



HBO-ICT: TICT-V1PROG-15

Programming

Werkboek

Versiebeheer

3.3	31-8-17	Tutorial Les 12 aangepast – voorbeeld TopLevel widget verwijderd (stap 6) Tutorial Les 12 aangepast – voorbeeld meerdere frames toegevoegd (stap 6). Installatiehandleiding aangepast naar PyCharm ipv PyCharm Edu ivm ondersteuning van SSH (naar de Raspberry Pi) in de eerstgenoemde.
3.4	2-9-17	Final Assignment les 6 herzien.

Inhoudsopgave

Versiebeheer	2
Inleiding.....	5
Installatie software.....	6
Les 1: Starten met Python (Perkovic – §1.1 t/m §2.3)	8
Vorbereiding.....	8
Perkovic Exercises	8
Practice Exercises	8
Les 2: Basisconcepten (Perkovic – §2.3 t/m 3.1).....	10
Vorbereiding.....	10
Perkovic Exercises	10
Practice Exercises	11
Final Assignment: Git & GitHub.....	12
Les 3: Control Structures (Perkovic – §3.2)	15
Vorbereiding.....	15
Perkovic Exercises	15
Practice Exercises	15
Les 4: Functions (Perkovic – §3.3 t/m 3.5)	17
Vorbereiding.....	17
Perkovic Exercises (/ Problems)	17
Practice Exercises	17
Final Assignment: NS-functies	18
Les 5: Strings, Text Data, and File I/O (Perkovic – §4.1 t/m 4.3).....	19
Vorbereiding.....	19
Perkovic Exercises (/ Problems)	19
Practice Exercises	19
Les 6: Execution Control Structures (Perkovic – §5.1 t/m 5.3).....	22
Vorbereiding.....	22
Perkovic Exercises (/ Problems)	22
Practice Exercises	22
Final Assignment: Bagagekluizen	23
Les 7: Control Structures & Dictionaries (Perkovic – §5.4 t/m 6.1).....	25
Vorbereiding.....	25

Perkovic Exercises (/ Problems)	25
Practice Exercises	25
Les 8: Containers (Sets and Chars) (Perkovic – §6.2 t/m 6.4)	27
Vorbereiding.....	27
Perkovic Exercises (/ Problems)	27
Practice Exercises	27
Final Assignment: NS-kaartautomaat.....	29
Les 9: Catching Exceptions, CSV-files (Perkovic – §4.4 & §7.3)	31
Vorbereiding.....	31
Perkovic Exercises (/ Problems)	31
Practice Exercises	31
Les 10: Namespaces / XML (Perkovic – §7.1 t/m 7.2)	33
Vorbereiding.....	33
Perkovic Exercises (/ Problems)	33
Practice Exercises	34
Final Assignment: XML-stationslijsten	38
Les 11: Vorbereiding Miniproject: NS-API.....	40
Vorbereiding.....	40
Tutorial: De NS-API.....	41
Les 12: Vorbereiding Miniproject: Tkinter	44
Vorbereiding.....	44
Tutorial: Tkinter	44

Inleiding

Om correcte programma's te kunnen schrijven moet je de theorie kennen. Maar programmeren is voornamelijk heel veel “doen”: programma's van anderen onderzoeken en zelf programma's schrijven. In deze **reader** staat hoe we het boek van **Perkovic** gaan gebruiken en welke opdrachten we gaan maken. Meer achtergrondinformatie over de cursus, de toetsing, leerdoelen en nog veel meer kun je vinden in de **studentenhandleiding**, neem deze ook goed door.

Literatuur: **Introduction to Computing Using Python, an application development focus**
(verplicht) **Ljubomir Perkovic, Wiley, ISBN: 978-0-470-61846-2, Second Edition¹**

Literatuur: De programmeursleerling, Leren coderen met Python 3, Pieter Spronck, **gratis te**
(optioneel) **downloaden** via <http://www.spronck.net/pythonbook/pythonboek.pdf>

Elke les heeft ongeveer dezelfde opbouw:

- **Voorbereiding op de les:** De docent gaat er van uit dat je de theorie van de les hebt bestudeerd en eventuele opdrachten hebt gemaakt.
- **De les:** Tijdens de les wordt de nieuwe theorie voornamelijk aan de hand van oefenopgaven besproken. De docent inventariseert welke vragen er leven naar aanleiding van de voorbereiding en gaat daarop in. Je maakt opdrachten om die lesstof te oefenen. Met alleen tijdens de les programmeren krijg je niet voldoende oefening, dus thuis afmaken!
- **Opdrachten tijdens de les en erna:** Door het maken van opdrachten verwerk je de lesstof van die les, en ga je steeds beter programmeren. Er worden iedere les meerdere opdrachten opgegeven.

Waar staat de informatie van deze cursus?

- Het **Python**-boek van **Perkovic** is de basis voor de theorie. In dit boek kun je de **Perkovic Exercises** vinden, en op deze theorie zijn de **Practice Exercises** en **Final Assignments** gebaseerd. Alle Practice Exercises en Final Assignments staan in dit werkboek.
- De **studentenhandleiding** bevat alle algemene informatie over deze cursus.
- Dit **werkboek** kun je echter gebruiken als **praktische leidraad** om het boek van Perkovic te doorlopen en je voor te bereiden op het tentamen.

Wat moet ik maken?

- De Perkovic Exercises waar dit werkboek regelmatig naar verwijst, zijn **niet** verplicht. Alle overige opdrachten in dit werkboek (Practice Exercises & Final Assignments) zijn **wel** verplicht.
- Zie voor de deadlines en wijze van opleveren de studentenhandleiding, §2.1.

¹ De Second Edition behandelt dezelfde theorie, als de First Edition, maar de bladzijdenummering is anders!

Installatie software

We gaan er vanuit dat je een laptop hebt waarop je de programmeeromgeving kunt gaan installeren. Deze omgeving bestaat uit:

- De programmeertaal “Python” (deze moet je **eerst** installeren) en
- De editor die het je makkelijker maakt om een Python programma te maken.

Tijdens les 1 moet je hiermee opdrachten maken en daarom gaan we die omgeving eerst installeren!

Python Installeren

Ga naar www.python.org. Ga naar Downloads en installeer de laatste versie van Python. Op het moment van schrijven is dit versie 3.6.2. De reader is op deze versie gebaseerd! **Let op:** de versie kan in de loop van het jaar veranderen.

IDE installeren

Om in Python programma's te maken, moet je Python-bestanden schrijven die eindigen op .py. De bestanden kun je vervolgens uitvoeren (met een interpreter), zodat jouw voor-mensen-leesbare-code wordt omgezet naar voor-computers-leesbare-code.

Om dat allemaal wat makkelijker te maken, bestaan er Integrated Development Environments (IDE), programma's die je helpen met de code te schrijven in een .py-bestand, maar ze helpen je ook om je code te controleren op fouten. Wij gebruiken de IDE **PyCharm**. Deze is niet gratis, maar via een Student-account kan je deze tijdens je studie wel zonder kosten gebruiken.

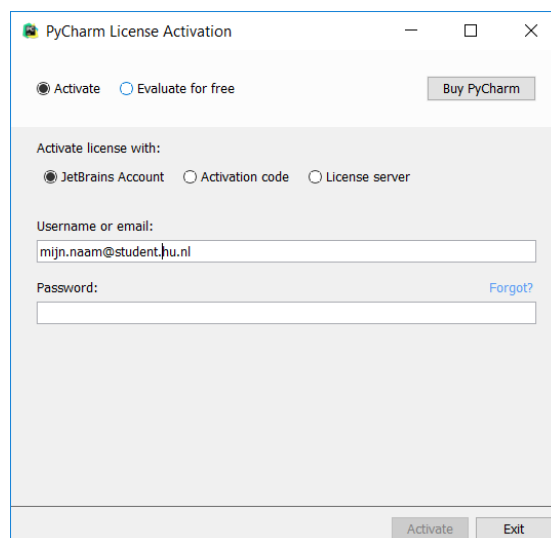
- Vraag [hier](#) toegang tot de JetBrains producten en maak daarbij een JetBrains account aan.
- Download ([hier](#)) de **Professional** editie van Pycharm en installeer deze!

Windows | Linux | Mac

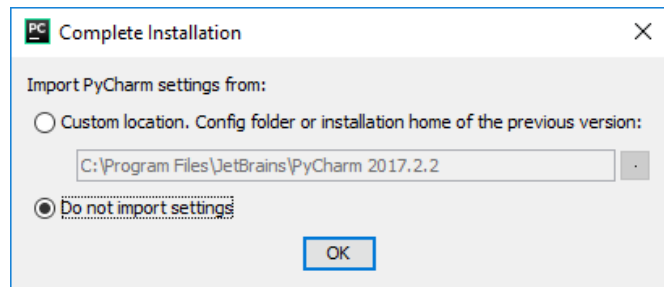
De HU ondersteunt het werken met Python 3 en Pycharm op het **Windows** platform. Het is niet onmogelijk en ook niet verboden om deze combinatie te installeren op het Linux platform (bijvoorbeeld Ubuntu) of Mac OS. De docent kan hier echter niet altijd ondersteuning voor bieden! Als je dit overweegt, lees dan [deze](#) pagina eerst door!

PyCharm!

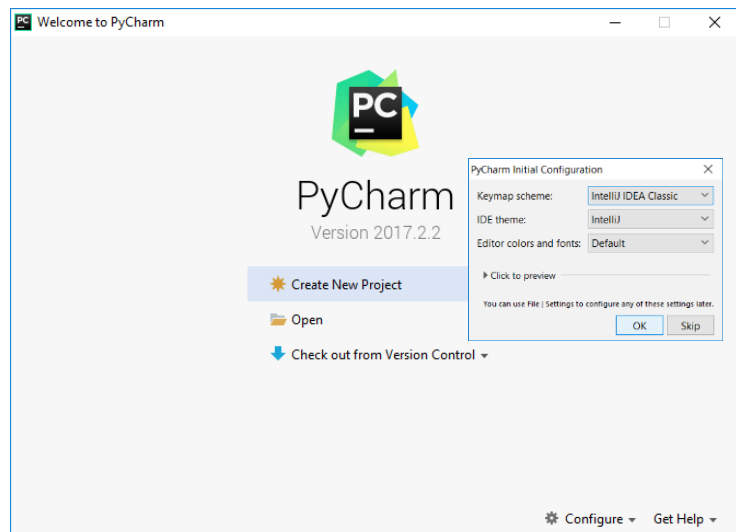
Start PyCharm op. De eerste keer krijg je het venster te zien dat hiernaast is weergegeven om jouw PyCharm licentie te activeren. Log in hier met het eerder aangemaakte JetBrains account:



Daarna krijg je de vraag of je instellingen van eerdere PyCharm edities wilt importeren. Je hebt waarschijnlijk nog niet eerder in Python geprogrammeerd, kies dan voor de onderste optie en klik **OK**:

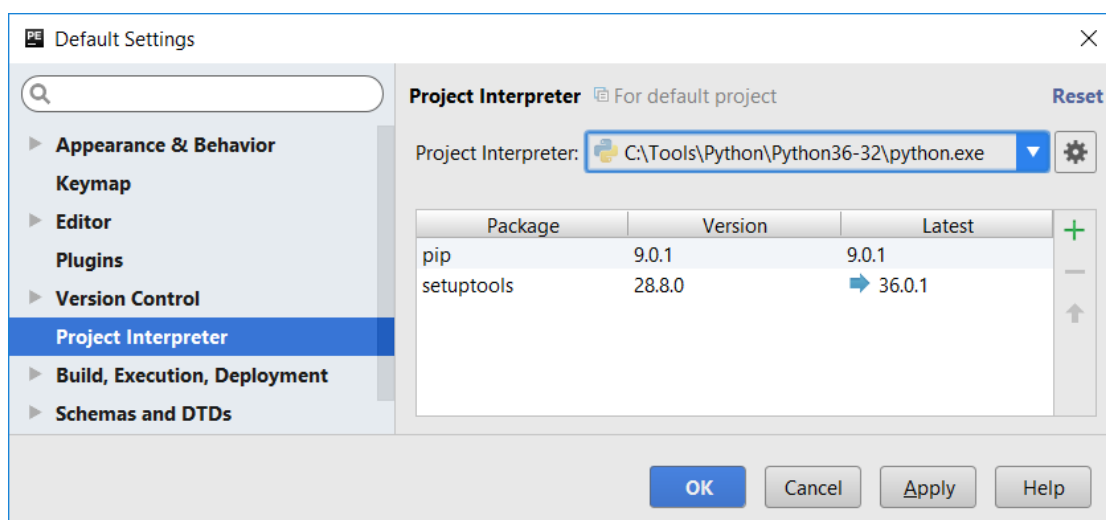


Het startscherm wordt geopend, klik **OK** om de voorgesteld initiële configuratie te accepteren:



We gaan eerst instellen welke versie van Python PyCharm moet gebruiken.

- Klik op **Configure** en daarna op **Settings** (Windows).
- Klik op **Project Interpreter**.
- Kies in het drop-downmenu voor de geïnstalleerde versie van Python (3.6.1) en klik op **OK**!
- **!!** soms staat de juiste versie er (nog) niet tussen. Herstart PyCharm om dit te verhelpen **!!**



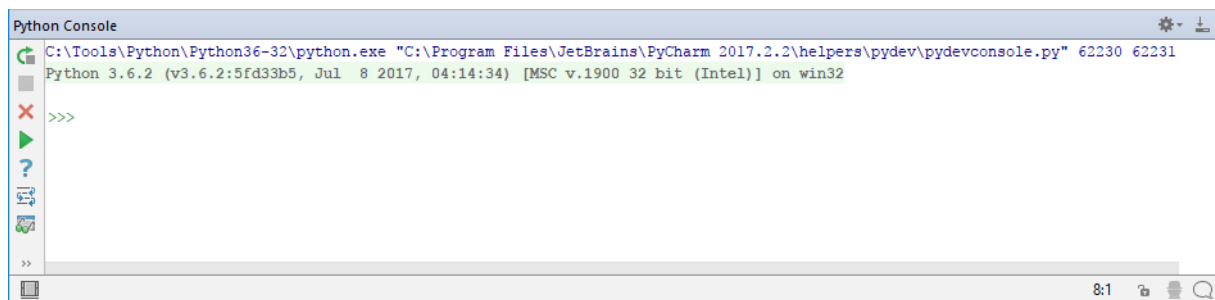
Je bent nu klaar om je eerste programma te schrijven. Kies daarvoor in het hoofdscherm voor Create New Project, wijzig de naam van het project (untitled, dit staat achter de locatie van het project) in een zelfgekozen naam, bijvoorbeeld **Programming**. Klik vervolgens Create.

Les 1: Starten met Python (Perkovic – §1.1 t/m §2.3)

Vorbereiding

Lees ter voorbereiding de literatuur van deze les uit Perkovic (§1.1 t/m 2.3)! Probeer daarna vast (een aantal van) de Perkovic Exercises en/of Practice Exercises van deze les te maken!

