ?

Tot slot kun je commentaar aan het begin zetten over wat er gaat gebeuren

***# Berekenen van het volume van een kegel met straal 7.5 en hoogte 8.25***

***pi = 3.14159265***

***straal = 7.5***

***hoogte = 8.25***

***volume\_van\_de\_kegel = pi \* straal \* straal \* hoogte / 3***

***print ( volume\_van\_de\_kegel )***

Opg 5 Kun je ontdekken wat er in het volgende programma fout gaat ?

Zo niet schrijf dit programma eens kijk wat je voor foutmelding krijgt

Verbeter het programma zodat het wel werkt. ( er zijn veel mogelijkheden)

***nr1 = 5***

***nr2 = 4***

***nr3 = 5***

***print ( nr3 / ( nr1 % nr2 ) )***

***nr1 = nr1 + 1***

***print ( nr3 / ( nr1 % nr2 ) )***

***nr1 = nr1 + 1***

***print ( nr3 / ( nr1 % nr2 ) )***

***nr1 = nr1 + 1***

***print ( nr3 / ( nr1 % nr2 ) )***

( kijk uit met het kopieren van dit programma want er worden spaties , enters, en

tabs meegenomen en deze worden niet herkend )

Je ziet verder dat er een aantal keer **nr1 = nr1 + 1** staat. Dit betekend dat de waarde van nr1 steeds met 1 opgehoogd wordt

Je kunt dat ook korter schrijven door **nr1 += 1**

Dan weet je ook wat ***nr1 -= 1*** gaat doen

opg 6 Hieronder staat een deel van een programma wat 2 variabelen omwisselt zonder

gebruik te maken van een derde variabele.

Je moet nog 2 regels toevoegen. Doe dit en bekijk het resultaat

***a = 17***

***b = 23***

***print ( “a = “ , a , “en b= “ , b )***

***a += b***

***# Nu komen jou 2 regels***

***……………………………………………***

***……………………………………………***

***print ( “nu heeft a de waarde “ , a , “en b= “ , b )***

Eenvoudige Funcies

In het voorgaande gedeelte heb je al kennis gemaakt met een aantal eenvoudige functies zoals **print()** en **int()**

In dit gedeelte komen er een paar nieuwe bij en over een paar hoofdstukken leer je zelf nieuwe functies maken

Een functie is een blok code wat een actie uitvoert

Om een functie aan het werk te zetten moet je hem aanroepen met **“call”**

Je moet 3 dingen weten van de functie nl :

De naam van de functie

De parameters die de functie nodig heeft ( als die er zijn )

De waarde die de functie teruggeeft ( als die er is )

DE NAAM : Iedere functie heeft een naam. Daar gelden dezelfde regels voor als bij de

variabelen.

PARAMETERS parameters zijn waarden die je aangeeft aan een functie om te verwerken

Een parameter geeft dus eigenlijk een waarde door

Voorbeeld : ***x = 1.56***

***print ( int (x ) )***

***print ( x )***

Als je dit uitvoert zie je dat er niets met de waarde van x gebeurt

Het programma geeft eerst een 1 en daarna gewoon 1.54

Voorbeeld : ***basis = 2***

***exponent = 3***

***print ( pow ( basis , exponent ))***

In dit voorbeeld geef je twee parameters mee nl basis en exponent ( 2 en 3 )

De functie pow() staat voor machtsverheffen ( power ), dus hier wordt 23 = 8

DE RETOURWAARDE ook wel de teruggeefwaarde

Voorbeeld : ***x = 2.1***

***y = “3”***

***z = int ( x )***

***print ( z )***

***print ( int ( y ) )***

In de laatste regel krijgt de print() functie als waarde de aanroep van

int() functie

Klinkt allemaal heftig maar als je dadelijk gaat programmeren zul je merken dat het meevalt

Enkele basisfuncties

* float ( ) is al besproken
* int ( ) is al besproken
* str ( ) is al besproken
* abs ( ) geeft altijd een positieve waarde terug ***print (abs( 1 )) geeft 1***

***print (abs( -1)) geeft 1***

* max() geeft het grootste getal terug  ***print (max(1,2,3)) geeft 3***
* min() geeft het kleinste getal terug ***print (min(1,2,3,4)) geeft 1***
* pow() is al besproken
* round() geeft de wiskundige afronding ***print(round(1.27,1)) geeft 1.3***

rond 1.27 af op 1 cijfer achter

de komma

* len() geeft het aantal tekens in een string ***print ( len(”man go”)) geeft 6***

ook de spatie is een teken

* input() geeft een string terug die de gebruiker

intypt tijdens een lopend programma ***tekst = input(“typ wat in “)***

***print(“je hebt “,tekst,” getypt”)***

wat je ingetypt hebt komt op het scherm

als je op enter hebt gedrukt

kijk uit als je met waarden gaat werken

want je voert een string in

vb : ***getal = input ( “geef een getal “)***

***print ( “het kwadraat is “, getal \* getal )***

dit levert een foutmelding op want getal is een string en geen getal !!

