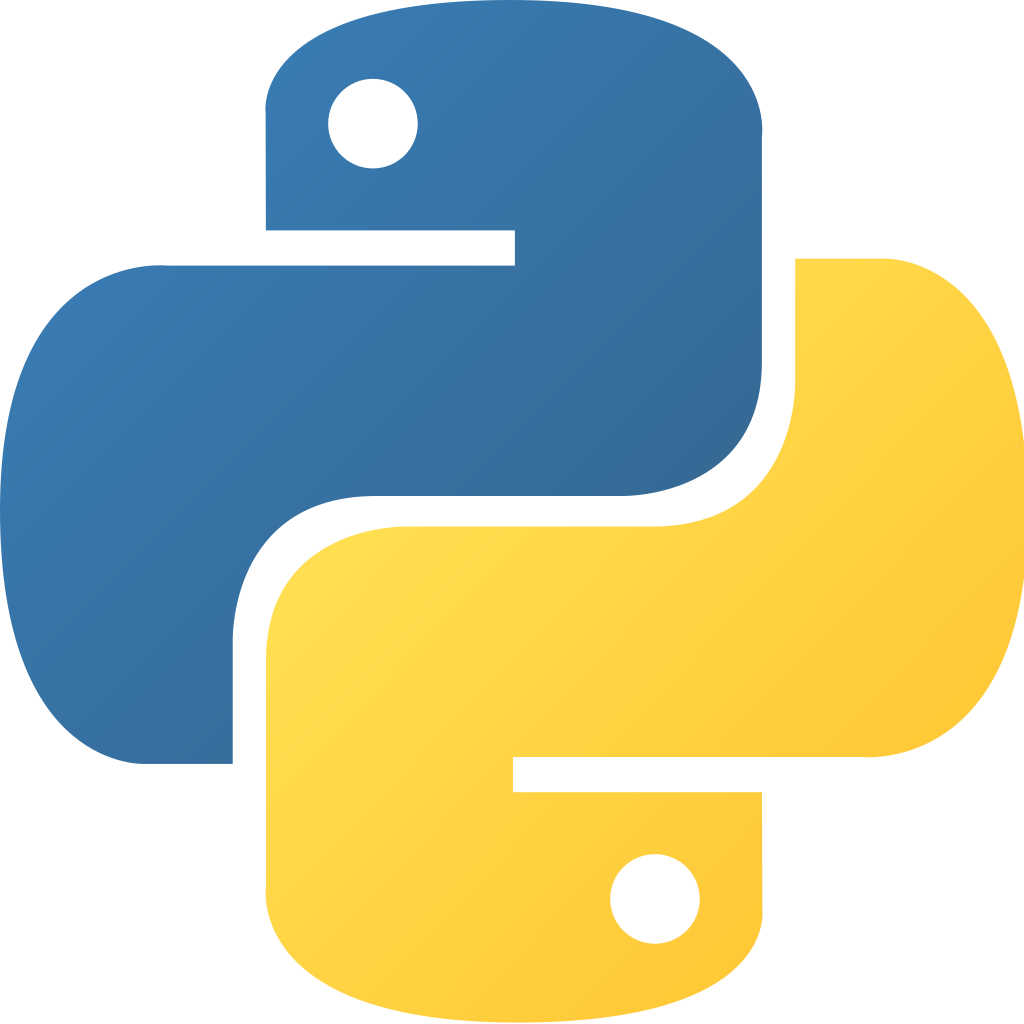
Programmeren in Python

Een introductie in de wereld van scripttalen

@language=python@

# Welkom!

Welkom bij de module “Programmeren in Python” in het informaticaprogramma van de Q-Highschool! Hierin leer je de basis van Python, een programmeertaal die je uiteindelijk gaat gebruiken om je zelfgekozen eindopdracht te maken!

## Waarom Python?

Maar, waarom eigenlijk Python? Python is één van de simpelste programmeertalen om mee te beginnen. Het brengt je in contact met de manier waarop code geïnterpreteerd wordt door je systeem en laat je kennismaken met de opbouw van een scripttaal, een taal die werkt op de achtergrond en veel gebruikt wordt voor automatisering van taken.

Dit is superhandig, want hoewel er zeker verschillen zitten tussen de formulering en de vaktermen tussen talen, is ook heel veel nagenoeg hetzelfde. Vooral de structuur verandert nauwelijks tussen programmeertalen.

Ook wordt Python zelf gebruikt in de vakwereld. Neem Instagram, één van de grootste online sociale media om foto’s te delen. Instagram was, qua functionaliteit, vrijwel volledig gebouwd in Python! Ook andere tech-giganten gebruiken deze taal, zoals Spotify, Netflix en zelfs Google!