Let op: Perkovic gebruikt in deze hoofdstukken de *interactive shell*. Hiermee kan je Python steeds korte opdrachten geven. Na installeren van Python staat deze shell ook op jouw computer. Die kun je zelf starten, maar je kunt de shell ook gebruiken vanuit **PyCharm**, via menu **Tools > Python Console**:



Multimedia (ter ondersteuning, geen vervanging, van Perkovic)



Perkovic Exercises

- 2.12
- 2.13
- 2.14
- 2.18
- 2.19

Practice Exercises

!! Let op: noteer de antwoorden van deze opgaven in je opdracht-portfolio!!

Practice Exercise 1_1 (expressions)

Noteer van elke expressie wat de uitkomst en het type is!

	uitkomst	type
5	5	integer
5.0	5.0	
5 % 2		
5 > 1		
'5'		
5 * 2		

'5' * 2		
'5' + '2'		
5 / 2		
5 // 2		
[5, 2, 1]		
5 in [1, 4, 6]		

Practice Exercise 1_2 (strings)

Schrijf en evalueer Python expressies die de volgende vragen beantwoorden:

- Hoeveel letters zijn er in 'Supercalifragilisticexpialidocious'?
- Komt in 'Supercalifragilisticexpialidocious' de tekst 'ice' voor?
- Is het woord 'Antidisestablishmentarianism' langer dan 'Honorificabilitudinitatibus'?
- Welke componist komt in alfabetische volgorde het eerst: 'Berlioz', 'Borodin', 'Brian', 'Bartok', 'Bellini', 'Buxtehude', 'Bernstein'? Welke het laatst?

Practice Exercise 1_3 (statements)

Schrijf Python statements die het volgende doen:

- Ken de waarde 6 toe aan variabele **a**, en waarde 7 aan variabele **b**.
- Ken aan variabele **c** als waarde het gemiddelde van **a** en **b** toe.
- Ken aan variabele **inventaris** de een lijst van strings toe: 'papier', 'nietjes', en 'pennen'.
- Ken aan variabelen **voornaam**, **tussenvoegsel** en **achternaam** je eigen naamgegevens toe.
- Ken aan variabele **mijnaam** de variabelen van opdracht 4 (met spaties er tussen) toe.

Practice Exercise 1_4 (boolean expressions)

Schrijf booleaanse expressies die van de variabelen van Practice Exercise 1_3 evalueren of:

- 6.75 groter is dan **a** en kleiner **b**.
- de lengte van **inventaris** meer dan 5 keer zo groot is als de lengte van variabele **mijnaam**.
- de lijst **inventaris** leeg is, of juist meer dan 10 items bevat.

Practice Exercise 1_5 (lists)

We gaan een lijst bijhouden met je favoriete artiesten. We gaan de lijst eerst creëren met 1 artiest en dan uitbreiden. Schrijf per stap een expressie om:

- een nieuwe **list** met 1 artiest aan te maken met de naam **favorieten**.
- de lijst uit te breiden met een tweede artiest.
- de tweede artiest te vervangen door een andere naam.

Practice Exercise 1_6 (lists)

Het bereik van een lijst is het verschil tussen het grootste en het kleinste getal. Schrijf een Python expressie die het bereik van een lijst berekent. Als bijvoorbeeld variabele **list** bestaat uit de getallen 3, 7, -2 en 12, dan moet de expressie evalueren naar 14 (verschil tussen 12 en -2). Zorg dat de expressie altijd werkt, ook al bestaat de lijst uit andere waarden!

Les 2: Basisconcepten (Perkovic – §2.3 t/m 3.1)

Vorbereiding

Lees ter voorbereiding de literatuur van deze les uit Perkovic (§2.3 t/m 3.1)! Probeer daarna vast (een aantal van) de Perkovic Exercises en/of Practice Exercises van deze les te maken!


Het boek gebruikt regelmatig de interactive shell, maar in de lessen schrijven we vanaf nu onze programma's vrijwel alleen in Python-bestanden omdat deze ook ingeleverd moeten worden. Je kunt als volgt in PyCharm bestanden maken en uitvoeren:

Start PyCharm (kies in het opstartscherm voor het project dat je in les 1 hebt gemaakt). Maak daarna in het project een nieuwe directory aan:

- Rechtermuisklik in de **Project View** (Alt+1) op de naam van je project, en kies New > Directory
- Kies voor **Directory**
- Kies een logische naam (bijvoorbeeld Les2) en druk op OK
- Klap eventueel het project uit door erop te dubbelklikken zodat je de nieuwe directory ziet

Een **nieuw bestand** aanmaken:

- Rechtermuisklik op de nieuwe directory, kies New > Python File
- Kies voor **Python File**
- Kies als naam **pe2_1** en druk op OK
- Probeer nu in deze file Practice Exercise 2.1 alvast te maken.

Je kunt nu een Python programma schrijven in dit bestand. Uitvoeren ervan kan met de knop  (Ctrl+Shift+F10). Het is belangrijk om overzicht te houden in de code die je schrijft. **Maak daarom per les een nieuwe directory aan in het project.** Bij het miniproject lever je dan een ZIP-bestand in van deze directory met daarin uitwerkingen van alle verplichte opdrachten!!

Multimedia (ter ondersteuning, geen vervanging, van Perkovic)



Perkovic Exercises

- 2.28
- 2.31
- 3.17

!! Let op: de meeste opdrachten zou je ook in de interactieve shell kunnen uitvoeren, maar het is voor je portfolio en de Final Assignment van deze week het handigste als je deze exercises in een Python-bestand programmeert. Je moet er wel aan denken dat je altijd de functie print() moet gebruiken om iets op het scherm te krijgen!!

Practice Exercises

Practice Exercise 2_1 (tuples)

De tuple **letters** kan in willekeurige volgorde de letters A, B en C bevatten. Bijvoorbeeld:

```
letters = ('A', 'C', 'B', 'B', 'C', 'A', 'C', 'C', 'B')
```

Maak een nieuw bestand, bijvoorbeeld **pe2_1.py**, en schrijf een programma dat een nieuwe lijst maakt (en print) met het aantal voorkomens van de letters in alfabetische volgorde. Tuple **letters** bevat 4 keer 'A', 3 keer 'B' en 5 keer 'C'. De lijst die dit programma maakt (en print) is dan: **[2, 3, 4]**.

Practice Exercise 2_2 (getallen, strings en conversie)

De Hogeschool Utrecht wil studenten financieel ondersteunen bij hun studie, afhankelijk van de cijfers die een student haalt. Voor elk cijferpunt krijg je € 30,-. Voor een 1,0 krijgt je dus 30 euro, voor een 2,5 krijgt je 75 euro en voor een 10,0 beloont de HU een student met € 300,-.

Maak variabelen **cijferICOR**, **cijferPROG** en **cijferCSN**. Geef ze alle drie de waarde die jij verwacht dat je voor de betreffende vakken in blok 1 zult gaan halen. Maak nu vervolgens ook de volgende variabelen aan, en bereken de bijbehorende waarden m.b.v. een Python expressie:

gemiddelde	het gemiddelde van de variabelen cijferICOR, cijferPROG en cijferCSN.
beloning	de totale beloning voor deze drie cursussen.
overzicht	een string met een tekstuele omschrijving het gemiddelde en de beloning:

'Mijn cijfers (gemiddeld een 7.5) leveren een beloning van € 675.0 op!'

Print tot slot variabele **overzicht**! Schrijf dit programma in een nieuw bestand, bijvoorbeeld **pe2_2.py**.

Practice Exercise 2_3 (operator voorrang)

Voeg haakjes toe aan de volgende expressies zodat ze naar True evalueren. Print het resultaat!

1. `0 == 1 == 2`
2. `2 + 3 == 4 + 5 == 7`
3. `1 < -1 == 3 > 4`

Schrijf dit programma in een nieuw bestand, bijvoorbeeld **pe2_3.py**. Doe dit vanaf nu bij elke opdracht!

Practice Exercise 2_4 (Input/Output)

Schrijf een programma dat de gebruiker vraagt om zijn **uurloon**, het **aantal uur** dat hij of zij gewerkt heeft en dat daarna het salaris uitprint.

Voorbeelduitvoer:

```
Wat verdien je per uur: 3.80
Hoeveel uur heb je gewerkt: 20
20 uur werken levert 76.0 Euro op
```

Final Assignment: Git & GitHub

Je hebt code geschreven. Natuurlijk wil je niet dat je door een vergissing niet meer terug kan naar een eerdere versie van de source. Dit kan je bereiken met een versiebeheersysteem zoals SVN, CVS, TFS of... **Git**. Deze laatste gaan wij gebruiken. Met Git kan je voor elk project een eigen opslagplaats maken waar oudere versies worden bewaard. Zo'n opslagplaats wordt ook wel **repository** genoemd.

Zo'n versiebeheersysteem op je eigen laptop is al erg handig, maar als je wilt voorkomen dat broncode verloren kan gaan als je computer crasht, of als je wilt samenwerken (zoals in het miniproject) heb je ook een centrale repository nodig waar iedereen bij kan. Daarvoor maken we gebruik van **GitHub**. Dit is een online platform waar je een centrale kopie (met eigen versiebeheer) van de code kunt bijhouden.

Zo kan je vanaf andere apparaten en locaties gemakkelijk bij je eigen code. Ook kan je andere mensen toegang geven tot jouw code, zodat je samen aan een project kunt werken! Volg het stappenplan om kennis te maken met Git en GitHub:

Stap 1: Student-Pack aanvragen

Op je laptop kan je onbeperkt repositories aanmaken, maar op GitHub kan je alleen publiek zichtbare repositories maken, tenzij je een betalende gebruiker bent. Als student kan je echter het **Student-Pack** aanvragen (<https://education.github.com/pack>). Hiermee kan je een onbeperkt aantal **private repositories** aanmaken! Vraag nu eerst het Student-Pack aan!

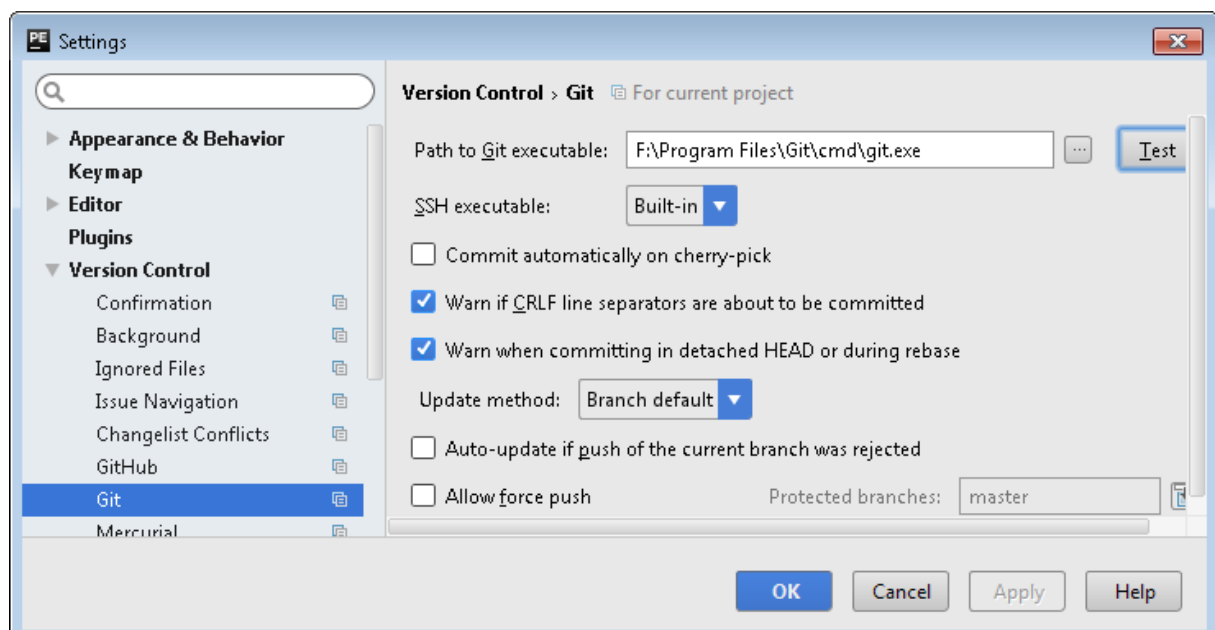
Jouw docent kan vragen om uitwerkingen in te leveren via GitHub; je bent dan voor deze cursus verplicht om dit via een private repository te doen! Geef je docent dan toegang tot jouw repository!

Stap 2: Versiebeheersysteem

De tweede stap is om Git op je eigen laptop te installeren! Op <http://git-scm.com/download/> kan je Git voor jouw besturingssysteem vinden. Installeer Git en accepteer hierbij alle default instellingen.