***getal = input ( “geef een getal “)***

***getal = float ( getal)***

***print ( “het kwadraat is “, getal \* getal )***

het bovenstaande werkt beter

opg 7. Schrijf een programma waarbij de gebruiker 2 getallen moet invoeren en als

uitkomst geeft je programma de som en het verschil van deze getallen

* print() ook aan print() kun je parameters toewijzen, nl ***sep*** , ***end*** en ***format***

opg 8. Neem onderstaande over en bekijk het resultaat

print (“X” , ”X” , ”X”, sep = “x”)

print (“X” , end = “” )

print (“Y”, end = “”)

print (“Z” )

Op het format gebeuren komen we nog terug ( als je het wilt weten : effe googlen )

MODULES

We hebben nu een aantal basis functies besproken maar er zijn er veel meer.

Om geen al te grote chaos te krijgen zijn deze verdeelt in een aantal modules

Bv om wat wiskundige berekening te doen heb je de module MATH nodig

Er zijn ook teken modules en nog een aantal maar als het zover is dan merk je het wel

Om van een module gebruik te maken moet je deze eerst importeren

Dit komt altijd boven in je programma te staan

Bijvoorbeeld : ***import math***

***print (math.sqrt (4))***

In dit voorbeeld roep je de module math aan

Vervolgens zet je programma √4 = 2 op het scherm

( als je alleen in je programma werkt met √ en geen andere wiskunde termen kun je ook het volgende gebruiken. Je hoeft dan bij print geen aanroep te doen op math )

***from math import sqrt***

***print ( sqrt( 4 ))***

Er zijn heel veel wiskundige functies in de math module. We zullen ze niet allemaal bespreken dan moet je maar googlen.

Een aantal veel voorkomende functies zijn : **exp () , log () , log(10) , sqrt ()**

Een andere veel gebruikte module is de RANDOM module.

Deze genereert toevalsgetallen. ( Nou ja, een digitale computer genereert nooit echte toevalsgetallen omdat hij gebruik maakt van formules , tijdklokken , data enz )

**random()** geeft een float getal tussen de 0 en 1 waarbij 0.0 wel meedoet en 1.0 niet

**randint()** moet 2 parameters krijgen waarbij het eerste getal kleiner moet zijn dan de

tweede.

Bv ***random***.***randint ( 2,5 )*** kan de waarden 2,3,4 of 5 geven

**seed()** seed() initialiseert de toevalsgenerator van Python. Als je een lijst van toevals-

getallen een aantal keren in je programma nodig hebt kun je dat voor elkaar

krijgen door aan het begin van je programma seed() aan te roepen met een

vast getal. Bv ***seed (0 )***

voorbeeld : import random

seed()

print ( “een willekeurig getal tussen 1 en 10 is “,

random.randint( 1 , 10 ))

print ( “een ander is “) , random.randint(1,10 )

seed(0)

print(“3 toevalsgetallen zijn ook “,

random.random(),random.random(),random.random() )

seed(0)

print(“3 toevalsgetallen zijn ook “,

random.random(),random.random(),random.random() )

seed(0)

print(“Dezelfde 3 zijn “, random.random(),random.random(),

random.random() )

opg 9. Maak een programma waarbij de gebruiker 3 getallen invoert.

Je programma geeft vervolgens de grootste, kleinste en het gemiddelde van deze aan

Opg 10 Maak een programma waarbij de gebruiker 2 lengtes de rechthoekszijden van een

rechthoekige driehoek invoert, en het programma berekent dan de lengte van de

hypotenusa ( schuine zijde J )

CONDITIES

In een programma moeten er vaak beslissingen genomen onder bepaalde voorwaarden

Bv een app op een smartphone geeft een waarschuwing als de batterij minder dan

5% vol is.

Dat betekent dat de app test of een zekere variabele ( bv batterij\_energie ) kleiner

dan 5 is. Als dat zo is ( TRUE ) geef dan een waarschuwing, anders ( FALSE ) doe niets

In Python wordt “waar” weergegeven met TRUE en “onwaar” met FALSE. Dit zijn zgn “booleans”.

De meest gebruikte boolean expressies zijn vergelijkingen. Een vergelijking bestaat uit 2 waardes met een operator ertussen.