En buiten deze grote bedrijven wordt Python ook veel voor projecten gebruikt. Volgens GitHub, de grootste database van codeprojecten wereldwijd, is Python de derde meest populaire taal die gebruikt wordt!

## Introductie

Waarvoor gebruiken jullie Python eigenlijk? Bespreek in een groepje welke programma’s jullie dagelijks gebruiken en ga op onderzoek uit of deze programma’s Python gebruiken.

Wat weten jullie al over programmeren? Bespreek in een groepje wat jullie weten over hoe een scripttaal eigenlijk qua structuur in elkaar zit.

# Werken met deze module

Als je verder scrollt, zie je misschien al wel dat er weinig tekst staat. Dit komt doordat het overgrote deel van de stof die we gaan behandelen, en enkele oefenopgaven die erbij horen, staan op een externe site genaamd *cscircles.cemc.uwaterloo.ca*.

Elke paragraaf word je gelinkt naar een bepaalde pagina op deze site, waarin je de stof kan lezen en indien nodig oefenopgaven kan maken. Dit is geen verplichting, maar wel handig. Tevens wordt waar mogelijk een andere externe bron gelinkt. Deze kan gebruikt worden voor extra oefening. Je vindt hier soms ook onbehandelde informatie in die je zou kunnen gebruiken.

Bij de meeste paragrafen staan op deze pagina ook nog aparte opdrachten. Dit zijn lastigere oefenopgaven, bedoeld voor jezelf om te testen of je de stof voldoende snapt en kan toepassen. Ook deze zijn niet verplicht, echter zeer aan te raden. Maak hierbij voor elke opdracht een nieuw document aan en sla deze op, om het later nog in te kunnen zien.

Sommige paragrafen staan *schuingedrukt* in de inhoudsopgave. Dit zijn inleveropdrachten, deze worden nagekeken. Nadere omschrijving en beoordelingscriteria staan in de paragrafen zelf.

Helemaal achterin (achter de eindopdracht) tref je een aantal extra onderwerpen. Hiervan hoef je de stof niet te kennen. Voor mensen die een uitdaging zoeken kan dit echter een mooie kans zijn. Ook kan toepassing van deze stof in de eindopdracht beloond worden met punten.

Als je behoefte hebt aan nog meer oefening, voel je vrij om op het web een opdracht op te zoeken die je leuk of nuttig lijkt. Dit geldt ook voor verdieping en verbreding van je kennis. Laat je dus niet inperken door wat in deze module staat, maar ga vooral ook zelf op onderzoek uit!

Bij de opdrachten mag je volledig te werk gaan zoals jij wil en zelf keuzes maken. De enige voorwaarde is dat de programma’s voldoen aan de eisen die in de desbetreffende opdracht staan.

Mocht een opdracht niet in één keer lukken, zit er dan even op! Meestal komt een strategie wel in je op als je het even laat inzinken.

Ook als je programma niet werkt zoals gewenst of helemaal niet werkt, geef niet op! Python heeft duidelijke error-messages en ingebouwde functies die debuggen eenvoudig zouden moeten maken. Bovendien is coderen een leerproces, en trial-en-error is een fundamenteel onderdeel hiervan.

# PyCharm

Voor deze module wordt aangeraden het programma PyCharm te gebruiken. Dit programma is namelijk makkelijk in gebruik en overzichtelijk, en heeft daarbij ook nog eens een ingebouwde interpreter (hetgeen je code leest en vertaalt naar computertaal).

Installeer eerst Python. Download de installer [hier](https://www.python.org/downloads/), en volg de instructies. Verander niets qua instelling, aangezien dit niet nodig is.

Download als dat klaar is PyCharm Community Edition [hier](https://www.jetbrains.com/pycharm/download/#section=windows). Bij de installatie, selecteer *Add launchers dir to the PATH*, *Create Associations - .py* en *Add “Open Folder as Project”*.  
Als alles gereed is kan je beginnen met het instellen. Open PyCharm, accepteer de *Terms and Conditions* en kies je favoriete theme. Plugins zijn niet nodig, sla deze over.  
Klik dan op *Create New Project*. Kies de gewilde locatie op je PC waar je het project wil neerzetten en de naam van het project. Klik dan op *Project Interpreter*. Selecteer *Existing Interpreter* en kies *Python 3.<getal>*. Als deze er niet tussen staat, klik op de drie puntjes, *System Interpreter* en selecteer de enige optie. Klik op *OK* en daarna *Create*.

Je hebt nu je eerste project! Om een nieuw bestand te maken klik je met de rechter-muisknop op het mapje met de naam die je het project gegeven hebt, onder *Project*. Selecteer *New*, *Python File*.  
Copy-paste @@print(“hi”)@@ in het bestand dat je net gemaakt hebt en klik bovenin op tab *Run, Run* (of klik Shift + F10). Je ziet onder in beeld een zwart scherm opkomen waar @@hi@@ in staat.