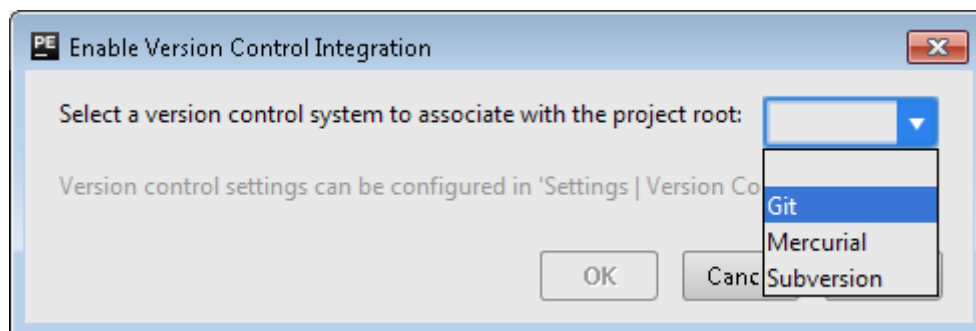
Stap 3: Git configureren in PyCharm

Zorg er nu voor dat je GIT vanuit PyCharm kunt gebruiken: Kies menu File > Settings, klap vervolgens Version Control uit en klik op **Git**. Geef, zoals hieronder, het juiste path op naar het bestand **git.exe**:

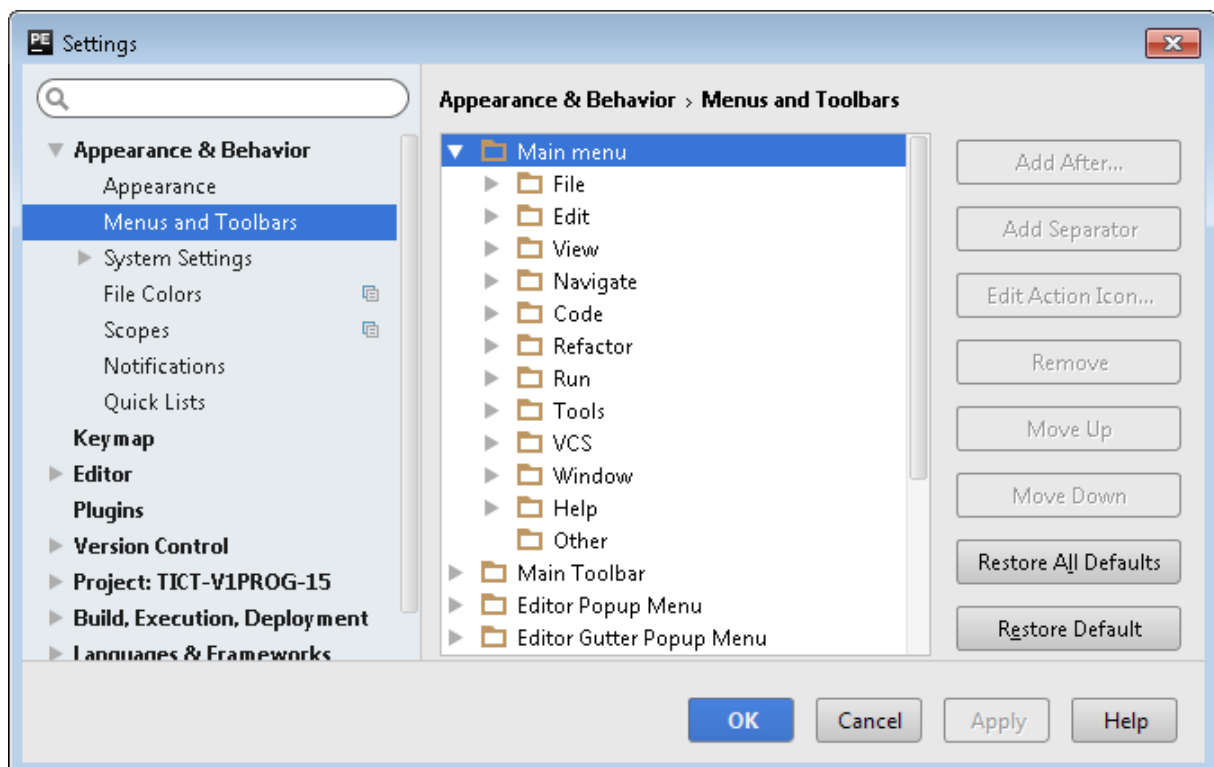


Stap 4: Activeer Git in PyCharm

Je moet nu nog instellen dat jouw project Git gaat gebruiken. Dan kan via menu VCS > 'Enable Version Control Integration'. Kies vervolgens voor Git en klik **OK**:



Let op: het kan zijn dat je geen menu VCS hebt of ziet! Kies dan voor File > Settings, en open via **Appearance & Behaviour** het onderdeel **Menus and Toolbars**. Klik in het rechterdeel van het scherm het **Main menu** aan, en kies **Restore Default**. Daarna ziet het scherm er als volgt uit:



Klik daarna **OK**. Als het goed is, moet het menu CVS nu zichtbaar zijn geworden. Stel nu het project in om Git te gebruiken (zie het begin van deze stap).

Stap 5: Code aan Git toevoegen

Er is nu een (onzichtbare) Git opslagplaats (repository) gemaakt in de directory waar jouw project in staat. Je moet er nu voor zorgen dat de source code terechtkomt in GIT. Heel globaal is de volgorde:

1. Maak een nieuwe file aan in je project (dit is niet nodig als je al een bestand hebt...)
2. Voeg de file toe aan Git. Klik met de rechtermuisknop op de file en kies **Git > + Add**

3. De file is nu onder “source control”. Niet elke wijziging zal echter worden opgeslagen. Je maakt een Python-programma zover af, totdat je een versie hebt die jou bevalt. Op het moment dat je daarmee klaar bent, klik je weer met rechtermuisknop op de file en kiest **Git > Commit file...** Dit werkt overigens ook voor gehele project-directories!

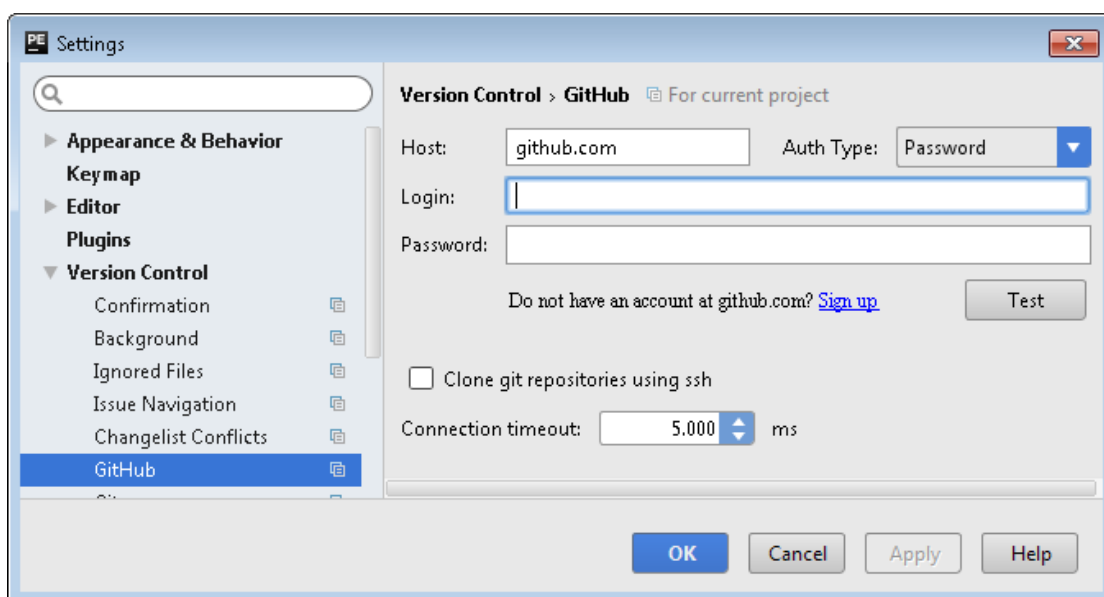
Je zult bij het committen gevraagd worden om een commit-message. Door een **commit** wordt je bestand in de lokale Git repository opgeslagen. Zorg bij iedere commit daarom voor een logische “commit message” zodat anderen en jijzelf later nog begrijpen waarom en wat je hebt gecommit. Bij een eerste commit wordt je gevraagd om een Git username en e-mailadres. Deze worden alleen gebruikt zodat achterhaald kan worden wie code heeft gecommit.

4. Pas wat aan in je script (regel erbij / regel eraf / regel aanpassen). Vergelijk je bestand nu met de versie dit is opgeslagen in Git (rechtermuisklik op het bestand, **Git > Compare with latest...**).

Stap 6: GitHub gebruiken

De lokale Git database is een goede oplossing, maar soms wil je met meerdere mensen samen werken of gewoon een backup hebben op een ander systeem. We kiezen nu voor opslag in de cloud door middel van GitHub.

1. Zorg ervoor dat je GitHub vanuit PyCharm kunt gebruiken, via menu File > Settings. Klap daarna onderdeel **Version Control** uit en klik op **GitHub**. Kies voor Auth(entication) Type **password** en voer je accountgegevens in (Login en Password). Test of er verbinding gemaakt kan worden:



2. Nu alleen nog je project publiceren op GitHub! Vanuit PyCharm: menu VCS > Import into Version Control > Share Project on GitHub. Geef de repository een logische naam. Kies **Share**.
3. Login op GitHub. Kan je daar je code terugvinden?
4. Je kan nu op elk moment je wijzigingen “committen” (lokaal opslaan) of “committen en pushen” (lokaal en ook op GitHub bewaren).




Tot slot: wil je de code van iemand anders binnenhalen (“clonen”), dan kan dat via menu VCS > Checkout from Version Control > GitHub... Maak afspraken met teamleden wie aan welk stuk code werkt. Als je tegelijkertijd een regel code wijzigt, moet dat conflict bij **pushen** opgelost worden!

Les 3: Control Structures (Perkovic – §3.2)

Voorbereiding

Lees ter voorbereiding de literatuur van deze les uit Perkovic (§3.2)! Probeer daarna vast (een aantal van) de Perkovic Exercises en/of Practice Exercises van deze les te maken!

Multimedia (ter ondersteuning, geen vervanging, van Perkovic)

		
if+else statement	if-statement	for-loop

Perkovic Exercises

- 3.18
- 3.22
- 3.23

!! Let op: de meeste opdrachten kun je in de interactieve shell uitvoeren, maar het is voor je portfolio het handigste als je deze exercises in een Python-bestand programmeert. Je moet er wel aan denken dat je altijd de functie `print()` moet gebruiken om iets op het scherm te krijgen!!

Practice Exercises

Practice Exercise 3_1 (if-statement)

Schrijf een programma dat de gebruiker vraagt om de score van een multiplechoice toets. Het programma bepaalt of het resultaat voldoende is. Bij meer dan 15 punten is de deelnemer geslaagd!

Voorbeelduitvoer:

```
Geef je score: 18
Gefeliciteerd!
Met een score van 18 ben je geslaagd!
```

Waarschijnlijk heb je de bovenstaande uitvoer geprogrammeerd met 2 **print()**-opdrachten. Wat gebeurt er als je de tweede `print()`-opdracht niet recht onder de eerste zou plaatsen maar bijvoorbeeld recht onder de 'i' van het **if-statement**?

Practice Exercise 3_2 (if met 2 booleaanse operators)

Je mag stemmen als je 18 of ouder bent **en** in het bezit bent van een Nederlands paspoort. Schrijf een programma dat de leeftijd van de gebruiker vraagt en of diegene een Nederlands paspoort heeft (ja/nee). Als aan **beide** voorwaarden is voldaan, print dan dat de gebruiker mag stemmen! Doe dit weer in een nieuw bestand, bijvoorbeeld **pe2_3.py**. In de conditie van een **if-statement** kun je meerdere voorwaarden tegelijk controleren met bijvoorbeeld **or** of **and** (zie Perkovic blz 18 en 19). Voor deze opgave mag je daarom **maximaal 1** keer een **if-statement** gebruiken.

Voorbeelduitvoer:

```
Geef je leeftijd: 18
Nederlands paspoort: ja
```

Gefeliciteerd, je mag stemmen!

Practice Exercise 3_3 (if-else)

Pas de uitwerking van exercise 3_2 aan en geef **ook** een melding als de gebruiker **niet** mag stemmen!

Practice Exercise 3_4 (for + if)

Schrijf een for-loop die over een lijst met strings itereert, en van elk woord de eerste twee karakters print. De lijst ['maandag', 'dinsdag', 'woensdag'] zou dus moeten resulteren in:

ma
di
wo

Practice Exercise 3_5 (for + if)

Schrijf een for-loop die over een lijst met getallen itereert, en alle even getallen print.

Practice Exercise 3_6 (for + if)

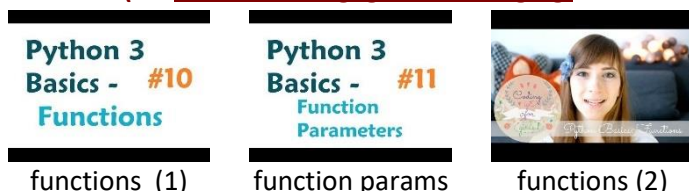
Schrijf een for-loop die langs alle letters van een string loopt en de letter uitprint, maar alleen als het een klinker is ('aeiou'). Gebruik string `s = "Guido van Rossum heeft programmeertaal Python bedacht."`

Les 4: Functions (Perkovic – §3.3 t/m 3.5)

Vorbereiding

Lees ter voorbereiding de literatuur van deze les uit Perkovic (§3.3 t/m 3.5)! Probeer daarna vast (een aantal van) de Perkovic Exercises en/of Practice Exercises van deze les te maken!

Multimedia (ter ondersteuning, geen vervanging, van Perkovic)



Perkovic Exercises (/ Problems)

- 3.33
- 3.40
- 3.42 (optioneel)
- 3.34
- 3.41
- 3.44

Practice Exercises

Practice Exercise 4_1 (functie met 3 parameters)

Maak een nieuwe Python file aan voor deze opdracht (vanaf nu is dat standaard en zal dat niet steeds meer bij de opdracht staan). Schrijf (en test) de functie **som()** die 3 parameters heeft: **getal1**, **getal2** en **getal3**. De return-waarde van de functie moet de som (optelling) van deze parameters zijn!

Practice Exercise 4_2 (functie met list-parameter)

Schrijf (en test) de functie **som()** die 1 parameter heeft: **getallenLijst**. Ga ervan uit dat dit een list is met integers. De return-waarde van de functie moet de som (optelling) van de getallen in de lijst zijn! Tip: bekijk nog eens de list-functies (Perkovic, blz. 28).

Practice Exercise 4_3 (functie met if)

Schrijf (en test) de functie **lang_genoeg()** die voor Efteling-attracties bepaalt of een gebruiker in een attractie mag. De functie heeft één parameter, namelijk de **lengte** van de gebruiker in centimeters. Als de gebruiker 120 centimeter of langer is de return-waarde van de functie “Je bent lang genoeg voor de attractie!”, anders is de melding “Sorry, je bent te klein!”.

Practice Exercise 4_4 (functie met if)

Schrijf (en test) een functie **new_password()** die 2 parameters heeft: **oldpassword** en **newpassword**. De return-waarde is **True** als het nieuwe password voldoet aan de eisen. Het nieuwe password wordt alleen geaccepteerd als het **verschilt** van het oude password **en** als het minimaal 6 tekens lang is. Als dat niet zo is, is de return-waarde **False**.

Optioneel: zorg dat het nieuwe password minstens 1 cijfer moet bevatten!

Practice Exercise 4_5 (functie met list-parameter en for-loop)

Schrijf (en test) de functie **kwadraten_som()** die 1 parameter heeft: **grondgetallen**. Dit is een list is met integers. De return-waarde van de functie is de som van de kwadraten van alle **positieve** getallen in de lijst! Een lijst van [4, 5, 3, -81] heeft als return-waarde dus 50 (16 + 9 + 25).

Practice Exercise 4_6 (functie met (im)mutable parameter)

Schrijf (en test) functie **wijzig()** met één parameter: **letterlijst**. Zorg dat de functie de lijst leegt en de letters ['d', 'e', 'f'] toevoegt. Er is **geen** return-waarde! Test je programma als volgt:

```
lijst = ['a', 'b', 'c']
print(lijst)
wijzig(lijst)
print(lijst)
```

Uitvoer:

```
['a', 'b', 'c']
['d', 'e', 'f']
```

- Waarom kun je in de functie niet de opdracht `lijst = ['d', 'e', 'f']` geven?
- Werkt jouw functie ook met een string-parameter? Probeer dit! Waarom werkt het wel/niet?
- Welke rol speelt (im)mutabiliteit in deze vraag?