Operatoren zijn : < kleiner dan

<= kleiner dan of gelijk aan

== is gelijk aan

>= groter dan of gelijk aan

> groter dan

!= is niet gelijk aan

Opg 11. Neem over en voer het programma uit. Kijk goed of je alle uitkomsten snapt

***print ( "1. ", 2 < 5)***

***print ( "2. ", 2 <= 5)***

***print ( "3. ", 3 > 3)***

***print ( "4. ", 3 >= 3)***

***print ( "5. ", 3 == 3.0)***

***print ( "6. ", 3 == "3")***

***print ( "7. ", "syntax" == "syntax")***

***print ( "8. ", "syntax" == "semantiek")***

***print ( "9. ", "syntax" == " syntax")***

***print ( "10. ", "Python" != "rommel")***

***print ( "11. ", "Python" > "Perl" )***

***print ( "12. ", "banaan" < "mango")***

***print ( "13. ", "banaan" < "Mango")***

Je ziet dat je behalve waarden ook teksten kunt vergelijken.

Hoofdletters worden altijd beschouwd als kleiner dan de kleine letters

Een andere operator is de **in** operator. Je kunt testen of een teken of een serie tekens in een string voorkomt.

Opg 12. Neem over en voer het programma uit. Kijk goed of je de uitkomsten snapt

***print ( "y" in "Python")***

***print ( "x" in "Python")***

***print ( "p" in "Python")***

***print ( "th" in "Python")***

***print ( "to" in "Python")***

***print ( "y" not in "Python")***

Nu kunnen de boolean expressies ook gecombineerd worden door logische operatoren

Er zijn er drie nl **and** , **or** en **not**

Conditionele statements

Naast voorgaande statements of condities komen nu de if …. statements aan de orde

Dit gaan we doen ahv een voorbeeldje :

***getal = input ("voer een getal in en druk op enter ")***

***getal = float ( getal )***

***if getal == 5 :***

***print ("getal is 5 ")***

In de eerste regel voeren we een waarde in en deze zetten we in variabele **getal**

In de tweede regel wordt de **string getal** omgezet naar een **float waarde**

In de derde regel kijken we of de waarde van **getal** **gelijk is aan 5** ( dan een : )

Als dit **waar** is print dan getal is 5 op het scherm

Als dit **niet waar** is dan doe niets

*Let goed op : de voorwaarde waar aan moet voldoen ( print(“getal is 5”) ) springt in*

*Gebruik dus een TAB*

Als er meerdere acties uitgevoerd moeten ook deze beginnen met een TAB

In het voorgaande voorbeeld hebben we gezien dat er **getal is 5** op het scherm komt als je ook daadwerkelijk 5 hebt ingevoerd. Als je iets anders invoert komt er niets op het scherm

We gaan nu gebruik maken van de zgn twee-weg belsissingen. Er komt een **else** statement bij

Het programma kan er bv uitzien zoals je op de volgende pagina kunt zien

***getal = input ("voer een getal in en druk op enter ")***

***getal = float ( getal )***

***if getal == 5 :***

***print ("getal is 5 ")***

***else:***

***print(“getal is niet 5”)***

Opg 13 Je kunt nu testen of een ingevoerd getal even of oneven is

Schrijf een programma wat dit doet

Maak gebruik van bovenstaand voorbeeld en gebruik de % operator

Als het goed is weet je nog dat als je een getal door 2 deelt de rest 0 is

In Python : x%2

MEER-WEG-BESLISSINGEN

Het komt voor dat je in de situatie komt dat je niet meer tussen 2 dingen moet kiezen ( TRUE en FALSE ) maar tussen meerdere mogelijkheden.

Python biedt deze mogelijk door gebruik te maken van **elif** ( afkorting van else if )

Bekijk het volgende voorbeeld , het zal wel duidelijk zijn , zo niet dan voer dit programma uit en verander de waarde van leeftijd eens een paar keer

***leeftijd = 21***

***if leeftijd < 12:***

***print (“je bent een kind”)***

***elif lijftijd < 18:***

***print (“je bent een teenager”)***

***elif leeftijd < 30:***

***print (“je bent nog jong”)***

***elif leeftijd < 50:***

***print (“ de eerste grijze haren komen”)***

***else:***

***print(“ bijna met pensioen”)***

opg 14. Bij deze opgave voer je een lengte in meters in, en een gewicht in kilo’s

Vervolgens rekent het programma je BMI uit met de formule :

GEWICHT

BMI = ----------------------------- ( dit is nog de oude formule)

LENGTE x LENGTE

Daarna geeft het programma aan of je ondergewicht hebt – BMI < 18

gezond bent ------------- BMI 18-25

overgewicht hebt -------BMI 25-30

obesitas hebt ------------BMI 30- 40

Maak eerst een PSD !!!!!