Nu weet je hoe je PyCharm gebruikt!

Verdere informatie kan je hier vinden: <https://www.jetbrains.com/help/pycharm/quick-start-guide.html>

Een andere editor gebruiken mag altijd. Hiervoor moet je echter wel zelf opzoeken hoe je de interpreter (als deze er is) instelt.

# @exercise@Introductie & Variables@-exercise@

## Leerdoelen

Je leert @@print()@@-statements maken en zo waarden te printen.  
Je leert variables opslaan, oproepen en waar nodig overschrijven.  
Ook leer je errors goed genoeg lezen om te begrijpen waar het probleem ligt en wat het probleem is, om zo de errors te gebruiken om je code te debuggen.

CSCircles stof en opdrachten: *0: Hello!* en *1: Variables*.  
W3Schools: *[Python Variables](https://www.w3schools.com/python/python_variables.asp)*  
Extra opdrachten: *[Variables Exercises](https://pythonbasics.org/variables/" \l "Exercises)*

## Opdrachten

1. Analyseer de volgende regel code:

@@print(Hello World)@@

* 1. Deze code veroorzaakt een error. Bedenk en noteer waarom.
  2. Voer de code uit. Je ziet in de console (het zwarte gedeelte onderaan je scherm) de error-message in het rood. Lees deze door en noteer voor elk onderdeel wat het betekent.
  3. Verander de code zo, dat de output @@Hello World@@ is.

1. Je maakt een script om je naam om te keren.

Om iets om te keren, moet je er @@[::-1]@@achter zetten.

Je hoeft nu nog niet te begrijpen hoe dit werkt.

1. Maak een variable aan met als naam @@name@@en sla hierin je eigen naam op.
2. Maak een tweede variable aan genaamd @@reversedName@@ en zorg dat deze je variable uit *a* omdraait.
3. Print vervolgens je omgekeerde naam.
4. Je maakt een script om van een waarde de volgende 3 exponenten (machten) uit te rekenen.  
   1. Maak een variable waarin je de oorspronkelijke waarde zet. Noem deze @@ogValue*@@* (kort voor “originalValue”).

Hierna ga je de berekeningen uitvoeren. Hiervoor heb je **alleen** vermenigvuldigen nodig; machtsverheffen wordt in een later hoofdstuk pas behandeld.

Uiteindelijk wil je natuurlijk een output die de resultaten weergeeft. Er zijn twee manieren die tot dit resultaat leiden.  
Bij de één gebruik je een variable waarin je alle berekeningen opslaat. Hier gebruik je de plus-operator (+) om strings “bij elkaar op te tellen”.  
Bij de ander sla je elke berekening op in een aparte variable, en gebruik je de komma in @@print(x, y)@@om automatisch spaties te laten verschijnen tussen alle values.

* 1. Codeer beide manieren. Zorg ervoor dat voor beide manieren de output het format hebben van @@<origineleWaarde> <exponent1> <exponent2> <exponent3>@@.  
       
     NOOT: Let erop dat Python **strings** (karakters) en **integers** (gehele getallen) niet bij elkaar kan optellen! Gebruik @@str(<geheel getal>)@@ om het getal te transformeren naar een string.

# @exercise@Functions@-exercise@

## Leerdoelen

Je leert ingebouwde functions te gebruiken om complexe taken niet zelf te hoeven coderen.  
Je snapt hoe arguments werken en waarom deze bestaan.

CSCircles stof en opdrachten: *2: Functions* en *4: Types*  
Lijst Built-In functions: *[Built-in Functions](https://www.w3schools.com/python/python_ref_functions.asp)*  
W3Schools types: *[Python Numbers](https://www.w3schools.com/python/python_numbers.asp)* (geen complex), [*Python Casting*](https://www.w3schools.com/python/python_casting.asp)

## Opdrachten

1. Bij de laatste opdracht van het vorige hoofdstuk heb je al gezien dat types invloed kunnen hebben op hoe je code zichzelf uitvoert. Ook heb je toen kennis gemaakt met een functie genaamd @@str()@@, die je gebruikt hebt om een integer om te zetten naar een string.  
     
   Om verder te gaan op deze types ga je de functie @@type()@@ gebruiken. @@type()@@neemt een **value** (algemeen woord voor een waarde, ongeacht het type) en geeft het type ervan weer. Dit is onmisbaar bij het debuggen van code, omdat redelijk wat fouten bijvoorbeeld komen van het aanroepen van een functie met een verkeerd type value.  
   1. Gebruik de @@type()@@-functie voor de waarden: @@“hi”@@, @@9@@, @@34.2313@@ en @@9/13@@. Test als je wil met meer waarden. Noteer welke Nederlandse begrippen corresponderen met de Python-types.  
        
