|  |  |
| --- | --- |
| Logo_intec-01_300dpi.jpg | **Datum**: 15/11/2019  **Opleiding**: Java IoT  **Lesmodule**: Python  **Test**: Praktijk |
| **Resultaat**: **/ 15** | **Naam**: Patryk Sitko |

Het is de bedoeling dat je een project aanmaakt met hierin minimum 1 directory, hierin plaats je de verschillende praktijkopdrachten (Je mag de filenames de titel geven van de opdracht bv: opdracht-1.py; opdracht-2.py, ...).

Probeer zoveel mogelijk goed practices te tonen.

Indien je extra functionaliteit wil toevoegen [denk aan een functie die helpt bij je opdracht dan mag dit, maar doe dit pas wanneer alle opdrachten klaar zijn].

Ik verwacht dat je een online repo aanmaakt [github] en hierin je code plaatst, zorg ervoor dat ik aan je code kan. Stuur een bericht via teams wanneer je klaar bent met een link naar je online repo.

Opdracht 1. Calling in sick

In deze oefening maak je gebruik van een paar variabelen die **willekeurig** worden ingesteld op Booleaanse waarden (True, False).

* **actually\_sick**
* **kinda\_sick**
* **dont\_feel\_like\_it**

Je maakt ook de variable **sick\_days** aan die een random waarde tussen 0 en 10 heeft.

Ten slotte is er een variabele met de naam **calling\_in\_sick** die je moet instellen op True of False op basis van de volgende scenario's

Stel in op True als:

\* Je bent eigenlijk ziek en je hebt nog ziektedagen.

\* Je bent een beetje ziek, je hebt geen zin om te gaan werken en je hebt nog ziektedagen.

Maak gebruik van een control flow statement om indien nodig sick days af te trekken van je resterende aantal dagen [bijgehouden in **sick\_days**].

Maak ook gebruik van while loop om meerdere malen te itereren , geef telkens nieuwe random booleaanse waarde aan bovenstaande variabelen.

Op het einde van je loop print je elke keer de variabele **calling\_in\_sick** af in een formatted string (maak deze meaningful). De loop eindigt wanneer je geen sick days meer overhoud.

Opdracht 2. Creëren van een list

Maak volgende lijsten aan:

* **my\_data**:

Moet minstens 4 elementen lang zijn. Je mag zelf data verzinnen. Maar het moet minstens 1 string en 1 float bevatten.

* **numbers**

Dit is een lijst met alle even nummers tussen 1 en 1000(inclusief 1000). Je mag dit niet hard coded aanmaken [dus niet elke waarde zelf meegeven]. Maak gebruik van wat je geleerd hebt [list comprehension, range, ...].

Toon een aantal verschillende manieren (minstens 2) om een list (mag met of zonder data, probeer een andere manier dan hierboven te gebruiken) aan te maken.

Opdracht 3a. List comprehension oefening

Je mag onderstaande stukje code overnemen:

group\_of\_people = [‘Alex’, ‘Eliot’, ’Veronica’, ‘Lucy’, ‘Wouter’, ‘Bart’]

De bedoeling is dat je een list maakt genaamd first\_chars hierin zal je telkens de eerste char van elke string plaatsen. Dit gebeurd aan de hand van list comprehension.

Ook het volgende stukje code mag je overnemen:

numbers = list(range(100))

Maak de variabele sum aan en plaats hierin het resultaat van alle getallen in de list numbers.

Print deze waarde af naar de console

Opdracht 3b. List comprehension oefening

Geef mij een correct voorbeeld van list comprehension [je bepaald zelf het algoritme]. Probeer hiermee het nut aan te tonen van waarom je list comprehension zou gebruiken ten opzichte van de standaard for loop [deze moet ook in je code staan].

Tip:

Wees creatief!

Opdracht 3c. List comprehension oefening

Gebruikmakend van list comprehension probeer onderstaande list matrix aan te maken.

Het gaat hier om een 10 x een nested list (elke list bevat de cijfers 0-9). Je moet geen rekening houden met formatting/spacing.