Je mag ook de nieuwe formule gebruiken , die staat op de volgende pagina

GEWICHT x 1,3

BMI = ---------------------------------------

(LENGTE) 2,5

Geneste condities

Binnen een if … elif …. else statement kun je ook weer een if .. elif..else statement aanroepen. Dit heet dan een geneste conditie

Kijk eens naar het volgende voorbeeld. Hier wordt gekeken of een getal deelbaar is door 7 en 11

***x = input (“ voer een getal in “)***

***x = int ( x )***

***if x%7 ==0:***

# --- hier begint een geneste code –

if x%11==0:

print(x, “is deelbaar door 7 en 11 “)

else:

print(x,”is deelbaar door 7 maar niet door 11”)

# --- hier eindigt de geneste code ---

***elif x%11 ==0:***

***print(x , “is deelbaar door 11 maar niet door 7”)***

***else:***

***print(x,”is niet deelbaar door 7 of 11”)***

opg 15. Bij tweedegraads vergelijkingen ( Ax2 + Bx + C = 0 ) kun je 0,1 of 2

oplossingen krijgen

Dit kun je berekenen met de Discriminant. ( D = B2 – 4AC )

Maak een programma waarbij je de waarde van A,B en C invoert en

Het resultaat weergeeft hoeveel oplossingen er zijn.

( Voor de gevorderden : laat ook de uitkomsten zien )

Herhalingen

Stel je eens voor dat je 500 getallen moet invoeren, dan kun je een programma maken met ongeveer 1000 regels.

***getal1 = input(“geef getal “)***

***getal1 = int(getal1)***

***getal2 = input(“geef getal”)***

***getal2 = int(getal2***)

……………. Enz

Onbegonnen werk, voor dit soort werk kun je een **loop** gebruiken

Er zijn er een aantal en we beginnen met de **while-loop**

Een voorbeeldje om de nummers 1 t/m 100 te printen

***nummer = 1***

***while nummer <= 100 :***

***nummer = int(nummer)***

***print ( nummer )***

***nummer +=1***

***print (“klaar”)***

Hieronder staat het stroomdiagram :

start

nummer = 1

nummer <= 100 True

False print (nummer)

Nummer +=1

Print (“klaar”)

Stop

Kijk eens of je het programma en het stroomdiagram door hebt. Zo niet kom even langs

Stel je nu eens voor dat je niet weet hoeveel getallen je moet invoeren maar dat het programma geen getallen meer vraagt als de 0 invoert

Dan kun je het volgende gebruiken :

***nummer = -1***

***totaal = 0***

***while nummer != 0 :***

***nummer = input(“geef een nummer “)***

***nummer = int( nummer )***

***totaal += nummer***

***print ( “het totaal = “ , totaal )***

opg 16 Maak een programma waarbij de gebruiker een getal invoert

Het programma telt dan af met 1 en stopt bij 0 en geeft die reeks getallen op het

scherm

Maw als je 5 invoert komt op het scherm 5,4,3,2,1.

Maak een stroomdiagram

Voor de gevorderden onder ons , druk alleen de oneven getallen af

Tot slot nog even dit : Kijk uit dat je niet in een eindeloze loop komt ( zie hieronder )

***getal = input (“voer een getal in “)***

***getal = int ( getal)***

***foutje = 5***

***while foutje ==5:***

***print (getal)***

***getal +=1***

***print (“einde”)***

Er gebeurt niets met de variabele **foutje**. Deze blijft altijd 5 , en de variabele **getal** blijft iedere keer 1 ophogen

Een andere loop is de **for-loop**. For-loops zijn gemakkelijker te gebruiken maar kunnen minder dan de while-loops.

Een voorbeeldje van een for-loop met een string

***for letter in “banaan”:***

***print(letter)***

***print (“klaar”)***

Als je dit programma uitvoert dan zie je dat Python het woord *banaan* als een reeks letters beschouwd met de letter *b* als eerste en de letter *n* als laatste.

Dus eerst wordt de letter ( *b )* geprint. Het **for .. in** statement hoogt zelf een tellertje met 1 op en dan wordt de tweede letter ( *a* ) geprint.

Dit gaat net zolang door tot dat de laatste letter ( *n* ) geprint is

Dus op je scherm staat dus b

a

n

a

a

n

Wel handig in de for-loop is de **range()** functie.

Neem het volgende over en kijk wat de uitvoer op je scherm is

***for x in range(10):***

***print(x)***

Kijk ook eens naar het volgende :

***for x in range(10,20):***

***print(x)***

En ook deze :

***for x in range(1,15,3):***

***print(x)***

Je kunt ook zelf een range bepalen , zoiets noemen we een *tuple*

***for x in (“appel”, “peer”, “druif”, “banaan”, “mango”, “kers” ) :***

***print (x)***

Je hebt bij de loops gezien dat ze een aantal keer iets achter elkaar doen maar daarna is het afgelopen. Vaak willen we dat het programma verder gaat met andere opdrachten of dat we zelfs tijdens een loop iets anders willen doen.