      NOOT: Zorg wel dat je alle function calls print, anders krijg je niets te zien!

Je hebt al kennis gemaakt met @@str()@@. Ook @@int()@@ en @@float()@@ bestaan. Deze zijn er allemaal om waarden van het ene naar het andere type te transformeren.

* 1. Test uit welke outputs deze functies geven bij verschillende types inputs.

# @exercise@Comments & Quotes@-exercise@

## Leerdoelen

Je leert code te commenten, tot op de hoogte dat iemand zonder enige Python-experience je script zou begrijpen door enkel de comments te lezen.  
Je leert hoe escape characters werken en wanneer deze gebruikt worden.

CSCircles stof en opdrachten: *3: Comments, Quotes*  
W3Schools: [*Python Comments*](https://www.w3schools.com/python/python_comments.asp)

Voor dit hoofdstuk zijn geen extra opdrachten.

NOOT: bij opdrachten voor een cijfer word je ook op je comments beoordeeld

# @exercise@input()@-exercise@

## Leerdoelen

Je leert @@input()@@ te gebruiken om degene die je script gebruikt te vragen om een waarde in te geven.  
Je leert hoe je een ingevoerde waarde moet verwerken in je script.

CSCircles stof en opdrachten: *5: Input*  
W3Schools: [*input() function*](https://www.w3schools.com/python/ref_func_input.asp)

## Opdrachten

1. We verbeteren opdracht 3 van *Introductie & Variables* (de exponenten-opdracht). Toen heb je zelf de originele waarde die je zou gaan machtsverheffen opgeslagen in variable @@ogValue@@.  
     
   Verander die variable zo, dat deze als originele value een input neemt.  
   Vanwege kleine onzuiverheden in het afronden van floats, hoeft dit enkel te werken op ints. Geef waar nodig dus aanwijzingen, om te voorkomen dat het script een error geeft.
2. Je maakt een script om ship names te genereren.  
     
   Een ship name is een combinatie van de twee namen van twee personen die aan elkaar gekoppeld worden.  
     
   Hoewel een ship name meestal een vloeiende naam is, heb je op dit moment nog niet de kennis om dit te maken. Wel ga je twee namen samenvoegen.  
   1. Maak twee variabelen aan die allebei een input aannemen. Deze twee inputs zijn de namen die we samen gaan voegen.

Om te kijken hoeveel letters de een en hoeveel letters de andere naam af moet staan ga je @@randint@@uit module @@random@@ gebruiken. Dit zit echter niet standaard in Python, en moeten we importen. Dit gaat als volgt:

@@import random@@

En deze function gebruik je zo: @@random.randint(begin, end)@@  
Waar *begin* het eerste gehele getal is waar de function uit kan kiezen en *end* het laatste (en alles ertussenin).

* 1. Gebruik de function om twee willekeurige gehele getallen tussen 1 en de lengte van de naam (1 en lengte *niet*meegerekend) te kiezen, en sla deze getallen op in een variable.  
     Let hierbij dat je de lengte van een string kan bepalen door middel van @@len(arg)@, waar *arg* een string (of een variable die een string bevat) is.

We gaan nu de de hoeveelheid letters die je net bepaald hebt uit de namen halen. Voor het gemak zeggen we dat de eerst ingevulde naam het voorste deel afstaat, en de laatst ingevulde naam het achterste deel.  
Hiervoor gebruiken we indices (enkelvoud: index). De index van een karakter in een string zegt gewoon op welke plek deze in de string staat. Door hier gebruik van te maken kunnen we niet alleen één letter in een string selecteren, maar ook een aantal letters tegelijk.

Aangezien je van de eerste naam het eerste deel van de willekeurig bepaalde lengte wil, gebruik je het volgende: @@nameOne[0:randOne]@@.  
Dit betekent: van @@nameOne@@, pak alle letters van index @@0@@ (de allereerste letter) t/m de letter met index @@randOne@@ (de willekeurig gegenereerde index).

* 1. Vul dit in met je eigen variabelen en sla het op in een nieuwe variabele.

Zo’n zelfde idee geldt ook voor de tweede naam, alleen dan het laatste deel.

Hiervoor gebruiken we: @@nameTwo[len(nameTwo) – randTwo : len(nameTwo)]@@.  
Dit betekent: van @@nameTwo@@, pak alle letters van index @@len(nameTwo)-randTwo@@(dus, @@randTwo@@ letters terug vanaf het einde van de string) t/m index @@len(nameTwo)@@(het einde van de string).

* 1. Doe hiervoor hetzelfde als bij c.
  2. Print het eerste deel en het tweede deel uit, zonder spatie of newline ertussen.