Final Assignment: NS-functies

De NS heeft standaardtarieven voor treinreizen, maar onder sommige omstandigheden krijgen reizigers korting. Bijvoorbeeld omdat ze in een bepaalde leeftijdscategorie vallen. In deze opdracht maak je twee functies: **standaardtarief(..)** en **ritprijs(..)**. De eerste functie bepaalt het standaardbedrag voor een bepaalde rit. De tweede functie maakt hier gebruik van, maar bepaalt zelf ook nog welke kortingen van toepassing zijn en levert als return-waarde de definitieve prijs.

Deel 1: functie standaardprijs

Schrijf functie **standaardprijs(afstandKM)**. Iedere treinrit kost 80 cent per kilometer, maar als de rit langer is dan 50 kilometer betaal je een vast bedrag van €15,- plus 60 cent per kilometer. Ga bij invoer van negatieve afstanden uit van een afstand van 0 kilometer (prijs is dan dus 0 Euro).

Deel 2: functie ritprijs

Schrijf nu ook de functie **ritprijs(leeftijd, weekendrit, afstandKM)**. De parameter weekendrit is een boolean die aangeeft of de rit in het weekend plaats zal hebben (True) of niet (False). Het eerste wat de functie **ritprijs** moet doen, is het **aanroepen** van de functie **standaardprijs**, waarbij de afstand in kilometers doorgegeven moet worden, om de standaardprijs voor de rit op te vragen. Aan de hand van de return-waarde kan de ritprijs worden berekend. De regels zijn als volgt:

Op werkdagen reizen kinderen (onder 12 jaar) en ouderen (65 en ouder) met 30% korting. In het weekend reist deze groep met 35% korting. De overige leeftijdsgroepen reizen betalen de gewone prijs, behalve in het weekend, dan reist deze leeftijdsgroep met 40% korting.

Deel 3: testen

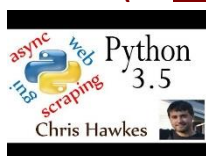
De diverse parameters kunnen voor heel veel verschillende executie-paden zorgen. Dat wil zeggen dat de code op heel veel verschillende manieren doorlopen kan worden. Verstandig is om je programma in ieder geval op grenswaarden te testen. Voor leeftijd test je dan leeftijden 11, 12, 64 en 65, maar voor al die leeftijden ook weer of het een weekendrit is of niet ($4 \times 2 = 8$ combinaties). Voor elk van die 8 combinaties kan er ook weer een korte of lange treinrit (of een negatieve afstand) ingevoerd worden, wat $8 \times 3 = 24$ testpaden oplevert. Test jouw methode **ritprijs** zo goed mogelijk!

Les 5: Strings, Text Data, and File I/O (Perkovic – §4.1 t/m 4.3)

Voorbereiding

Lees ter voorbereiding de literatuur van deze les uit Perkovic (§4.1 t/m 4.3)! Probeer daarna vast (een aantal van) de Perkovic Exercises en/of Practice Exercises van deze les te maken!

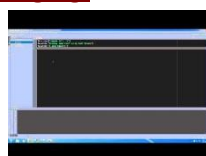
Multimedia (ter ondersteuning, geen vervanging, van Perkovic)



I/O (4)



I/O (2)



I/O (3)



I/O met console

Perkovic Exercises (/ Problems)

- 4.13
- 4.15
- 4.18
- 4.29
- 4.30
- 4.32

Practice Exercises

Practice Exercise 5_1 (formatting)

Schrijf een functie **convert()** waar je een temperatuur in graden Celsius (als parameter van deze functie) kunt omzetten naar graden Fahrenheit. Je kunt de temperatuur in Fahrenheit berekenen met de formule $T_{(F)} = T_{(C)} \times 1.8 + 32$. Dus $25\text{ }^{\circ}\text{C} = 25 \times 1.8 + 32 = 77\text{ }^{\circ}\text{F}$.

Schrijf nu ook een tweede functie **table()** waarin je met een for-loop van $-30\text{ }^{\circ}\text{C}$ t/m $40\text{ }^{\circ}\text{C}$ in stappen van 10 graden de temperatuur in Fahrenheit print. Zorg middels een geformatteerde output voor dezelfde precisie en uitlijning als het voorbeeld hieronder:

Uitvoer:

F	C
-22.0	-30.0
-4.0	-20.0
14.0	-10.0
32.0	0.0
50.0	10.0
68.0	20.0
86.0	30.0
104.0	40.0

Practice Exercise 5_2 (files lezen)

Maak met behulp van PyCharm (of Notepad) het onderstaande bestand '**kaartnummers.txt**', dat bestaat uit klantenkaartnummers en namen. Op iedere regel staan de gegevens van 1 kaart:

Kaartnummers.txt (met achter elke regel een enter!):

```
325255, Jan Jansen
334343, Erik Materus
235434, Ali Ahson
645345, Eva Versteeg
534545, Jan de Wilde
```

```
345355, Henk de Vries
```

Schrijf een Python programma waarmee je het bestand opent, en splits elke regel op de komma en zorg dat de uitvoer (op het scherm) is zoals op de volgende pagina is weergegeven! Vergeet niet het bestand te sluiten!

Uitvoer:

```
Jan Jansen heeft kaartnummer: 325255
Erik Materus heeft kaartnummer: 334343
Ali Ahson heeft kaartnummer: 235434
Eva Versteeg heeft kaartnummer: 645345
Jan de Wilde heeft kaartnummer: 534545
Henk de Vrie heeft kaartnummer: 345355
```

Practice Exercise 5_3 (files lezen)

Schrijf een programma dat het bestand **kaartnummers.txt** opnieuw opent en het aantal regels en het grootste kaartnummer in het bestand bepaalt. Print deze gegevens vervolgens uit:

Uitvoer:

```
Deze file telt 6 regels
Het grootste kaartnummer is: 645345 en dat staat op regel 4
```

Practice Exercise 5_4 (files schrijven)

Bij een marathonwedstrijd worden bij een controlepost alle voorbijkomende hardlopers genoteerd. De gegevens van elke hardloper worden in het bestand **hardlopers.txt** opgeslagen. Schrijf een programma waarmee een tekstbestand wordt **aangemaakt** (als het bestand nog niet bestaat) **of aangevuld** (gebruik de append-mode) met de gegevens van één hardloper (inlezen van toetsenbord).

Let op: je zult je programma in deze opdracht steeds opnieuw moeten uitvoeren voor elke nieuwe hardloper. Om dit te voorkomen zou je een while-loop kunnen gebruiken, maar die behandelen we pas volgende les. Je kunt er natuurlijk voor kiezen om daar alvast naar te kijken (niet verplicht).

Hardlopers.txt (na registratie van 5 hardlopers):

```
Thu 10 Mar 2016, 10:45:52, Miranda
Thu 10 Mar 2016, 10:46:04, Piet
Thu 10 Mar 2016, 10:47:27, Sacha
Thu 10 Mar 2016, 10:48:33, Karel
Thu 10 Mar 2016, 10:48:42, Kemal
```

Gebruik de Python-functie **strftime()** om de huidige tijd in een keurig formaat om te zetten! Je kunt de [documentatie](#) van deze functie bekijken of in het boek van Perkovic (blz. 106-107) lezen hoe je onderstaande voorbeeld kunt ombouwen tot de gewenste tijdsaanduiding! Op de volgende bladzijde van dit werkboek staat ook nog een voorbeeld-toepassing van de functie.

Functie strftime():

```
import datetime
vandaag = datetime.datetime.today()
s = vandaag.strftime("%a %d %b %Y")
print(s)
```

Uitvoer van bovenstaande code:

```
Mon 12 Sep 2016
```

Practice Exercise 5_5 (string functions)

Schrijf functie `gemiddelde()`, die de gebruiker vraagt om een willekeurige zin in te voeren. De functie berekent vervolgens de gemiddelde lengte van de woorden in de zin en print dit uit.

Les 6: Execution Control Structures (Perkovic – §5.1 t/m 5.3)

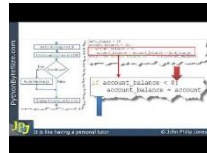
Vorbereiding

Lees ter voorbereiding de literatuur van deze les uit Perkovic (§5.1 t/m 5.3)! Probeer daarna vast (een aantal van) de Perkovic Exercises en/of Practice Exercises van deze les te maken!

Multimedia (ter ondersteuning, geen vervanging, van Perkovic)



for-loop



selection

Perkovic Exercises (/ Problems)

- 5.15
- 5.19
- 5.26
- 5.16
- 5.23
- 5.39

Practice Exercises

Practice Exercise 6_1 (decision control)

Schrijf een **functie seizoen(maand)** die als parameter maandnummer meekrijgt. Het functie-resultaat is het seizoen die bij die maand hoort. Nummers 3 t/m 5 horen bij **lente**, 9 t/m 11 **herfst** etc.

Practice Exercise 6_2 (lists)

Schrijf een nieuw programma waarin een lijst met minimaal 10 strings wordt ingelezen. Het programma plaatst alle vier-letter strings uit de ingelezen lijst in een nieuwe lijst en drukt deze af. Inlezen van een lijst kan met `eval(input("Geef lijst met minimaal 10 strings: "))`.

Uitvoer:

```
Geef lijst met minimaal 10 strings: ["boter", "kaas", "bier", "pizza",
"thee", "drop", "koek", "cola", "boterham", "stampot"]
De nieuw-gemaakte lijst met alle vier-letter strings is:
['kaas', 'bier', 'thee', 'drop', 'koek', 'cola']
```

Practice Exercise 6_3 (lists)

Gegeven is variabele `invoer = "5-9-7-1-7-8-3-2-4-8-7-9"`. Schrijf een nieuw programma waarin je deze variabele splitst in een lijst van getallen en print de gesorteerde lijst. Bepaal en print na het opsplitsen de grootste waarde, kleinste waarde, aantal getallen, de som en het gemiddelde!

Uitvoer:

```
Gesorteerde list van ints: [1, 2, 3, 4, 5, 7, 7, 7, 8, 8, 9, 9]
Grootste getal: 9 en Kleinste getal: 1
Aantal getallen: 12 en Som van de getallen: 70
Gemiddelde: 5.833333333333333
```

Practice Exercise 6_4 (two-dimensional-lists)

Van elke student worden 3 cijfers bijgehouden. In de lijst **studentencijfers** staan de gegevens van 4 studenten. Schrijf code voor twee functies die het gemiddelde cijfer voor iedere student en het gemiddelde van alle studenten berekent. De eerste functie heeft als return-waarde een één-dimensionale-lijst met alle gemiddelden per student, en de tweede functie heeft als return-waarde een int-getal met het gemiddelde van alle studenten! Maak het onderstaande programma af:

```
studentencijfers = [ [95, 92, 86], [66, 75, 54], [89, 72, 100], [34, 0, 0] ]

def gemiddelde_per_student(studentencijfers):
    . . . . .
    return antw

def gemiddelde_van_allen_students(studentencijfers):
    . . . . .
    return antw

print(gemiddelde_per_student(studentencijfers))
print(gemiddelde_van_allen_students(studentencijfers))
```

Practice Exercise 6_5 (nested loop)

Schrijf een programma met twee for-loops in elkaar (nested) om de tafels van 1 t/m 10 uit te printen.

Uitvoer:

```
1 x 1 = 1
2 x 1 = 2
...
10 x 10 = 100
```

Final Assignment: Bagagekluizen

Op veel stations kun je een bagagekluis huren. We willen graag dat jij de software realiseert voor de verhuur van de kluizen. Er zijn door de beheerders de volgende randvoorwaarden gesteld aan het systeem:



- Er zijn in totaal 12 kluizen (nr. 1 t/m 12).
- Elke kluis moet beveiligd zijn met een code (minimaal 4 tekens) die de klant zelf bedenkt.
- Als de stroom uitvalt mag de data niet verloren gaan en daarom moet je een bestand gebruiken voor de opslag van gegevens zoals kluisnummers en wachtwoorden. Een voorbeeld van zo'n bestand ('kluizen.txt') zie je hieronder:

```
11;6754
1;geheim
5;kluisvanpietje
12;z@terd@g
```

In het voorbeeld zijn er 4 kluizen in gebruik (nummer 11, 1, 5 en 12). Het nummer wordt gevolgd door een puntkomma, waarachter de code staat die de gebruiker heeft ingevoerd voor de betreffende kluis. Op elke regel staat informatie van maximaal 1 kluis!

Als je programma start, moet je het onderstaande menu te zien krijgen:

```
1: Ik wil weten hoeveel kluizen nog vrij zijn
2: Ik wil een nieuwe kluis
3: Ik wil even iets uit mijn kluis halen
4: Ik geef mijn kluis terug
```

- Kies je voor optie 1, dan krijg je te zien hoeveel kluizen er nog beschikbaar zijn.
- Kies je voor optie 2, dan controleert het programma of er kluizen vrij zijn. Zo ja, dan kan je een zelfgekozen code invoeren. Je krijgt een melding welke kluis je krijgt (bij een geldige code).
- Kies je voor optie 3, dan moet je jouw kluisnummer en code invoeren. Is de combinatie correct, dan krijg je de melding dat je toegang krijgt tot jouw kluis.
- Kies je voor optie 4, dan moet je jouw kluisnummer en code invoeren. Is de combinatie correct, dan zal jouw kluis weer beschikbaar zijn voor andere gebruikers. **Deze optie is optioneel!**

Hoe moet je beginnen?

1. Maak in je project (in PyCharm) een leeg bestand aan met de naam **kluizen.txt**.
2. Schrijf code om het menu met de opties 1-4 te printen. Laat de gebruiker de menukeuze als int-waarde invoeren. Geef een foutmelding als de gebruiker een ongeldige waarde invoert.
3. Schrijf voor elke menukeuze een functie die bijbehorende acties uitvoert. De functie moet aangeroepen worden zodra de gebruiker het betreffende getal in heeft ingevoerd. Om je op weg te helpen een suggestie hoe je deze opties kunt realiseren:

Functie `def toon_aantal_kluizen_vrij()`: lees alle regels van **kluizen.txt** in (zie ook les 5). Het aantal regels is het aantal bezette kluizen. Het is dan eenvoudig uit te rekenen hoeveel er nog vrij zijn. Het resultaat print je uit voor de gebruiker.

Functie `def nieuwe_kluis()`: maak een **list** met alle kluisnummers. Lees dan alle regels van **kluizen.txt** in, en doorloop deze met een for-loop. Elk kluisnummer dat je tegenkomt in **kluizen.txt** verwijder je uit de eerder gemaakte **list**. Hierna kan je controleren of er nog kluisnummers in de **list** staan. Zo ja, laat de gebruiker dan een code invoeren, en selecteer het eerste getal uit de lijst met beschikbare nummers. Schrijf het nummer en de code naar **kluizen.txt**. Geef de gebruiker een melding welk kluisnummer hij/zij krijgt. Meld het ook als er geen kluizen meer beschikbaar zijn.t!

Functie `def kluis_openen()`: Vraag de gebruiker om het kluisnummer en de kluiscode. Lees alle regels van **kluizen.txt** in en doorloop deze met een for-loop. Kom je de combinatie van kluisnummer en kluiscode tegen, dan heeft de gebruiker een correct combinatie ingevoerd. Geef hiervan een melding. Geef ook een melding als de ingevoerde gegevens niet kloppen!