Er zijn 3 mogelijkheden om dat voor elkaar te krijgen : **else , break en continue**

Eerst het else statement : ***i = 0***

***while i < 5 :***

***print (i)***

***i +=1***

***else :***

***print(“de loop eindigt , i is nu “, i )***

***print (“klaar”)***

Vervolgens het break statement :

Het break statement maakt het mogelijk om een loop voortijdig af te breken.

Als python een break tegenkomt stopt deze tijdens het verwerken van een loop en keert **niet** terug naar de eerste regel van de loop

Een voorbeeld : Deze code controleert een cijferlijst van een leerling. Als alle cijfers 5.5 of hoger zijn is deze leerling geslaagd, maar als er 1 of meer lager dan 5.5 zijn is hij/zij gezakt

***for cijfer in ( 8, 7.5 , 9 , 6 , 6 ,6 ,5.5 , 7 , 5 , 8 , 7 , 7.5 ) :***

***if cijfer < 5.5 :***

***print (“leerling is gezakt “)***

***break***

***else :***

***print (“leerling is geslaagd”)***

opg 17 Neem deze over en zie dat deze student is gezakt

Verwijder nu de 5 en zie dat de leerling is geslaagd

Verwijder ook de break eens in je programma en kijk eens wat er dan gebeurt

En tot slot het continue-statement : Als een continue statement in een loop wordt

aangetroffen, wordt onmiddellijk het uitvoeren van de

cyclus in deze loop beeindigd en teruggekeerd naar de

eerste regel van de loop

Voorbeeld : ***num = 0***

***while num < 100 :***

***num +=1***

***if num%2 == 0 :***

***continue***

***if num%3 == 0 :***

***continue***

***if num%10 == 9 :***

***continue***

***print ( num )***

Dit programma drukt alle getallen af tussen 1 en 100 die niet door 2 of 3 gedeeld kunnen worden en die niet op een 9 eindigen

Geneste loops

In een loop kun je weer een andere loop opnemen maar kijk er mee uit.

Als je het door hebt bespaart het je veel tijd maar er kan een boel mis gaan

***for x in range (3):***

***for y in range (5):***

***print (" (x,y ) = (" , x," , ",y, ") " )***

Dit levert het volgende op :

(x,y ) = ( 0 , 0 )

(x,y ) = ( 0 , 1 )

(x,y ) = ( 0 , 2 )

(x,y ) = ( 0 , 3 )

(x,y ) = ( 0 , 4 )

(x,y ) = ( 1 , 0 )

(x,y ) = ( 1 , 1 )

(x,y ) = ( 1 , 2 )

(x,y ) = ( 1 , 3 )

(x,y ) = ( 1 , 4 )

(x,y ) = ( 2 , 0 )

(x,y ) = ( 2 , 1 )

(x,y ) = ( 2 , 2 )

(x,y ) = ( 2 , 3 )

(x,y ) = ( 2 , 4 )

Opg 18 schrijf een programma dat de gebruiker een getal laat invoeren.

Het programma laat de tafel van vermenigvuldiging zien voor dat getal van 1 t/m 10

Dus , als de gebruiker 5 intypt komt op het scherm :

**geef een getal 5**

**1 \* 5 = 5**

**2 \* 5 = 10**

**3 \* 5 = 15**

**4 \* 5 = 20**

**5 \* 5 = 25**

**6 \* 5 = 30**

**7 \* 5 = 35**

**8 \* 5 = 40**

**9 \* 5 = 45**

**10 \* 5 = 50**

**Voor de liefhebber :**

Opg 19 De Fibonacci reeks is een serie getallen dat start met een 1 gevolgd door nog een 1

Ieder volgend getal is de som van de voorgaande 2 getallen

Dus 1,1,2,3,5,8,13,21, …………………… Schrijf een programma tot dat 1000 is bereikt

Opg 20 In opg 18 heb je een tafel van vermenigvuldiging gemaakt

In deze opgave moet je een vermenigvuldig-matrix maken

Dus als je het getal 9 ingeeft moet dit het resultaat worden :