[

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9], [0, 1, 2, 3, 4, 5, 6, 7, 8, 9],

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9], [0, 1, 2, 3, 4, 5, 6, 7, 8, 9],

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9], [0, 1, 2, 3, 4, 5, 6, 7, 8, 9],

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9], [0, 1, 2, 3, 4, 5, 6, 7, 8, 9],

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9], [0, 1, 2, 3, 4, 5, 6, 7, 8, 9],

]

**Let op**: formatting hierboven is enkel om je een idee te geven.

Opdracht 4. Function frequency

Maak een functie frequency aan. Deze zal een list accepteren, doorzoek de list op unieke waarden.

Zorg ervoor dat elke unieke waarde in de list wordt bijgehouden in een dictionary.

De key zal onze unieke waarde zijn, de value het aantal keren dat deze waarde voorkomt in de list.

vb: products = [‘appel’, ‘aap’, ‘opa’, ‘bompa’, 5, True, True]

voorbeeld = {appel: 1, aap: 1, opa: 1, bompa: 1, 5: 1, True: 2}

Bonus:

Wees creatief met het creëren van je data. Denk hierbij aan een csv file, text file, json, …

Opdracht 5. Function list\_manipulation

Schrijf een functie genaamd list\_manipulation. Deze functie zou 4 parameters hebben: list, command, location, value.

* Wanneer command=”remove” en location=”end” is, moet de functie de laatste value in list verwijderen en de verwijderde value teruggeven.
* Wanneer command=”remove” is en location=”beginning”, moet de functie de eerste value in de list verwijderen en de verwijderde value teruggeven.
* Wanneer command=”add” is en location=”beginning”, moet de functie het vierde argument uit de parameterlijst [value] in het begin van de list toevoegen en de list returnen.
* Wanneer command=”add” is en location=”end”, moet de functie het vierde argument uit de parameterlijst [value] aan het einde van de list toevoegen en de list returnen.

Tip: Denk eraan dat onderstaande zaken mogelijk moeten zijn:

list\_manipulation([1, 2, 3], “remove”, “end”) # print out: 3

list\_manipulation([1, 2, 3], “remove”, “beginning”) # print out: 1

list\_manipulation([1, 2, 3], “add”, “beginning”, 20) # print out: [20, 1, 2, 3]

list\_manipulation([1, 2, 3], “add”, “end”, 30) # print out: [1, 2, 3, 30]

Opdracht 6. Built-In Modules:

Definieer een functie met de naam contains\_keyword die een willekeurig aantal string argumenten accepteert.

Het zou True moeten retourneren als een van de argumenten als Python-keywords worden beschouwd [dingen zoals 'def', 'return', 'if', enz ...).

Anders zou False moeten worden geretourneerd.

Belangrijk:

Python heeft een ingebouwde module genaamd keyword die een methode bevat die iskeyword wordt genoemd.

import keyword en gebruik vervolgens de iskeyword methode in je eigen functie om te bepalen of een gegeven string een keyword is.

Opmerking: Controleer niet hardcoded op de zoekwoorden die je kent, zoals return, def, if en for. Bij het testen van je methode kan er gebruik gemaakt worden van elk keyword [ook die je misschien nog niet kent].

Enkele voorbeelden:

contains\_keyword(“hello”, “goodbye”) # print out: False

contains\_keyword(“def”, “haha”, “lol”, “chickens are evil”, 42) # print out: True

contains\_keyword(“four”, “for”, “if”) # print out: True

contains\_keyword(“blabla”, “doggo”, “crab”, “anchor”) # print out: False

contains\_keyword(“grizzly”, “ignore”, “return”, “False”) # print out: True

Opdracht 7. Bank Account OOP oefening:

OOP Oefening bankrekening

Definieer een nieuwe klasse met de naam BankAcount.

Elke BankAccount moet een eigenaar hebben die is opgegeven wanneer een nieuwe BankAccount wordt gemaakt, zoals BankAccount ("Charlie").

Elke BankAccount moet een balance attribuut hebben die altijd begint met 0.0 als initiële waarde.

Elke instantie moet over een deposit methode beschikken die een getal accepteert en deze optelt bij het balance attribuut. De methode moet de nieuwe balance teruggeven.

Elke instantie moet een withdraw methode hebben die een getal accepteert en van de beschikbare balance aftrekt. Ook deze functie returned de overgebleven balance.

Belangrijk:

Houd rekening met de good practices die we samen gezien hebben.

Volgende code zal minimum moeten werken met jouw klasse:

account = BankAccount(“Alex”)

account.owner

account.balance

account.deposit(10)

account.withdraw(3)