# @exercise@If/else statements & boolean operators@-exercise@

## Leerdoelen

Je kan nu if/else-statements en boolean operators gebruiken om conditionals te maken in een als/dan/anders-constructie.

CSCircles stof en opdrachten: *6: If* en *9: Else, And, Or, Not*  
W3Schools: [*Python Conditions*](https://www.w3schools.com/python/python_conditions.asp)  
Extra opdrachten: [*If-statements Exercises*](https://pythonbasics.org/if-statements/#Exercise)

## Opdrachten

1. Je maakt een script dat twee strings aanneemt als input en de langste van de twee print.   
   Als beide strings even lang zijn, print je script beide strings uit. Gebruik hiervoor de ingebouwde functie @@len(“string”)@@.
2. Voor 3 cijfer-inputs (X, Y en Z), maak variabelen x, y en x zo, x = kleinste van XYZ, y = middelste, z = grootste (oftewel x <= y <= z). Gebruik hiervoor alleen if/else statements en simpele variabelen.
3. Neem een functie genaamd @@isvalid(string)@@ over (later meer over functies). De functie wordt hieronder weergegeven:

@@@  
def isvalid(string):  
 if 0 <= int(string) <= 10:  
 print("This number lies between 0 and 10!")  
 elif string.isdigit():  
 print("This number doesn't lie between 0 and 10!")  
 else: print("This is either a float or not a number!")  
@@@

Deze code werkt niet zoals gewenst.

1. Bedenk wat hieraan fout zou kunnen zijn en check vervolgens door voorbeeld-inputs.
2. Debug deze code op twee manieren: zonder het aanmaken van een nieuwe regel, en door nested (in elkaar gezette) if-else statements te gebruiken.

# @exercise@ABC-formule@-exercise@

De ABC-formule:

## Opdracht

* Maak een programma dat de ABC-formule kan toepassen op gehele getallen.
* Zorg dat je programma via de discriminant checkt of er 0, 1 of 2 reële uitkomsten zijn.
* Bij 0 uitkomsten, print een melding dat er geen reële resultaten zijn.  
  Bij 1, print dat er één versie is en reken één of beide versies uit. Ga verder met het volgende deel.  
  Bij 2, reken beide versies uit. Ga verder met het volgende deel.
* Gebruik de volgende line om te bepalen of een floatwaarde een int is. Pas je print statement zo aan, dat als het volgende geldt, je script “exact <integer-waarde>” print in plaats van een float. Doe deze check bij beide variabelen in het geval van twee reële uitkomsten.

@@if jouwResultaat.is\_integer():@@

* Als het nummer geen int is, gebruik @@Fraction()@@ uit module @@fractions@@ om te checken of de lengte van de resulterende breuk kleiner of gelijk aan 10 is. Als dit het geval is, print de breuk in plaats van het kommagetal. Haal wel eerst de helen ervan af! Gebruik hiervoor @@int()@@. Deze functie rondt positieve getallen af naar beneden en negatieve naar boven.
* Als het nummer geen van beiden is, print zijn float-waarde.

TIP: voor betere leesbaarheid, zoek de functie @@.format()@@ op op internet.

Beoordeling  
Je wordt beoordeeld op het volgende:

* Voldoe je volledig aan de opdracht *45 pt.*
* Is je programma gebruiksvriendelijk (goede prompts, formeel taalgebruik, volzinnen, etc) *20 pt.*
* Is je code leesbaar en logisch geordend; zou iemand met evenveel Python-kennis als jij dit programma kunnen begrijpen zonder comments te lezen *20 pt.*
* Heb je voldoende en nuttig gecomment; kan iemand die geen Python kan jouw programma begrijpen door enkel de comments te lezen *15 pt.*

# @exercise@Loops, indices en beter rekenen@-exercise@

## Leerdoelen

Je leert for loops te gebruiken voor vastgestelde ranges of voor alle karakters in een string.  
Je leert while loops te gebruiken voor een conditional loop (een loop die alleen maar werkt als de condition klopt).  
Je leert wanneer je for en wanneer while moet gebruiken.  
Je leert indices te gebruiken om bepaalde stukken tekst uit een string op te vragen.  
Je leert nieuwe operators gebruiken om beter, efficienter en leesbaarder te rekenen, zowel met numbers als met strings.