. | 1 2 3 4 5 6 7 8 9

---------------------------------------

1 | 1 2 3 4 5 6 7 8 9

2 | 2 4 6 8 10 12 14 16 18

3 | 3 6 9 12 15 18 21 24 27

4 | 4 8 12 16 20 24 28 32 36

5 | 5 10 15 20 25 30 35 40 45

6 | 6 12 18 24 30 36 42 48 54

7 | 7 14 21 28 35 42 49 56 63

8 | 8 16 24 32 40 48 56 64 72

9 | 9 18 27 36 45 54 63 72 81

Bij deze opgave kun je gebruik maken van het format-statement

Dit is om de getallen netjes onder elkaar te krijgen

**print ( “ { :.3f } “.format ( 7/11) )** doet bijna hetzelfde **als print ( round ( 7/11 , 3) )**

Ze delen 11 door 7 en ronden af op 3 decimalen , alleen de eerste reserveert 10

posities voor de uitkomst en het antwoord moet je links uitlijnen

Je hoeft je hier niet druk om te maken want het format statement is vrij heftig,

Dus als de getallen niet netjes onder elkaar staan is het geen probleem

Als je het getal 4 ingeeft zal het nog mooi onder elkaar staan

Het **print()** geeft een **enter**

FUNCTIES

We gaan nu zelf functies maken.

Er zijn veel voordelen voor het maken van functies maar de 3 belangrijkste zijn toch :

1. De overzichtelijkheid van je programma
2. Als je meerdere keren hetzelfde stukje code nodig hebt in een programma

dan kun je een functie hiervan maken en dan aanroepen zodat je programma

niet overdreven lang wordt

1. Als je probleem te moeilijk is om in 1 keer op te lossen doe je het in stukjes

Hoe maak je nu een functie in Python ?

Eigenlijk vrij eenvoudig nl met het statement **def ……. ()**

Voorbeeld : ***def totZiens():***

***print (“ Tot ziens !”)***

***print (“Hallo !”)***

***totZiens()***

Als je dit uitvoert zie je dat er eerst “Hallo” wordt geprint en daarna “Tot ziens”

De regel dat Python “Tot ziens” moet printen staat pas na print “Hallo” !!!!

In de eerste regel maken we de functie totZiens en in de laatste regel wordt deze pas uitgevoerd.

Python onthoudt nu tijdens het hele programma dat de functie totZiens() bestaat en je kunt deze nu net zo vaak gebruiken ( of aanroepen) als je wilt

Deze functie heeft nog geen parameters ( er staat niets tussen de haakjes () ) , maar dat kunnen we wel gaan doen

Voorbeeld : ***def college( naam ):***

***print( “Hallo , {} ! “ . format(naam) )***

***college (“Tinus”)***

***college (“Beppie”)***

***college (“Greetje”)***

***college (“Frits”)***

Er wordt een functie **college** gemaakt met de parameter **naam**. Dus om de functie college aan te roepen moet je er een **parameter** ( Tinus,Beppie,Greetje en Frits in ons geval) aan toe voegen

Natuurlijk kunnen er meerdere parameters aan een functie worden toegevoegd

Voorbeeld : ***def maal ( x , y ) :***

***antwoord = x \* y***

***print ( antwoord)***

***maal ( 6 , 3 )***

***maal ( 1234 , 5678 )***

Het RETURN commando :

Parameters worden vaak gebruikt om gegevens van buiten de functie naar de functie toe te brengen. ( zie de vorige voorbeelden )

Maar soms willen we ook informatie vanuit een functie naar buiten de functie brengen , en hiervoor hebben we het **return** commando nodig

Voorbeeld : **from** math **impor**t sqrt # je mag ook import math gebruiken

**def** pythagoras ( a, b ) :

**return** sqrt ( a \* a + b \* b )

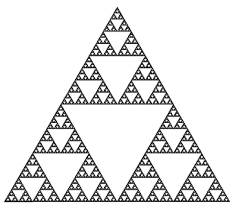
c= pythagoras ( 3 , 4 )

print ( c )

je ziet dat de functie pythagoras de wortel berekent van a \* a + b \* b

de uitkomst komt buiten de functie in de variabele c

Recursie (alleen voor Gymnasiasten en enthousiaste vwo-ers)

[](http://www.google.nl/url?sa=i&rct=j&q=&esrc=s&source=images&cd=&cad=rja&uact=8&ved=2ahUKEwjWubKWw9rcAhUP-aQKHRhHDMcQjRx6BAgBEAQ&url=http://www.fisme.science.uu.nl/wiskrant/artikelen/212/212december_boon.pdf&psig=AOvVaw1gPGexdzckDpxLmUcSEUFc&ust=1533716877566942)

Dit onderdeel is vrij heftig maar wel heel mooi om te doen.