Functie `def kluis_teruggeven()`: **Let op:** deze optie is **optioneel!** Vraag de gebruiker om een kluisnummer en kluiscode. Klopt dat, dan moet je de betreffende regel uit **kluizen.txt** verwijderen en de gebruiker melden dat de kluis is vrijgegeven. Werk hiervan de stappen zelf verder uit!

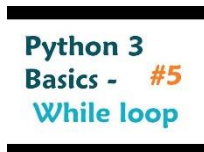
*Je kunt er **eventueel** ook voor zorgen dat je het programma niet steeds opnieuw hoeft te starten. Dat kan door de code om het menu te printen, in een loop te plaatsen, en het menu uit te breiden met optie **5: 'ik wil stoppen'**. Het handigste is om een while-loop toe te passen i.p.v. een for-loop, omdat je niet van tevoren weet hoe vaak de gebruiker het menu wil gebruiken! De while-loop bespreken we in de volgende les, maar je kunt je alvast hierin verdiepen ter voorbereiding!*

Les 7: Control Structures & Dictionaries (Perkovic – §5.4 t/m 6.1)

Voorbereiding

Lees ter voorbereiding de literatuur van deze les uit Perkovic (§5.4 t/m 6.1)! Probeer daarna vast (een aantal van) de Perkovic Exercises en/of Practice Exercises van deze les te maken!

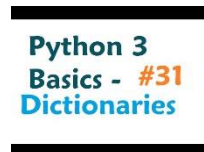
Multimedia (ter ondersteuning, geen vervanging, van Perkovic)



while-loop



loop-control



dictionaries



dict functions



control flow

Perkovic Exercises (/ Problems)

- 5.42
- 6.20
- 6.26

Practice Exercises

Practice Exercise 7_1 (while-loop)

Schrijf een while-loop die steeds getallen van de gebruiker vraagt tot deze 0 ingeeft:

Uitvoer:

```
Geef een getal: 5
Geef een getal: -1
Geef een getal: 0
Er zijn 2 getallen ingevoerd, de som is: 4
```

Practice Exercise 7_2 (while-loop)

Schrijf een nieuw programma waarin de gebruiker een woord moet invoeren. Dit woord moet uit vier letters bestaan. Is dat niet zo, dan wordt een foutmelding getoond en moet de gebruiker opnieuw een woord invoeren, net zolang totdat er een woord is ingevoerd dat uit vier letters bestaat. Dan eindigt het programma. Gebruik in ieder geval een **while-loop**, gecombineerd met het **break**-statement!

Uitvoer:

```
Geef een string van vier letters: worst
worst heeft 5 letters
Geef een string van vier letters: oliebol
oliebol heeft 7 letters
Geef een string van vier letters: drop
Inlezen van correcte string: drop is geslaagd
```

Practice Exercise 7_3 (dict)

Maak een dictionary aan met cijfers voor een cursus. De studentnamen zijn de sleutels en de cijfers zijn de waarden in de dictionary. Zorg dat er minimaal 8 cijfers in de dictionary staan. Schrijf vervolgens code om op basis van die dictionary alle cijfers (met naam) boven een 9,0 te printen.

Practice Exercise 7_4 (file & dict)

Een 'ticker symbol' is een unieke afkorting om aandelen van een bepaald bedrijf aan de beurs te identificeren. YHOO is bijvoorbeeld het ticker symbol van Yahoo, terwijl GOOG dat is voor Google. Gegeven is het onderstaande bestand met **ticker symbols**:

```
YAHOO:YHOO
GOOGLE INC:GOOG
Harley-Davidson:HOG
Yamana Gold:AUY
Sotheby's:BID
inBev:BUD
```

Schrijf een functie **ticker(filename)**. De functie leest uit de file zowel de bedrijfsnaam als het bijbehorende ticker-symbool en slaat die op in een dictionary. Dit is de return-waarde van de functie.

Schrijf nu ook een programma die deze functie gebruikt om als iemand een ticker-symbool ingeeft de bedrijfsnaam te printen, en andersom. Het programma stopt hierna:

Uitvoer:

```
Enter Company name: YAHOO
Ticker symbol: YHOO

Enter Ticker symbol: BUD
Company name: inBev
```

Practice Exercise 7_5 (dict)

Schrijf een functie **namen()** die de gebruiker steeds opnieuw vraagt om de voornaam van een student in de klas. Als de gebruiker een lege tekst invoert, stopt het programma en moet van elke naam geprint worden hoe vaak de naam in de klas voorkomt. Plaats de namen als sleutel (key) in een dict, en hou als waarde (value) bij hoe vaak ze zijn voorgekomen. Zie uitvoer op de volgende bladzijde!

Uitvoer:

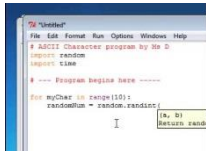
```
>>> namen()
Volgende naam: Valeria
Volgende naam: Bob
Volgende naam: Valeria
Volgende naam: Amelie
Volgende naam: Bob
Volgende naam:
Er is 1 student met de naam Amelie
Er zijn 2 studenten met de naam Bob
Er zijn 2 studenten met de naam Valeria
```

Les 8: Containers (Sets and Chars) (Perkovic – §6.2 t/m 6.4)

Vorbereitung

Lees ter voorbereiding de literatuur van deze les uit Perkovic (§6.2 t/m 6.4)! Probeer daarna vast (een aantal van) de Perkovic Exercises en/of Practice Exercises van deze les te maken!

Multimedia (ter ondersteuning, geen vervanging, van Perkovic)



Python & ascii



sets

Perkovic Exercises (/ Problems)

- 6.11
- 6.17
- 6.34 (lastig, optioneel)

Practice Exercises

Practice Exercise 8_1 (sets)

Hieronder staat een afbeelding van een tweetal trajecten (bruin en groen). Zet beide trajecten allebei in een set met de namen “bruin” en “groen”...

Print daarna eerst met een set-functie welke plaatsen in beide trajecten worden aangedaan (de overeenkomst).

Gebruik vervolgens opnieuw een set-functie om te printen hoe het traject “bruin” verschilt van het traject “groen”. Je moet dan dus op het scherm zien welke plaatsen van traject “bruin” ze **niet allebei** aandoen!

Print ook alle stations op beide trajecten uit. Print elk station maar 1! Gebruik weer een set-functie!



Practice Exercise 8_2 (random)

Schrijf functie **monopolyworp()**, die een het gooien van twee dobbelstenen voor het spel **Monopoly** simuleert en afdruckt. Je mag nogmaals gooien als beide stenen dezelfde waarde hebben. Zorg dat de functie die worpen ook simuleert! Na driemaal dubbel moet de speler naar de gevangenis!

Uitvoer:

```
>>> monopolyworp()
1 + 4 = 5
>>> monopolyworp()
5 + 5 = 10 (dubbel)
5 + 4 = 9
>>> monopolyworp()
3 + 3 = 6 (dubbel)
1 + 1 = 2 (dubbel)
2 + 2 = (direct naar gevangenis)
```

Practice Exercise 8_3 (ascii)

Als extra beveiliging wil de NS op haar E-ticket nog een unieke code afbeelden. Er is gekozen voor een hele eenvoudige beveiliging: Neem de naam van de gebruiker+beginstation+eindstation, vertaal elk karakter naar ASCII en maak die waarde 3 groter. De "a" wordt dus een "d", de "b" wordt een "e", etc. De "A" wordt een "D", de "W" wordt een "Z", etc. En de spatie " " wordt een "#".



DEC	0	16	32	48	64	80	96	112	128	144	160	176	192	208	224	240
0			space	0	@	P	`	p	€	□	abp	°	À	Ð	à	ð
1			!	1	A	Q	a	q	□	'	j	±	Á	Ñ	á	ñ
2			"	2	B	R	b	r	,	'	g	z	Â	Ò	â	ò
3			#	3	C	S	c	s	f	"	é	z	Ã	Ó	ã	ó
4			\$	4	D	T	d	t	"	"	u	'	Ä	Ô	ä	ô
5			%	5	E	U	e	u	...	•	¶	µ	Å	Ö	å	ö
6			&	6	F	V	f	v	†	-		¶	Æ	Ø	æ	ø
7			'	7	G	W	g	w	‡	-	§	·	Ç	×	ç	×
8			(8	H	X	h	x	^	"	-	ˆ	È	Θ	è	θ
9	TAB)	9	I	Y	i	y	%	™	@	ˆ	É	Ú	é	ú	
10	LF	*	:	J	Z	j	z	Š	š	*	°	Ê	Û	ê	û	
11		+	;	K	[k	{	<	>	«	»	Ë	Ü	ë	ü	
12		,	<	L	\	l		Œ	œ	˜	¼	Ì	Ý	ì	ý	
13	CR	-	=	M]	m	}	□	□		½	Í	Þ	í	þ	
14		.	>	N	^	n	~	Ž	ž	@	¾	Î	ß	î	ÿ	
15		/	?	O	_	o	□	□	Ÿ	-	¿	Ï	ä	ï	ä	ÿ

Zorg ervoor dat je code maakt die deze uitvoer maakt op basis van de invoer van de gebruiker. Schrijf een functie: `def code(invoerstring):` die ieder teken van `invoerstring` omzet naar zijn rangordnummer met bibliotheekfunctie `ord()`, en – na er 3 bij te hebben opgeteld – die int-waarde weer terug vertaalt naar het bijbehorende ASCII-teken met bibliotheekfunctie `chr()`.

Voorbeeld: `code("RutteAlkmaarDen Helder")` levert op: `UxwwhDonpdduGhq#Khoghu`

Practice Exercise 8_4 (containers)

De volgende tabel toont verschillende Python container-types in de eerste kolom, en eigenschappen die deze types wel of niet hebben in de volgende kolommen. Maak deze tabel af door in alle cellen JA in te vullen als dat datatype die eigenschap heeft, of NEE als dat niet zo is. Zoek het antwoord in het boek of schrijf zelf een programma om te controleren of je antwoord klopt!

	Geordend	Muteerbaar	Iterable	Dubbele waarden toegestaan
Tuple				
Dictionary				
Set				
List				

Final Assignment: NS-kaartautomaat

In deze opdracht maak je een NS-kaartautomaat. Gegeven is het **treintraject Schagen - Maastricht**:

Schagen, Heerhugowaard, Alkmaar, Castricum, Zaandam, Amsterdam Sloterdijk, Amsterdam Centraal, Amsterdam Amstel, Utrecht Centraal, 's-Hertogenbosch, Eindhoven, Weert, Roermond, Sittard, Maastricht.

De bedoeling is dat je 3 functies schrijft waarmee je 1) het beginstation aan een reiziger vraagt, 2) het eindstation aan een reiziger vraagt, en 3) de reisgegevens op het scherm print. Jouw functies zouden als volgt gebruikt moeten kunnen worden:

Jouw functies komen hier!!

```
stations = ['Schagen', 'Heerhugowaard', ..., 'Sittard', 'Maastricht']
beginstation = inlezen_beginstation(stations)
eindstation = inlezen_eindstation(stations, beginstation)
omroepen_reis(stations, beginstation, eindstation)
```

Een mogelijke uitvoer van deze code (als de functies zijn geschreven), staat onderaan de volgende pagina. Op de volgende pagina staat ook een stappenplan waarmee je dit programma kunt realiseren.

Stap 1: Neem de gegeven code over, en vul de lijst met alle 15 stationsnamen.

Stap 2: Schrijf functie `def inlezen_beginstation(stations):`

De **returnwaarde** van de functie is een stationsnaam uit de meegegeven lijst!

De functie vraagt een reiziger om het beginstation van de reis in te voeren. Er wordt gecontroleerd of de ingevoerde stationsnaam voorkomt in de lijst met stations van het traject Schagen – Maastricht. Zo niet, dan **vraagt het programma steeds opnieuw om invoer van het beginstation, totdat een correcte stationsnaam is ingevoerd**. Deze wordt geretourneerd.

Stap 3: Schrijf functie `def inlezen_eindstation(stations, beginstation):`

De **returnwaarde** van de functie is een stationsnaam uit de lijst!

De functie vraagt om het **eindstation**. Ook in dit geval wordt gecontroleerd of het opgegeven eindstation voorkomt in lijst **stations**. Daarna wordt gecontroleerd of het eindstation een hogere index in de lijst heeft dan het beginstation. Zo niet, dan moet er een foutmelding geprint worden **en moet de gebruiker opnieuw het eindstation invoeren. Dit moet het programma herhalen totdat er een correct station is ingevoerd**. Deze wordt geretourneerd.

Stap 4: Schrijf functie `def omroepen_reis(stations, beginstation, eindstation):`

Deze functie **print** de volgende informatie op het scherm (zie de uitvoer hieronder):

- De naam van het beginstation van de reis, en tevens het rangnummer in de lijst met stationsnamen (Schagen heeft rangnummer 1 en Maastricht heeft rangnummer 15).
- De naam van het eindstation, en zijn rangnummer binnen het traject.

- De afstand van de reis (in aantallen stations).
- De ritprijs. Het kost 5 euro om te reizen van een station naar het volgende station.
- Het omroepbericht voor jouw reis met vermelding van beginstations, tussenliggende stations (als die er zijn) en eindstation.

Een mogelijke uitvoer:

Wat is je beginstation? : Eindhoven

Wat is je eindstation? : Groningen

Deze trein komt niet in Groningen

Wat is je eindstation? : Roermond

Het beginstation Eindhoven is het 11e station in het traject.

Het eindstation Roermond is het 13e station in het traject.

De afstand bedraagt 2 station(s).

De prijs van het kaartje is 10 euro.

Jij stapt in de trein in: Eindhoven

- Weert

Jij stapt uit in: Roermond

Les 9: Catching Exceptions, CSV-files (Perkovic – §4.4 & §7.3)

Voorbereiding

Lees ter voorbereiding de literatuur van deze les uit Perkovic (§4.4 & §7.3)! Helaas staat er in het boek van Perkovic niets over CSV-bestanden. De docent zal hieraan aandacht besteden in de les. Je kunt ook zelf vast de slides bestuderen. Probeer daarna vast (een aantal van) de Perkovic Exercises en/of Practice Exercises van deze les te maken!