CSCircles stof en opdrachten: *7: Rich*, alle subcategorieën  
W3Schools: [*Python While Loops*](https://www.w3schools.com/python/python_while_loops.asp), [*Python For Loops*](https://www.w3schools.com/python/python_for_loops.asp), [*Python Strings*](https://www.w3schools.com/python/python_strings.asp), [*Python Operators*](https://www.w3schools.com/python/python_operators.asp)  
Extra opdrachten: [*Exercises For Loops*](https://pythonbasics.org/for-loops/#Exercise), [*Exercises While Loops*](https://pythonbasics.org/while-loop/#Exercise), [*Exercises Strings and Indices*](https://pythonbasics.org/strings/#Exercises)

## Opdrachten

1. Je gaat een eenvoudig script maken om karakters binnen een woord te vervangen, om er zo een raadwoord van te maken.

Zorg dat je script als input een woord neemt en uit dit woord willekeurige letters vervangt in *\_*(underscore). Bedenk zelf een eerlijke kans voor elke letter om vervangen te worden.

1. Je gaat een Caesar Cipher script maken. Een Caesar Cipher slaat op het @@<key>@@ aantal letters verplaatsen in het alfabet voor elke letter. De letter A met key 3 zou dus een D (*A* > B > C > *D*) worden, en D met key 23 wordt weer een A.

Jij gaat dit ook doen, maar in plaats van met alleen het alfabet, met de hele ASCII-tabel.

1. Zorg dat je script elk karakter uit een gebruiker-gegeven string een gebruiker-gegeven aantal plaatsen kan opschuiven.

Wat nu echter het geval is, is dat er maar 128 ASCII-karakters (maximaal aantal verschillende values om te passen in een binary van 7 bits) zijn. En je programma keert niet terug naar het begin van de lijst als je 127 passeert (0000000 🡪 0 in dec is ook een value).

Er zijn twee manieren om dit op te lossen.

1. Maak één script die enkel werkt als de key die de user geeft tussen -128 en 128 ligt. Zorg wel dat deze werkt als de waarde van een karakter + de key meer dan 128 is!

Maak ook een script waarin je het aantal “rondjes” dat je script maakt checkt en deze van de uiteindelijke value aftrekt, om zo voor elk omgezet karakter alsnog een waarde van 0 t/m 127 is.

# @exercise@Remix@-exercise@

Maak *8: Remix* op CSCircles.

Dit is een verzameling lastigere sommen die dienen ter herhaling van de stof die je tot nu toe hebt gehad, en deze combineert.

Hierbij zijn geen extra opdrachten.

# @exercise@def (zelf functies maken)@-exercise@

## Leerdoelen

Je leert functies te gebruiken om vaker uit te voeren stukken code slechts één keer uit te hoeven typen. Je leert functies te gebruiken om je code overzichtelijker en leesbaarder te maken.

CSCircles stof en opdrachten: *10: def* en *11B: Variable Scopes*  
W3Schools: [*Python Functions*](https://www.w3schools.com/python/python_functions.asp)  
Extra opdrachten: *[Function Exercises](https://pythonbasics.org/functions/" \l "Exercise)*

## Opdrachten

1. Je gaat een eenvoudige function maken die een string als input neemt en deze string vervolgens verandert in een reeks getallen, de ASCII-values van de letters in de string. Scheid deze getallen van elkaar met spaties.  
     
   Dit kan je op twee manieren doen.
   1. Maak gebruik van *global* om vanuit de function de variable op module-level te editen.
   2. Maak gebruik van een return statement om de variable over te schrijven.
2. Maak een programma dat een willekeurige integer tussen 0 en @@int(input)@@genereert, en die mensen laat raden en checkt of ze het correcte antwoord hebben (een lotingsprogramma).  
     
   Maak 2 functions:  
   Eén om te checken of het geraden cijfer tussen 0 en @@input()@@ ligt (is het een valide poging);  
   Eén om te checken of het geraden cijfer correct is.  
     
   Check ook of alle ingevoerde waardes wel integers zijn; @@int()@@ gebruiken op niet-int strings veroorzaakt namelijk een error. Gebruik hiervoor @@.isnumeric()@@.

# @exercise@Tips@-exercise@

Maak *12: Tips* op CSCircles. Deze paragraaf bevat vooral handigheidjes die je kan gebruiken om je code handiger/effectiever te maken.

Hierbij zijn geen extra opdrachten.

# @exercise@Lists & Dictionaries@-exercise@

## Leerdoelen

Je leert dictionaries en lists te gebruiken om grote hoeveelheden data in op te slaan.  
Je leert hoe de key-value verhouding werkt bij dictionaries en kan hiermee programmeren.  
Je leert hoe indices werken bij lists.