Recursie is een techniek waarbij een functie zichzelf aanroept. Maw een functie a() roept een functie b() aan en deze functie b() roept functie a() weer aan

Het grootste gevaar bij recursie is dat je in een eindeloze loop terecht komt dus is het reuze uitkijken met programmeren

Een van de eenvoudigste recursie problemen is het berekenen van een faculteit ( n! )

5! = 5 \* 4 \* 3 \* 2 \* 1 = 120

In Python : ***def faculteit (n) :***

***if n <= 1 :***

***return 1***

***return n \* faculteit ( n – 1 )***

***print ( faculteit ( 5 ) )***

Laten we eens kijken wat er gebeurt : De tweede en derde regel zijn slechts een beveiliging

( je kunt geen faculteit van neg. getallen berekenen)

***aanroep faculteit (5 )***

***aanroep faculteit (4 )***

***aanroep faculteit (3 )***

***aanroep faculteit (2 )***

***aanroep faculteit (1 )***

***return 1***

***return 2 \* 1***

***return 3 \* 2***

***return 4 \* 6***

***return 5 \* 24***

***print ( 120)***

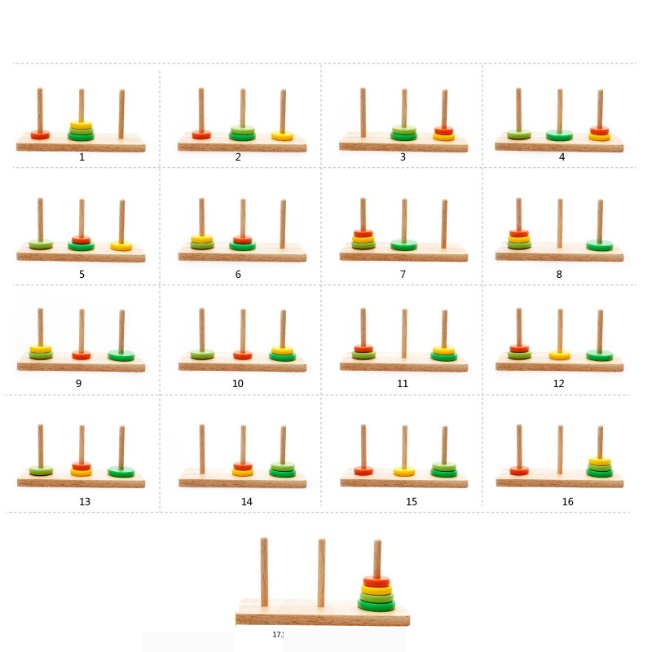
Nu nemen alle aanroepen een stukje geheugen in beslag , je krijgt al gauw een stack overflow mededeling , of het berekenen gaat bijzonder traag

Problemen die je met recursie op kunt lossen zijn bv : algoritme van Euclides om de

grootste gemene deler van 2

getallen te bepalen

De torens van Hanoi

[](https://www.google.nl/url?sa=i&rct=j&q=&esrc=s&source=images&cd=&cad=rja&uact=8&ved=2ahUKEwiA9vaEydrcAhWOJlAKHeRaC1wQjRx6BAgBEAU&url=https://nl.aliexpress.com/item/Tower-of-Hanoi-intellective-Game-Colorful-Rainbow-Rings-Kids-Toys-Educational-Toys-Children-Wooden-Puzzle-Toy/32643680593.html&psig=AOvVaw3P6F0ueYikhgyEEnnTdKmh&ust=1533718540578273)

Als je het leuk lijkt kun je proberen om deze problemen op te lossen

Hier zijn de oplossingen :

Euclides ***def gcd( m, n ):***

***if m % n == 0:***

***return n***

***return gcd( n, m%n )***

***print( gcd( 7\*5\*13, 2\*3\*7\*11 ) )***

Torens van Hanoi : ***GROOTTE = 4***

***def hanoi( p\_van, p\_tmp, p\_naar, grootte ):***

***if grootte == 1:***

***print( "Schijf 1 van", p\_van, "naar", p\_naar )***

***return 1***

***stappen = hanoi( p\_van, p\_naar, p\_tmp, grootte-1 )***

***print( "Schijf", grootte, "van", p\_van, "naar", p\_naar )***

***stappen += 1+hanoi( p\_tmp, p\_van, p\_naar, grootte-1 )***

***return stappen***

***stappen = hanoi( 'A', 'B', 'C', GROOTTE )***

***print( stappen, "stappen gedaan" )***

Tot zover iets over recursie , als je er meer van wilt weten is er genoeg te vinden op internet

STRINGS

Tot nu toe hebben we heel veel met getallen gewerkt maar we gaan nu kijken hoe we met teksten om moeten gaan