Perkovic Exercises (/ Problems)

- 7.16
- 7.19
- 7.21

Practice Exercises

Practice Exercise 9_1 (catching exceptions)

Je reist met een grote groep en hebt een hotel afgehuurd. Dat kost € 4356,-. Dit is kostbaar, daarom wil je weten hoeveel dit per persoon kost. Schrijf een programma dat de gebruiker vraagt om het aantal personen dat mee op reis gaat. Deel het bedrag door het aantal reizigers en print dat uit! Gebruik een try-except; zorg dat bij foute invoer deze **verschillende** foutmeldingen geprint worden:

- | | |
|---------------------|---|
| Aantal = 0 | - Delen door nul kan niet! |
| Aantal = -20 | - Negatieve getallen zijn niet toegestaan! |
| Aantal = 'twintig' | - Gebruik cijfers voor het invoeren van het aantal! |
| Alle andere fouten: | - Onjuiste invoer! |

Practice Exercise 9_2 (CSV-files schrijven)

Met onderstaande code worden de gegevens van iedereen vastgelegd die wil inloggen in een overheidssysteem. Voorletters, achternaam, geboortedatum en e-mailadres worden ingelezen via de console en weggeschreven naar een CSV-bestand, voorafgegaan door de huidige datum en tijd. Van meerdere personen moeten deze gegevens ingelezen kunnen worden en weggeschreven naar het CSV-bestand. Het programma stopt na intikken van de **naam** 'einde'. Maak het programma af:

```
import datetime
import csv

bestand = 'inloggers.csv'

#gebruik hier een herhalingslus:
naam = input("Wat is je achternaam? ")
voorl = input("Wat zijn je voorletters? ")
gbdatum = input("Wat is je geboortedatum? ")
email = input("Wat is je e-mail adres? ")
#wanneer de volgende persoon inlogt is onbekend, dus schrijf meteen naar file
```

Een **mogelijke** uitvoer zou dan worden:

```
Wat is je achternaam:    van Oranje
Wat zijn je voorletters: W.A.
Wat is je geboortedatum: 27-04-1967
Wat is je e-mail adres: willem@nederland.nl
```

```
Wat is je achternaam: Rutte
Wat zijn je voorletters: M.
Wat is je geboortedatum: 12-12-1971
Wat is je e-mail adres: MarkRutte@torentje.me
Wat is je achternaam: einde
```

Het CSV-bestand ziet er hierna zo uit:

```
Tue 03 May 2016 at 12:06;W.A.;van Oranje;27-04-1967;willem@nederland.nl
Tue 03 May 2016 at 13:12;M.;Rutte;12-12-1971;MarkRutte@torentje.me
```

Practice Exercise 9_3 (CSV-files lezen)

In deze opdracht krijg je een CSV-bestand met informatie van een aantal gamers. Per regel staan er drie gegevens van een gamer: de naam, de speeldatum en de behaalde score. Het programma leest alle regels in en bepaalt welke gamer de hoogste score heeft behaald en op welke datum dat was.

Uitvoer:

```
De hoogste score is: 69 op 11-05-2016 behaald door Douwe Bob
```

Deze informatie komt uit het volgende CSV-bestand, neem deze over:

```
Anton;12-05-2016;29
Douwe Bob;13-05-2016;44
Anton;11-05-2016;39
Douwe Bob;12-05-2016;55
Anton;10-05-2016;29
Douwe Bob;11-05-2016;69
```

Practice Exercise 9_4 (CSV-files met headers)

De onderstaande tabel bevat de gegevens van een aantal producten:

Artikelnummer	Artikelcode	Naam	Voorraad	Prijs
121	ABC123	Highlight pen	231	0.56
123	PQR678	Nietmachine	587	9.99
128	ZYX163	Bureaulamp	34	19.95
137	MLK709	Monitorstandaard	66	32.50
271	TRS665	Ipad hoes	155	19.01

Stap 1: Maak (met Python) een CSV-bestand met bovenstaande gegevens (inclusief headers).

Stap 2: Lees vervolgens het bestand met een Python programma weer in en print **1.)** de naam en de prijs van het duurste product **2.)** het artikelnummer en de voorraad van het artikel met de kleinste voorraad en **3.)** het totaal aantal producten in voorraad!

Uitvoer:

```
Het duurste artikel is Monitorstandaard en die kost 32.50 Euro
Er zijn slechts 34 exemplaren in voorraad van het product met nummer 128
In totaal hebben wij 1073 producten in ons magazijn liggen
```


Practice Exercises

Practice Exercise 10_1 (XML-bestanden schrijven)

In Practice Exercise 9_4 heb je een CSV bestand aangemaakt van deze producten:

Artikelnummer	Artikelcode	Naam	Voorraad	Prijs
121	ABC123	Highlight pen	231	0.56
123	PQR678	Nietmachine	587	9.99
128	ZYX163	Bureaulamp	34	19.95
137	MLK709	Monitorstandaard	66	32.50
271	TRS665	Ipad hoes	155	19.01

Maak nu, met een teksteditor en op basis van deze tabel, een XML-bestand dat er zo uitziet:

Uitvoer:

```
<?xml version="1.0" encoding="UTF-8"?>
<artikelen>
  <artikel nummer="121">
    <code>ABC123</code>
    <naam>Highlight pen</naam>
    <voorraad>231</voorraad>
    <prijs>0.56</prijs>
  </artikel>
  <artikel nummer="123">
    <code>PQR678</code>
    <naam>Nietmachine</naam>
    <voorraad>587</voorraad>
    <prijs>9.99</prijs>
  </artikel>
  ...
</artikelen>
```

Installeer nu ook de module 'xmldict'. File > Settings > Zoek op 'interpreter' en klik op 'Project Interpreter'. Klik rechts in het scherm op het plusje. Zoek nu op 'xmldict' en installeer het package.

Schrijf een programma waarmee je het XML-bestand inleest en alle artikelnamen onder elkaar print!

Practice Exercise 10_2 (Namespaces)

In onderstaande programma's wordt **verkeerd** gebruik gemaakt van namespaces. Verbeter de code!

```
b = 7

def verdubbelB():
    b = b + b

verdubbelB()

print(b)
```

```
print(time.strftime("%H:%M:%S"))
```

```
def f(y):
    return 2*y + 1

print(f(3)+g(3))

def g(x):
    return 5 + x + 10
```

Practice Exercise 10_3 (Locals & Globals)

In de onderstaande programma's worden locals en globals gebruikt. Ga na wat de uitvoer is van deze programma's. Het is natuurlijk erg makkelijk om ze gewoon uit te voeren, maar probeer voordat je dat doet te beredeneren wat de uitvoer is en controleer dan pas jouw antwoord!

Uitvoer 8_3A:

- A) 9
- B) 3
- C) 6

```
a = 3

def f(y):
    global a
    a = 9
    return 2*y + a

print(a)
```

Uitvoer 8_3B:

- A) 4 4
- B) 3 4
- C) 4 3
- D) 3 3

```
x = 1
y = 4

def fun():
    x = 2
    global y
    y = 3
    print(y, end = ' ')

fun()
print(y, end = ' ')
```

Uitvoer 8_3C:

- A) 3 15
- B) 3 10
- C) 6 15
- D) 6 10

```
x = 2
y = 5

def fun():
    y = 3
    global x
    x = 1
    print(x*y, end = ' ')
    return x*y

x = fun()
print(x*y, end = ' ')
```

Uitvoer 8_3D:

- A) a: 9 a: 13
- B) a: 0 a: 13
- C) a: 9 a: 0
- D) a: 0 a: 0

```
a = 3

def fun1():
    global a
    print("a:", a, end = ' ')
    b = 7
    a = 0
    return b

def fun2(y):
    a = y + fun1()
    b = 7
    a += 1
    return a

a = 9
fun2(5)
print("a:", a)
```

Uitvoer 8_3E:

- A) 0
- B) 4
- C) 7
- D) 8

```
x = 1
y = 4

def doe1():
    global x
    y = 7
    x = 0
    return y

def doe2():
    x = doe1()
    x += 1
    return x

x = doe1()
print(x)
```

Uitvoer 8_3F:

- A) a: 5 a: 13
- B) a: 10 a: 10
- C) a: 10 a: 13
- D) a: 5 a: 10

```
a = 5

def fun1():
    global a
    b = 2
    a = 4
    return a+b

def fun2(y):
    global a
    a = y + fun1()
    a += 1
    return a

print("a:", a, end = ' ')
a = fun2(3)
print("a:", a)
```

Uitvoer 8_3G:

- A) 0
- B) 1
- C) 4
- D) 6

```
x = 1
y = 3

def doe1():
    global x
    y = 4
    x = 0
    return x+y

def doe2():
    x = doe1()
    x += 2
    return x

doe2()
print(x)
```

Uitvoer 8_3H:

- A) 0
- B) x bestaat niet
- C) 7
- D) 8

```
def doe1():
    y = 7
    x = 0
    return y

def doe2():
    global x
    x = doe1()
    x += 1

doe2()
print(x)
```

Final Assignment: XML-stationslijsten

In deze opdracht lees je gegevens uit een XML-bestand. Dit bestand is op de volgende bladzijde gegeven. Het bestand kun je overnemen voor je eigen programma: en bevat de gegevens van vier stations. Van ieder station is staat vermeld:

- Code
- Type
- Namen (Kort, Middel & Lang)
- Land
- Synoniemen (niet altijd aanwezig)

Lees alle gegevens uit het bestand en plaats deze in een dict! Laat je programma nu achtereenvolgens de onderstaande gegevens afdrukken op het scherm:

1. Van alle stations de code en het type.
2. Van alle stations de code en synoniemen (maar alleen als er synoniemen zijn).
3. Van alle stations de code en de lange naam.

Let op: bij stap 2 mag je de dict met alle synoniemen uitprinten, maar dat kan natuurlijk netter. Bij stap 3 is het de bedoeling dat je wel netjes de lange naam van een station uit de namen-dict haalt!

Mogelijke uitvoer:

```
Dit zijn de codes en types van de 4 stations:
HT - knooppuntIntercitystation
ALMO - stoptreinstation
ATN - stoptreinstation
ASA - intercitystation

Dit zijn alle stations met één of meer synoniemen:
HT - OrderedDict([('Synoniem', ["Hertogenbosch ('s)", 'Den Bosch']]))

Dit is de lange naam van elk station:
HT - 's-Hertogenbosch
ALMO - Almere Oostvaarders
ATN - Aalten
ASA - Amsterdam Amstel
```

Het XML-bestand hieronder kun je overnemen voor je eigen programma:

```
<?xml version="1.0" encoding="UTF-8"?>

<Stations>

  <Station>
    <Code>HT</Code>
    <Type>knooppuntIntercitystation</Type>
    <Namen>
      <Kort>Den Bosch</Kort>
      <Middel>'s-Hertogenbosch</Middel>
      <Lang>'s-Hertogenbosch</Lang>
    </Namen>
    <Land>NL</Land>
    <Synoniemen>
      <Synoniem>Hertogenbosch ('s)</Synoniem>
      <Synoniem>Den Bosch</Synoniem>
    </Synoniemen>
  </Station>

  <Station>
    <Code>ALMO</Code>
    <Type>stoptreinstation</Type>
    <Namen>
      <Kort>Oostvaard</Kort>
      <Middel>Oostvaarders</Middel>
      <Lang>Almere Oostvaarders</Lang>
    </Namen>
    <Land>NL</Land>
    <Synoniemen></Synoniemen>
  </Station>

  <Station>
    <Code>ATN</Code>
    <Type>stoptreinstation</Type>
    <Namen>
      <Kort>Aalten</Kort>
      <Middel>Aalten</Middel>
      <Lang>Aalten</Lang>
    </Namen>
    <Land>NL</Land>
    <Synoniemen></Synoniemen>
  </Station>

  <Station>
    <Code>ASA</Code>
    <Type>intercitystation</Type>
    <Namen>
      <Kort>Amstel</Kort>
      <Middel>Amsterdam Amstel</Middel>
      <Lang>Amsterdam Amstel</Lang>
    </Namen>
    <Land>NL</Land>
    <Synoniemen></Synoniemen>
  </Station>

</Stations>
```

Les 11: Voorbereiding Miniproject: NS-API

Voorbereiding

Les 11 kent geen Final Assignment omdat de stof van deze les niet getoetst zal worden op het tentamen. Dat wil niet zeggen dat er niets te doen is... Voor het miniproject is het de bedoeling dat je verbinding maakt met een **Application Programming Interface (API)**. Een API kan je zien als een soort service die een bedrijf, instantie of persoon aanbiedt, en waar je informatie 'op maat' van kunt opvragen. Je kunt denken aan een **Weer-API**, waar je op basis van bijvoorbeeld gps-coördinaten de weersverwachting van een bepaald gebied kunt opvragen, of de weerhistorie van de afgelopen 10 jaar van de plaats waar je woont.

Misschien denk je: "wat heb je daar nu weer aan, dat kan ik toch ook op de website van Buienradar vinden?". Dat is natuurlijk zeker waar, maar de informatie van een API wordt meestal als losstaande informatie aangeboden, die je kunt verwerken in je eigen applicatie zodat je alleen die informatie verwerkt waar jouw gebruikers behoefte aan hebben.

Deze les bestaat uit een tutorial om een programma te schrijven waarmee je gegevens ophaalt van de NS-API. Eén van de miniprojecten is ook gebaseerd op deze API. Er bestaan echter nog meer APIs, dus als je meer uitdaging wilt, dan kan je dit als voorbeeld gebruiken en tijdens het miniproject een nieuwe API uitproberen.

Let op: de antwoorden van de NS-API zijn verpakt in XML, wat tijdens de lessen ook aan de orde is geweest. Niet alle APIs leveren XML op, sommigen gebruiken bijvoorbeeld JSON, wat weer een andere notatiewijze is. Zoek dit eerst uit voordat je een keuze maakt voor een miniproject!

Het is zeer aan te raden om de tutorials van les 11 en 12 te doorlopen **voordat** je aan het miniproject begint. Doe je dat niet, dan heb je direct een achterstand (op de andere teamleden). Je hoeft jouw uitwerking echter niet verplicht in te leveren, dus je kunt de tutorials ook tijdens de onderwijsvrije week doorlopen!

Uit te voeren ter voorbereiding van deze tutorial:

1. Vraag een NS-API key aan op <http://www.ns.nl/reisinformatie/ns-api>.


Tutorial: De NS-API

Deze tutorial bespreekt drie stappen die je kunt doorlopen om vanuit Python een API aan te roepen. Neem deze stappen door, en probeer de code-voorbeelden ook zelf uit te voeren en te begrijpen!

Stap 1: API-informatie

Op de website van waar je een account hebt aangevraagd staat een beschrijving van de NS-API. Hier kan je uit opmaken dat de API informatie biedt met betrekking tot prijzen, vertrektijden, storingen etc. We gaan in deze tutorial de actuele vertrektijden opvragen rondom Utrecht.

Als je informatie wilt hebben, dan moet je een **request** (verzoek) indienen. Klik op de NS-API-site op 'Documentatie actuele vertrektijden', en scroll naar het kopje **Request**. Je ziet daar dit:


Menu
Zoeken
Mijn NS
English

Request

Actuele vertrekinformatie kan worden opgehaald met behulp van de volgende url:

`https://webservices.ns.nl/ns-api-avt?station=${Naam of afkorting Station}`

Parameter	Omschrijving	Verplicht
station	De code (afkorting) of korte naam of middellange naam of volledige naam of synoniem van de stationsnaam	Ja

Er zullen minimaal 10 vertrektijden worden geretourneerd en minimaal de vertrektijden voor het komende uur. Een voorbeeld request voor de actuele vertrektijden van station Utrecht Centraal:

<https://webservices.ns.nl/ns-api-avt?station=ut>

Omdat vanaf Utrecht Centraal meer dan 10 treinen per uur vertrekken, bevat de response alle vertrekinformatie voor het komende uur.