CSCircles stof en opdrachten: *13: Lists* en *15A: Termination Determination*  
Dictionaries stof en opdrachten: *[LearnPython Dictionaries](https://www.learnpython.org/en/Dictionaries)*  
W3Schools: [*Python Lists*](https://www.w3schools.com/python/python_lists.asp), [*Python Dictionaries*](https://www.w3schools.com/python/python_dictionaries.asp)  
Extra opdrachten: *[Dictionary Exercises](https://pythonbasics.org/dictionary/" \l "Exercise)*, [*Split Exercises*](https://pythonbasics.org/split/#Exercises), [*List Exercises*](https://pythonbasics.org/list-operations/#Exercise)

## Opdrachten

1. Schrijf een telefoonboek-programma. Je programma moet bij invoeren van een naam het geregistreerde telefoonnummer teruggeven. Ook moet je een modus hebben voor het toevoegen en verwijderen van telefoonnummers, waar je in blijft totdat de gebruiker @@exit@@ ingeeft.

# @exercise@Galgje met een twist@-exercise@

## Opdracht

Maak een woordraadspel dat werkt als volgt:

* Je script vraagt een input en verandert in die string 40% van de letters in \_(underscore).
* Gebruikers kunnen een index gebruiken om op de positie van een underscore een letter te raden. Deze indexgetallen dienen telkens onder het aangepaste woord geprint te worden.
* Als ze deze letter goed geraden hebben, wordt de underscore vervangen door de letter.
* Als ze de letter fout geraden hebben, geeft je script een hint of de letter een klinker of een medeklinker is.
* Gebruikers hebben een aantal levens, scalair aan het aantal lege vakken in het woord (oftewel: kan 1 x aantal, 2 x aantal, 0.5 x aantal, 1.2 x aantal afgerond, etc. zijn), en als deze op zijn is het game-over.
* Als gebruikers denken het woord te weten, kunnen ze het volledig raden. Hiervoor hebben ze één kans; daarna is het ook game-over.

*Let op: Je dient overal checks in te zetten of ingevoerde waarden kloppen! De code mag niet breekbaar zijn!*

## Beoordeling

Je wordt beoordeeld op het volgende:

* Voldoe je volledig aan de opdracht en valt je programma niet te breken? *50 pt.*
* Zijn de prompts goed, gebruiksvriendelijk en overzichtelijk in de console? *20 pt.*
* Is de code voor een programmeur met jouw kennis goed leesbaar, met waar nodig comments voor verduidelijking? *20 pt.*
* Is de code in het algemeen efficient tot zover jouw kennis rijkt; heb je overal de makkelijke en efficiente methodes voor gekozen? *10 pt.*

Verder wordt naar inzicht van de docent voor nuttige/leuke extra’s een bonus van maximaal *10 pt.* uitgereikt.

Je cijfer wordt bepaald door het uiteindelijke aantal punten te delen door 10.

*Deze opdracht telt voor 30% mee voor je eindcijfer Python.*

# @exercise@Eindopdracht@-exercise@

## Opdracht

Nu mag je zelf iets bedenken en gaan maken! Dit kan van alles zijn. Bijvoorbeeld:

* Boter, kaas en eieren;
* Een spel gebaseerd op tekst;
* Een password-generator;
* Iets anders dat je zelf leuk lijkt!

Je mag deze opdracht in je eentje of met z’n tweeën maken. Zodra je een idee hebt, werk je deze uit. Kom dan naar de docent toe en vraag om goedkeuring. Een aantal belangrijke punten waar de docent op gaat letten zijn:

* Is de opdracht qua niveau goed? (niet te hoog gegrepen maar vooral niet te makkelijk)
* Duurt de opdracht niet te lang/te kort om te maken? (gestelde tijd is ongeveer 4 uur per leerling)
* Kan je de vaardigheden die je geleerd hebt goed in de opdracht kwijt? (oftewel: je kan wel een lastig project maken met alleen if/else statements, maar kan je erin ook loops verwerken)

Zorg dus wel dat je je idee voldoende diep hebt uitgewerkt, dat jij en de docent samen een goede tijds- en niveauinschatting kunnen maken!

Als je groen licht hebt gekregen van de docent mag je beginnen!

TIP: als je niet op ideeën kan komen, kijk dan vooral op internet! Er zijn genoeg artikelen die Python-projecten van eenvoudig tot lastig behandelen!

## Beoordeling

Je wordt beoordeeld op het volgende:

* Werkt je programma zoals verwacht en valt de code niet te breken? *50 pt.*
* Zijn de prompts goed, gebruiksvriendelijk en overzichtelijk in de console? *25 pt.*
* Is de code voor een programmeur met jouw/jullie kennis goed leesbaar, met waar nodig comments voor verduidelijking? *15 pt.*
* Is de code in het algemeen efficient tot zover jouw kennis rijkt; heb je overal de makkelijke en efficiente methodes voor gekozen? *10 pt.*

Verder wordt naar inzicht van de docent aan de hand van moeilijkheidsgraad de score hierboven vermenigvuldigd met een waarde tussen *0.8x* en *1.2x*, waarbij zeer moeilijke projecten hun punten x 1.2, zeer makkelijke projecten hun punten x 0.8 krijgen en middelmatige projecten hun punten x 1.0 krijgen.