Je weet dat een string een stuk tekst is dat is omsloten door aanhalingstekens

De eerste twee bewerkingen met een string zijn + en \*

Neem het volgende voorbeeld over en kijk of je het snapt

***s1 = “appel”***

***s2 = “banaan”***

***print ( s1 )***

***print ( s2 )***

***print ( s1 + s2 )***

***print ( 3 \* s1 )***

***print ( s2 \* 3 )***

***print ( 2 \* s1 + 2 \* s2 )***

met de **len()** functie kun je de lengte van een string bepalen ( het aantal tekens )

dus print ( len (s1)) inhet vorige voorbeeld zal als antwoord 5 opleveren want appel bestaat uit 5 tekens

Met de for in loop kun je bepalen welke letters dat zijn

***for letter in s1 :***

***print ( letter )***

zet deze twee laatste regels maar eens na de laatste regel van ons voorbeeld

Als je een string hebt die uit meerdere regels bestaat kun je gebruik maken van de backslash ( \ )

Deze geeft aan dat de tekst verder gaat op de volgende regel (s )

Voorbeeld : ***langestring = “Het is de laatste weken ontzettend warm geweest hier in het \***

***zuid-oosten van Nederland. Maar je hebt altijd mensen die met \***

***deze temperaturen een cursus Python aan het schrijven zijn “***

***print ( langestring )***

Er zijn een aantal speciale tekens die je in een string kunt opnemen , deze worden altijd voorafgegaan door een backslash

\n betekent “newline” en zorgt ervoor dat de tekst op de volgende regel doorgaat

\’ wordt gebruikt om een “ in een tekst weer te geven

\\ gebruik je om een \ weer te geven in een tekst

Ieder teken in een string heeft een bepaalde positie ( een zgn index ) . deze indices beginnen bij 0

Dus het woord Python heeft 6 indices

Het commando print ( “Python”[4]) zal als resultaat de letter o opleveren

Opg 21 De gebruiker voert een string in en het programma geeft alle letters afzonderlijk

weer. Dus als je appel invoert komt als resultaat a

p

p

e

l

op het scherm.

Tip : gebruik **len ( … )** en in **range**

Nog een aardigheidje :

Kijk eens wat er gebeurt als je het volgende uitvoert :

***fruit = “aalbes”***

***print ( fruit [ :] )***

***print ( fruit [ 0: ] )***

***print ( fruit [ :6] )***

***print ( fruit [ : 100] )***

***print ( fruit [2] , fruit [ 1:5] )***

Er zijn in Python al een paar **methodes** beschikbaar die je zonder een import opdracht kunt gebruiken. Nu hebben we het nog niet over methodes gehad maar dat komt nog aan bod

**strip()** : Deze verwijderd de spaties aan het begin en einde van een string.

Als je iets anders wilt verwijderen dan geef je het een parameter mee

( hangt van de python versie af of dit werkt )

upper () Zet alles om in hoofdletters

lower() Zet alles om in kleine letters

replace() Vervangt letters

find () zoekt de indexwaarde op

***s = " Goede Morgen "***

***print (s)***

***print (s.strip())***

***print (s.upper())***

***print (s.lower())***

***print (s.replace("e", "a"))***

***print(s.find(“e”))***

Codering van tekens

Alle computersystemen gebruiken een manier om tekens te coderen

De basiscodering is de ASCII code.

Voor een ASCII tabel kun je googlen en dan zie je bv dat “A” het getal 65 krijgt

Omdat in Python voor elkaar te krijgen gebruik je de **ord()** functie

Dus ord(“A”) geeft 65

De tegenhanger is **chr()**

En geeft chr(65) de letter “A”

Hier kun je allerlei trucjes mee uithalen

Wat doet deze opdracht : ***print( “de twaalfde letter na g is “, chr( ord( “g”) + 12 ))***

We hebben ook nog de UNICODE ( deze kan veel meer “vreemde tekens” bevatten )

De meest gebruikelijke versie van de Unicode is UTF – 8

Als je een teken uit de Unicode wilt opnemen dan moet je het UTF-8 nummer van het teken weten.

Je kunt dan de code : \uxxxx opnemen waarbij xxxx staat voor het hexadecimaal getal

Voorbeeld : ***tekens = “\u0391”***

***for i in range (25) :***

***print ( chr( ord ( tekens ) + i ) , end= “ “ )***

Als antwoord krijg je : Α Β Γ Δ Ε Ζ Η Θ Ι Κ Λ Μ Ν Ξ Ο Π Ρ ΢ Σ Τ Υ Φ Χ Ψ Ω

Opg 22. Druk een lijst af met alle hoofdletters van A t/m Z en hun ASCII nummer

Opg 23 De gebruiker voert een string in. Als uitkomst komt er op het scherm hoeveel

maal de letter e voorkomt.

Test je programma met het woord : zeeeendeei

Deze is niet verplicht ( google even wat **reversed** doet)

Opg 24 De gebruiker voert een string in en op het scherm kom deze string achterste-

voren. Dus college wordt egelloc