Een voorbeeld request voor de actuele vertrektijden van station Kampen:

<https://webservices.ns.nl/ns-api-avt?station=Kampen>

Omdat vanaf Kampen 2 treinen per uur vertrekken, worden er 10 vertrektijden geretourneerd.

Hier staat dus de **beschrijving van de API**. We kunnen bijvoorbeeld afleiden dat je één parameters mee moet geven aan deze API-service! Je ziet ook al twee voorbeeld request-URLs (blauwgedrukt) staan. We bekijken de eerste ervan wat beter:

<https://webservices.ns.nl/ns-api-avt?station=ut>

Deze voorbeeld-URL bestaat eigenlijk uit drie delen:

<http://webservices.ns.nl/ns-api-avt>

Basis-URL voor deze service

[?](#)

Einde adres, beginteken voor parameters

[station=ut](#)

parameter **station**, met als waarde 'ut'

Als je deze API aanroept, krijg je een reactie (**response**). Een mogelijke XML-response is:

```
<?xml version="1.0" encoding="UTF-8"?>
<ActueleVertrekTijden>
  <VertrekkendeTrein>
    <RitNummer>3156</RitNummer>
    <VertrekTijd>2016-09-27T16:58:00+0200</VertrekTijd>
    <EindBestemming>Schiphol Airport</EindBestemming>
    <TreinSoort>Intercity</TreinSoort>
    <RouteTekst>Bijlmer ArenA, A'dam Zuid</RouteTekst>
    <Vervoerder>NS</Vervoerder>
    <VertrekSpoor wijziging="true">7</VertrekSpoor>
  </VertrekkendeTrein>
  <VertrekkendeTrein>
    <RitNummer>11756</RitNummer>
    <VertrekTijd>2016-09-27T16:59:00+0200</VertrekTijd>
    <EindBestemming>Den Haag Centraal</EindBestemming>
    <TreinSoort>Intercity</TreinSoort>
    <RouteTekst>Gouda</RouteTekst>
    <Vervoerder>NS</Vervoerder>
    <VertrekSpoor wijziging="true">8a</VertrekSpoor>
  </VertrekkendeTrein>
</ActueleVertrekTijden>
```

Je kunt in een browser de URL invoeren en uitproberen. Probeer nu zelf ook eens de vertrektijden van een ander station, bijvoorbeeld waar je elke dag langs reist, op te vragen. Kijk ook eens wat de service doet als je de verplichte parameter weglaat!

Stap 2: Python en APIs

Je kunt de API-resultaten nu wel in een browser weergeven, maar het idee van een API is juist dat je de uitvoer ook kunt gebruiken in een Python programma. We schrijven daarom een Python-programma dat voor ons een request uitvoert en het resultaat op het scherm print. We doorlopen daarvoor de volgende stappen:

1. Module **requests** importeren
2. API-URL en loggegevens (authorisatie) in het programma opnemen.
3. Request uitvoeren
4. XML-resultaten printen.

Voordat je aan stap 1 kunt beginnen moet je in PyCharm de module **requests** installeren. Kies in PyCharm menu File > Settings > Zoek op 'interpreter' en klik op 'Project Interpreter'. Klik rechts in het scherm op het plusje. Zoek nu op '**requests**' en installeer het package.

Hieronder is een voorbeeld opgenomen waarin de vier beschreven stappen worden uitgevoerd:

```
import requests

auth_details = ('Jouw API-gebruikersnaam', 'Jouw API-wachtwoord')
api_url = 'http://webservices.ns.nl/ns-api-avt?station=ut'

response = requests.get(api_url, auth=auth_details)

print(response.text)
```

De code print het volledige XML-bericht uit. Maar misschien wil je de informatie niet printen, maar juist in een bestand opslaan. In dat geval moet je de print opdracht vervangen voor deze code:

```
with open('vertrektijden.xml', 'w') as myXMLFile:
    myXMLFile.write(response.text)
```

Stap 3: Python en XML-APIs

We hebben nu weliswaar de XML-resultaten, maar vaak heb je niet alle gegevens nodig, en wil je dus een selectie maken. In les 8 heb je dat al eens gedaan. Daarbij werd steeds een bestand ingelezen als string. Hier hoeft je geen bestand in te lezen, maar vraag je de XML-string op via de API. Vervolgens kan je weer de module `xmltodict` gebruiken om de gegevens verder te verwerken.

Als voorbeeld gaan we de **vertrektijd** en **eindbestemming** van elke vertrekkende trein uitprinten:

```
import requests
import xmltodict

auth_details = ('Jouw API-gebruikersnaam', 'Jouw API-wachtwoord')
api_url = 'http://webservices.ns.nl/ns-api-avt?station=ut'
response = requests.get(api_url, auth=auth_details)

vertrekXML = xmltodict.parse(response.text)

print('Dit zijn de vertrekkende treinen:')
for vertrek in vertrekXML['ActueleVertrekTijden']['VertrekkendeTrein']:
    eindbestemming = vertrek['EindBestemming']

    vertrektijd = vertrek['VertrekTijd']          # 2016-09-27T18:36:00+0200
    vertrektijd = vertrektijd[11:16]             # 18:36

    print('Om ' + vertrektijd + ' vertrekt een trein naar ' + eindbestemming)
```

Uiteraard kan je dit nog eindeloos uitbreiden, door bijvoorbeeld van de parameter **'station'** de waarde **'ut'** te vervangen door een stad die de gebruiker zelf heeft ingevuld. Ook kan je niet alleen de tijd, maar bijvoorbeeld ook het spoornummer of het type trein tonen.

Het is hier niet gedaan, maar het spreekt waarschijnlijk wel voor zichzelf dat wanneer de code die je hebt geschreven steeds groter zal worden, ook het opdelen in functies noodzakelijk is!

Stap 4: Aan de slag (optioneel)

Ga zelf op zoek naar een API die je interessant lijkt, en probeer deze aan te roepen!

Les 12: Voorbereiding Miniproject: Tkinter

Voorbereiding

Les 12 kent geen Final Assignment omdat de stof van deze les niet getoetst zal worden op het tentamen. Voor het miniproject is het echter wel de bedoeling dat je de gebruiker een Grafische User Interface (GUI) kunt aanbieden. Deze les bestaat onder andere uit een tutorial waarbij je basiskennis opdoet met betrekking tot het maken van een eenvoudige GUI in Python. Het is handig om daarvoor Perkovic, paragraaf 9.1 en 9.2 door te nemen, maar dat is niet noodzakelijk!

Tutorial: Tkinter

Bestudeer de verschillende stappen hieronder, en probeer de code uit op je eigen laptop. Verander ook zo nu en dan de code om te kijken wat het effect is en of je bijvoorbeeld het uiterlijk van je programma kunt beïnvloeden.

Stap 1: Een GUI openen

Om in Python een GUI te maken, gebruiken we module **tkinter**. Deze is standaard in Python aanwezig:

```
from tkinter import *

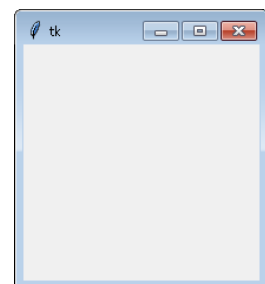
root = Tk()                                # Creëer het hoofdscherm
root.mainloop()                           # Toon het hoofdscherm
```

De eerste regel valt op, want hier importeren we op een andere manier dan gebruikelijk. Je zegt daarmee als het ware: importeer alles (*) van module tkinter. Het voordeel is dan dat je niet steeds voor alles wat je wilt gebruiken de modulenaam hoeft te zetten. Zie het verschil hieronder:

<pre>from tkinter import * root = Tk() root.mainloop()</pre>	<pre>import tkinter root = <u>tkinter</u>.Tk() root.mainloop()</pre>
---	---

Het uitvoeren van deze code levert overigens een leeg scherm op. Dit is de basis waarop je allerlei GUI-onderdelen kunt gaan plaatsen. Een onderdeel kan bijvoorbeeld een Label of Button zijn. In tkinter heten die onderdelen **widgets**.

We gaan nu achtereenvolgens de widgets Label, Button en Entry toevoegen aan het hoofdscherm! Probeer deze voorbeelden ook steeds uit op je eigen laptop!



Stap 2: Widgets toevoegen

We beginnen met het toevoegen van een Label:

```
from tkinter import *
root = Tk()

label = Label(master=root, text='Hello World', background='yellow')
label.pack()

root.mainloop()
```



Aan een Label geven we een aantal parameters mee:

<code>master=root</code>	= plaatsen op hoofdscherm (root)
<code>text='Hello World'</code>	= de tekst die getoond moet worden
<code>background='yellow'</code>	= de achtergrondkleur van het label

Methode **pack()** plaatst het label zodanig in de hoofdvenster (**root**), dat het label het window volledig opvult. Daarbij wordt het item door deze methode automatisch tegen de bovenkant (TOP) van het scherm geplakt (tenzij je anders aangeeft, dat zien we zo direct).


Er zijn nog veel meer parameters voor een Label mogelijk, zie bijvoorbeeld Perkovic, Table 9.1 (blz 293). De code hieronder toont bijvoorbeeld hoe je het lettertype, de letterkleur en de hoogte + breedte (in aantal tekst-units) van een Label kunt aanpassen:

```
from tkinter import *
root = Tk()

label = Label(master=root,
               text='Hello World',
               background='yellow',
               foreground='blue',
               font=('Helvetica', 16, 'bold italic'),
               width=14,
               height=3)

label.pack()

root.mainloop()
```




Omdat hier veel parameters worden ingesteld, zijn ze voor de leesbaarheid onder elkaar geplaatst! De tekst van een Label wordt automatisch in het midden uitgelijnd! We voegen nu ook een Button toe aan het hoofdscherm (`master=root`). Dat gaat op dezelfde wijze:

```
from tkinter import *
root = Tk()

label = Label(master=root, text='Hello World', height=2)
label.pack()

button = Button(master=root, text='Druk hier')
button.pack(pady=10)

root.mainloop()
```



De button **doet** nu nog helemaal niets als je erop klikt, daar kijken we pas naar bij stap 4. Merk verder op dat er enige ruimte is tussen de button en de rand van het window! Dat is gedaan door aan de functie pack een **padding** mee te geven (**pady**, dus op de y-as) van 10 pixels. Dit zorgt voor een lege rand van 10 pixels hoog, onder en boven de button.

- ☛ Je kan er ook nog voor kiezen om de button over de gehele breedte van het venster uit te rekken door de parameter **fill=X** mee te geven aan de pack functie. Probeer dit eens uit en bekijk het resultaat! Combineer dit met de parameter **padx**. Wat is het effect daarvan?

De volgende stap is het toevoegen van een invoerveld. We gebruiken wederom dezelfde werkwijze:

```
from tkinter import *
root = Tk()

label = Label(master=root, text='Hello World', height=2)
label.pack()

button = Button(master=root, text='Druk hier')
button.pack(pady=10)

entry = Entry(master=root)
entry.pack(padx=10, pady=10)

root.mainloop()
```



Naast deze belangrijke widgets zijn er nog 14 andere widgets die standaard in tkinter voorkomen. Deze zullen we niet allemaal bekijken, maar je kunt het overzicht vinden in de [documentatie](#) van tkinter. Je moet er wel rekening mee houden dat je niet bij elke widget dezelfde parameters kunt gebruiken! Lees daarom de documentatie of gebruik de **help()** functie om informatie over een widget op te vragen.

Er zijn echter nog een paar problemen: er gebeurt nog helemaal niets als we de knop indrukken, en wat als we veel meer widgets op verschillende posities aan het scherm willen toevoegen? We kijken daarom in stap 3 eerst hoe je ervoor kunt zorgen dat een knop ook werkt, en in stap 4 nemen we de layoutmanagers onder de loep.

Stap 3: Event-based widgets

We gebruiken het laatste programma als uitgangspunt, en we gaan ervoor zorgen dat als je op de knop klikt, de tekst in het invoerveld in het Label geplaatst wordt. We doen dat als volgt:

1. We schrijven een functie **clicked()** die deze acties uitvoert.
2. We koppelen deze functie als **event handler** aan de knop.

Deze code is redelijk eenvoudig:

```
from tkinter import *

def clicked():
    label["text"] = entry.get()

root = Tk()

label = Label(master=root, text='Hello World', height=2)
label.pack()

button = Button(master=root, text='Druk hier', command=clicked)
button.pack(pady=10)

entry = Entry(master=root)
entry.pack(padx=10, pady=10)

root.mainloop()
```



Doordat aan de Button nu de parameter **command=clicked** is toegevoegd, zal bij het klikken op de knop de functie **clicked** worden aangeroepen. Het enige dat de functie doet, is het veranderen van de eigenschap **text** van het Label. De nieuwe waarde wordt datgene dat de user heeft ingevoerd in de **Entry**. Deze invoer vraagt de functie op met **entry.get()**.

Let op: de functie **clicked** MOET een functie **ZONDER parameters** zijn! Er zijn ook andere soorten event handler-functies waarbij wel een parameter meegegeven moet worden, maar die behandelen we niet in deze tutorial. Zie daarvoor Perkovic, blz 306-308!

Let op: de functie **clicked** wordt pas aangeroepen als iemand op de knop klikt! Daarom staat er ook **command=clicked** en geen **command=clicked()**. Op deze manier geef je de functie **clicked** mee aan de Button als parameter, zonder de functie al uit te voeren!

Vanzelfsprekend kan je ook andere code laten uitvoeren. In de onderstaande code zal bijvoorbeeld de ingevoerde waarde in de Entry gekwadeerd worden. De uitkomst wordt weer in het Label geplaatst:

```
from tkinter import *

def clicked():
    grondtal = int(entry.get())
    kwadraat = grondtal ** 2
    tekst = "kwadraat: van {} = {}".format(grondtal, kwadraat)
    label["text"] = tekst

root = Tk()

label = Label(master=root, text='Hello World', height=2)
label.pack()

button = Button(master=root, text='Druk hier', command=clicked)
button.pack(pady=10)

entry = Entry(master=root)
entry.pack(padx=10, pady=10)

root.mainloop()
```



Deze korte stukjes code geven een aardig beeld van de mogelijkheden en hiermee ben je in staat om een eenvoudige GUI op te bouwen en aan te passen.

- ☛ Probeer eens wat het effect is als je toch **command=clicked()** (dus met haakjes) gebruikt!
- ☛ Probeer het programma aan te passen zodat er twee buttons zijn. De ene button kwadeert het getal in de Entry, en de andere button berekent de wortel van het ingevoerde getal. Schrijf voor de tweede button een aparte functie!