Als laatste wordt naar inzicht van de docent voor creativiteit een bonus van maximaal *10 pt.*uitgereikt. Deze punten zijn ongewogen (niet beïnvloed door de vermenigvuldiging).

Je cijfer wordt bepaald door het uiteindelijke aantal punten te delen door 10.

*Deze opdracht telt voor 50% mee voor je eindcijfer Python.*

# @exercise@Extra’s@-exercise@

Hierover moet je zelf onderzoek doen en oefenopgaven maken.

## Recursie

1. Een palindroom is een woord dat gelijk is als het omgekeerd gelezen wordt. “Lepel” of “meetsysteem” zijn hier voorbeelden van.  
     
   Schrijf een script dat een input aanneemt en hiervan checkt of het een palindroom is. Laat het niet hoofdlettergevoelig zijn (zowel @@lepel@@ als @@LEPEL@@ als @@LEpeL@@ moeten werken) en werken op alle tekens (@@[sdf0%%0fds]@@ is dus ook een palindroom). Gebruik een recursive function.  
     
   Als het een palindroom is, print “Dit is een palindroom!”. Anders, print “Dit is geen palindroom!”.

## Errors & Exceptions

## File Handling

1. Schrijf een script om een cijferlijst-bestand te maken.   
   Zorg dat deze als input een vak en een cijfer kan aannemen en kan schrijven naar het bestand. Laat het per vak cijfers kunnen ophalen, en de hele cijferlijst (uiteraard waarbij het duidelijk is voor welk vak welke cijfers zijn). Zorg ook dat up-to-date gemiddelden per vak in het bestand staan.

# @exercise@Breinbrekers@-exercise@

Lastige opgaven die alle stof, inclusief de extra’s, behandelt. *Maak deze dus pas als je de stof voldoende snapt.* Voor meer breinbrekers, ga vooral naar [Project Euler](https://projecteuler.net). Dit staat vol met uitdagende opgaven.

1. In een bepaalde regio zijn alle plaatsen door een rechtstreekse weg met elkaar verbonden. Maar voor alle wegen geldt éénrichtingsverkeer; als je van A naar B mag via de directe weg, mag je dus niet langs die weg van B naar A. Als je dat wel wilt zul je een omweg moeten nemen.  
     
   In zo’n gebied ga je op zoek naar een route waarbij je iedere plaats precies één keer bezoekt.   
     
   Schrijf een programma dat van [dit bestand](https://light-theme-hurts.my-ey.es/7yrsk1m.txt) met 100 gegevensreeksen eerst willekeurig één reeks kiest. De reeksen zijn gescheiden door blank lines.  
     
   Als de reeks bepaald is, laat je script regels hiervan inlezen.  
   Het aantal in te lezen regels wordt bepaald door de eerste regel van elke reeks, een getal N (2 < N < 27). Hiermee wordt het aantal plaatsen in het gebied aangegeven. De plaatsen hebben als naam een hoofdletter; de eerste N hoofdletters worden gebruikt als plaatsnaam.   
     
   Vervolgens leest je programma van de reeks in kwestie in N regels informatie over de richting van de verbindingswegen.   
   Op de eerste regel staan de N verbindingen van A naar alle andere plaatsen; een 0 betekent dat de weg in A eindigt, een 1 dat de weg in A begint. De volgende regels gaan over de volgende plaatsen.   
     
   Tussen twee plaatsen is altijd precies één weg; er zijn geen wegen van een plaats naar zichzelf!   
     
   Je programma schrijft naar standaard output één regel met daarop een route waarin alle plaatsen precies één keer worden bezocht. Er is altijd een oplossing mogelijk.  
     
   LET OP: Er wordt niet gevraagd om weer terug te gaan naar de plaats waar je begonnen bent.  
   TIP: Kijk het .txt-bestand met reeksen door voor beter begrip van het format.  
   Naar: Nederlandse Informatica Olympiade 2019, Eerste Ronde
2. Schrijf een functie dat de reeksen zoals hierboven beschreven genereerd.  
   Laat deze ook schrijven naar een bestand.
3. Schrijf een nieuw telefoonboek-script. Je programma moet bij invoeren van een naam het geregistreerde telefoonnummer teruggeven, en vice versa.  
   Ook moet je een modus hebben voor het toevoegen, verwijderen en bewerken (bestaand persoon, nieuw nummer) van telefoonnummers, waar je in blijft totdat je *exit* zegt.  
   Gebruik voor alle checks (bestaat deze persoon in het telefoonboek etc.) een try/except statement (dit vergemakkelijkt namelijk de code).  
   Zorg dat je script terugslaat op een telefoonboek-bestand.