Stap 4_1: Layoutmanagers: pack()

We hebben nu steeds gebruik gemaakt van de functie **pack()**. Deze functie heeft echter als nadeel dat je beperkte mogelijkheden hebt. Zo kan je wel de voorkeurspositie van een widget opgeven, maar bepaalt de layoutmanager van tkinter waar dat onderdeel precies komt te staan. We zullen eerst eens kijken hoe dat werkt, en daarna zullen we ook twee alternatieven bespreken. Je kunt dan zelf bepalen welke je wilt gebruiken.

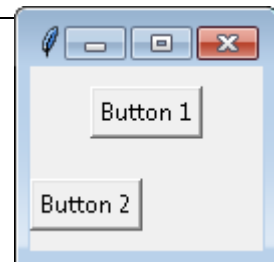
Stel je wilt twee buttons, dan zullen die met de functie **pack()** onder elkaar geplaatst worden. De functie probeert namelijk normaal gesproken de widget bovenaan het venster te plaatsen. Als daar al een widget staat, komt de volgende eronder. Dit gedrag kan je beïnvloeden met de parameter **side**:

```
from tkinter import *
root = Tk()

button1 = Button(master=root, text='Button 1')
button1.pack(pady=10)

button2 = Button(master=root, text='Button 2')
button2.pack(side=LEFT, pady=10)

root.mainloop()
```



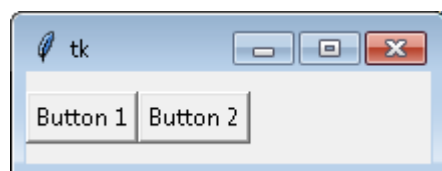
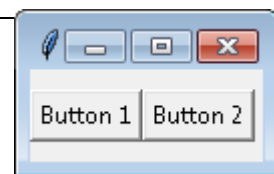
Er zijn 4 verschillende waarden mogelijk voor parameter **side**: TOP (default), LEFT, RIGHT en BOTTOM. Stel dat we button1 ook aan de linkerkant zouden plaatsen, dan krijg je een, misschien, onverwacht effect. Het bovenste plaatje is wat tkinter ervan maakt:

```
from tkinter import *
root = Tk()

button1 = Button(master=root, text='Button 1')
button1.pack(side=LEFT, pady=10)

button2 = Button(master=root, text='Button 2')
button2.pack(side=LEFT, pady=10)

root.mainloop()
```



Omdat button1 als eerste geplaatst is, staat deze helemaal links tegen de vensterrand. Button2 staat in feite ook links, maar dan naast het item dat er al stond. Dit is pas goed te zien als je het venster breder maakt, wat in het onderste plaatje is weergegeven. We bekijken nog een voorbeeld:

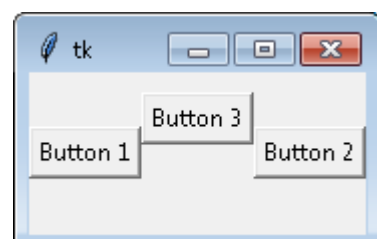
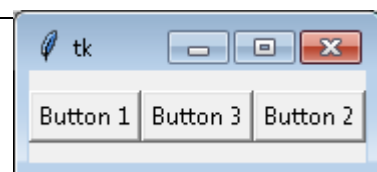
```
from tkinter import *
root = Tk()

button1 = Button(master=root, text='Button 1')
button1.pack(side=LEFT, pady=10)

button2 = Button(master=root, text='Button 2')
button2.pack(side=RIGHT, pady=10)

button3 = Button(master=root, text='Button 3')
button3.pack(side=TOP, pady=10)

root.mainloop()
```



Omdat er met de functie `pack()` slechts 4 verschillende posities mogelijk zijn, kan het soms wat lastig kan worden als je heel veel widgets wilt plaatsen. Stel dat je bijvoorbeeld twee buttons aan de linkerkant van het scherm wilt plaatsen, maar je wilt ze onder elkaar hebben. Dan kan je niet zeggen dat beide buttons links moeten komen te staan, want dan zullen ze automatisch naast elkaar geplaatst worden. Een oplossing kan dan zijn om een **Frame** te gebruiken!

Een Frame is als het ware een **container** waarin je weer andere widgets kunt plaatsen. Die container kan je dan als één element op het hoofdscherm plaatsen. Je kunt dan aangeven of de container LEFT, RIGHT, TOP of BOTTOM in het scherm moet komen. Binnen de container kan je ook weer een onderverdeling in LEFT, RIGHT, TOP of BOTTOM maken.

In onderstaande code is bijvoorbeeld een linkerframe gemaakt, waarin twee buttons zijn opgenomen. De buttons zijn zonder **side** parameter, en dat betekent dat ze bovenin de container geplaatst worden. Merk op dat de **master** van `button1` en `button2` dus **linkerframe** is. Het frame heeft zelf weer als master **root**, en wordt daarin aan de linkerkant geplaatst. Tot slot is er nog een `button3`, die rechts in het hoofdscherm is opgenomen:

```
from tkinter import *
root = Tk()

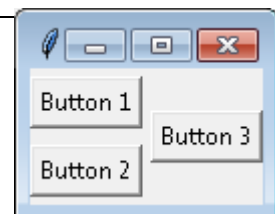
linkerframe = Frame(master=root)
linkerframe.pack(side=LEFT)

button1 = Button(master=linkerframe, text='Button 1')
button1.pack(pady=4)

button2 = Button(master=linkerframe, text='Button 2')
button2.pack(pady=4)

button3 = Button(master=root, text='Button 3')
button3.pack(side=RIGHT, pady=4)

root.mainloop()
```



Met een beetje nadenken kan je met de functie `pack()` een heel eind komen, en hier heb je dan ook waarschijnlijk genoeg aan voor een relatief eenvoudige GUI. Mocht dat niet het geval zijn, dan zijn er nog enkele alternatieven, die we kort in stap 4_2 en 4_3 bespreken.

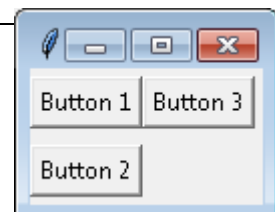
Stap 4_2: Layoutmanagers: `grid()`

Als je niet voldoende hebt aan `pack()`, dan biedt tkinter ook nog de **`grid()`** functie. Hiermee kan je het hele scherm in kolommen en rijen verdelen, waardoor een soort tabel met cellen ontstaat. Voor elke widget geef je dan aan in welke cel deze moet komen te staan. Een voorbeeld:

```
from tkinter import *
root = Tk()

button1 = Button(master=root, text='Button 1')
button1.grid(row=0, column=0, pady=4)

button2 = Button(master=root, text='Button 2')
button2.grid(row=1, column=0, pady=4)
```



```
button3 = Button(master=root, text='Button 3')
button3.grid(row=0, column=1, pady=4)

root.mainloop()
```

Door het gebruik van de functie **grid()** rekent tkinter zelf uit hoeveel rijen en kolommen er zijn. Dat hoeft je verder nergens op te geven. We bespreken deze functie hier verder niet, maar Perkovic behandelt de **grid()** functie in §9.2! Het is belangrijk om te beseffen dat je **pack()** OF **grid()** moet gebruiken, niet beiden!!

Stap 4_3: Layoutmanagers: place()

Als je dit allemaal maar ingewikkeld vindt, dan kan je eventueel nog terugvallen op de **place()** functie. Deze komt niet aan de orde in Perkovic, en is ook niet erg efficiënt, maar kan in sommige gevallen uitkomst bieden. Hierbij kan je de exacte coördinaten (x,y) van een widget opgeven. Hierbij loopt de x-as van linksboven naar rechtsboven, en de y-as van linksboven naar linksonder:

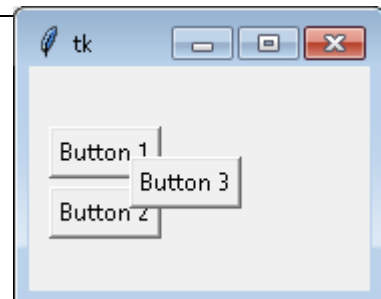
```
from tkinter import *
root = Tk()

button1 = Button(master=root, text='Button 1')
button1.place(x=10, y=30)

button2 = Button(master=root, text='Button 2')
button2.place(x=10, y=60)

button3 = Button(master=root, text='Button 3')
button3.place(x=50, y=45)

root.mainloop()
```



Zoals je kunt zien, kan je hierbij dus widgets (per ongeluk) over elkaar heen plaatsen. Daarnaast is deze oplossing niet erg schaalbaar, want als het scherm breder of hoger zou worden, blijven de widgets exact op dezelfde positie staan. Hier is binnen de functie **place()** wel iets aan te doen, maar ook dat valt buiten de scope van deze tutorial. Deze methode is echter wel toegestaan bij het miniproject!

Stap 5: Pop-up vensters (optioneel)

Mocht je de gebruiker een melding willen geven, dan kan je daarvoor het beste een Label gebruiken. In sommige gevallen is het echter mogelijk dat dit niet genoeg opvalt. Je kunt dan eventueel een popup-venster gebruiken:

```
from tkinter import *
from tkinter.messagebox import showinfo

def clicked():
    bericht = 'Dit een bericht voor de gebruiker!'
    showinfo(title='popup', message=bericht)

root = Tk()
button = Button(master=root, text='Druk hier', command=clicked)
button.pack(pady=10)

root.mainloop()
```



Stap 6: Meerdere Frames in een venster (optioneel)

Vaak wil je in een venster allerlei soorten van interactie met de gebruiker hebben. Maar niet alle knoppen en labels moeten altijd zichtbaar zijn. Zo is het soms noodzakelijk om een gebruiker eerst te laten inloggen voordat hij/zij bijvoorbeeld kan een mail kan versturen. Het is dan nodig om eerst de invoervelden en knoppen te tonen, en pas daarna een ander scherm. Dat betekent dus dat je knoppen en labels moet tonen, en later weer verwijderen.

Gelukkig kan dat eenvoudig met behulp van frames. Een frame is een soort container waarop je bijvoorbeeld een knop en invoerveld voor het inloggen zet. Als iemand is ingelogd, laat je in één keer het frame verdwijnen, en plaats je daarvoor in de plaats een ander frame. Zie hieronder een voorbeeld:

```
from tkinter import *

def toonLoginFrame():
    hoofdfree.pack_forget()
    loginframe.pack()

def toonHoofdFrame():
    loginframe.pack_forget()
    hoofdfree.pack()

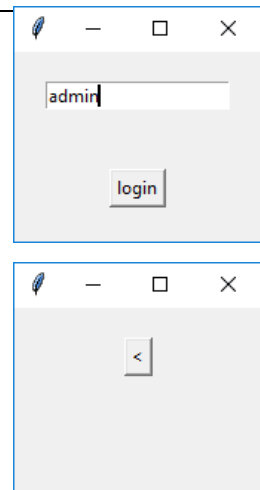
def login():
    if loginfield.get() == "admin":
        toonHoofdFrame()
    else:
        print('Verkeerde gebruikersnaam!')

root = Tk()

loginframe = Frame(master=root)
loginframe.pack(fill="both", expand=True)
loginfield = Entry(master=loginframe)
loginfield.pack(padx=20, pady=20)
loginbutton = Button(master=loginframe, text='login', command=login)
loginbutton.pack(padx=20, pady=20)

hoofdfree = Frame(master=root)
hoofdfree.pack(fill="both", expand=True)
backbutton = Button(master=hoofdfree, text='<', command=toonLoginFrame)
backbutton.pack(padx=20, pady=20)

toonLoginFrame()
root.mainloop()
```



Allereerst zal natuurlijk het script worden uitgevoerd, dat begint onder de functies. Zowel het loginframe als het hoofdframe moet eerst gecreëerd worden. In deze code zit iets nieuws:

```
loginframe.pack(fill="both", expand=True)
```

Hiermee zal het betreffende frame over de volledige breedte en hoogte van de beschikbare ruimte worden uitgespreid. Na het maken van de frames wordt de functie toonLoginFrame() aangeroepen. Deze functie gebruikt de volgende regels om dat voor elkaar te krijgen:

```
hoofdfree.pack_forget()    # verberg het hoofdframe
loginframe.pack()          # toon het loginframe (opnieuw)
```

Zodra de gebruiker op de loginknop klikt, zal functie login aangeroepen worden. Zie stap 3 als je niet meer precies weet hoe je dat ook alweer moest doen! De functie controleert of de gebruiker 'admin' heeft ingevoerd. Zo ja, dan zal het hoofdframe getoond worden, op dezelfde wijze als waarop het loginframe werd getoond. In het hoofdframe is een knop om terug naar het loginframe te gaan.

Deze manier van coderen zorgt er wel voor dat er veel code in een bestand te staan, dus het is verstandig om niet-GUI-gerelateerde code in aparte scripts te schrijven. Zie daarvoor de volgende stap.

Stap 7: Python scripts aanroepen

Het is verstandig om scripts waarin je een GUI opbouwt, niet te mengen met de scripts waarin je berekeningen en andere logica hebt geprogrammeerd. In Les 3 heb je bijvoorbeeld een functie moeten programmeren waarin de standaardprijs van een treinrit werd uitgerekend. Stel dat je die hebt opgeslagen in het bestand 'Treinfuncties.py', dat er als volgt uit ziet:

```
def standaardprijs(afstandKM):
    if afstandKM < 0:
        afstandKM = 0

    if afstandKM < 50:
        prijs = afstandKM * 0.80
    else:
        prijs = 15 + afstandKM * 0.60

    return prijs
```

We gaan ervan uit dat dit bestand in dezelfde directory staat als jouw GUI-script, en dat beide bestanden in de hoofddirectory van het project staan. Dan kan je deze functies importeren, en gebruiken in het GUI-script:

```
from tkinter import *
from Treinfuncties import *

def berekenTarief():
    afstand = afstandEntry.get()
    prijs = int(standaardprijs(afstand))
    label["text"] = "De ritprijs is: {}".format(prijs)

root = Tk()

afstandEntry = Entry(master=root)
afstandEntry.pack()

button = Button(master=root, text="Druk hier", command=berekenTarief)
button.pack()

label = Label(master=root)
label.pack()

root.mainloop()
```

Let op: als de bestanden in een subdirectory van je project staan, dan moet je op de tweede regel van het bovenstaande script dus het volledige path naar het betreffende Python-script opnemen, zodat Python dat script kan vinden als je jouw programma uitvoert.

Stap 8: Aan de slag (optioneel)

Maak een GUI waarin je de gebruiker een stationsnaam laat invoeren (Entry). Nadat de gebruiker op OK klikt (Button), worden de eerstvolgende vijf vertrektijden vanaf dat station getoond (5x Label). Neem hiervoor de code die je in les 9, tutorial 1 hebt gemaakt, als uitgangspunt. Als je deze opdracht succesvol weet af te ronden, ben je meer dan voorbereid op het miniproject! Het is ook niet erg als dat niet helemaal lukt, want tijdens het miniproject kan je jouw teamleden om hulp vragen!