Workshop Python voor Wis- en Natuur/Sterrenkundigen

Maandag 28 januari 2019, 11:00 - 13:00

In deze workshop maken we kennis met Python door middel van een aantal eenvoudige opgaven. We beginnen met een opdracht om bekend te worden met de mogelijkheden van de interactieve interpreter. Voor de daarop volgende opgaven is het de bedoeling deze uit te werken in een editor en te draaien met behulp van de interpreter. De interactieve interpreter mag natuurlijk wel als hulpmiddel worden gebruikt bij het opstellen van deze programma's!

1. Open een terminalvenster en start de Python interpreter op door het commando `python3` te geven. Doe vervolgens de volgende oefeningen:

   (a) Maak twee variabelen `a` en `b` aan met een geheel getal of floating-point getal als initiële waarde. Voer vervolgens een aantal berekeningen uit zoals `a + b`, `a * b`, `a ** b` enzovoort. Probeer ook het resultaat op te vangen in een variabele `c`. *Tip: je kunt met "pijl omhoog" je eerdere invoerregels terughalen.*

   (b) Wat gebeurt er als `a` een integer is en `b` een floating-point getal?

   (c) Maak een aantal meer variabelen aan en probeer een paar langere expressies: `(a + b) * c`, `(a ** 3 * c) / d`.

   (d) Bestudeer het verschil tussen `/` en `//`, floating-point respectievelijk integer-deling.

   (e) Ga na dat het optellen van een string en een getal niet werkt, maar het vermenigvuldigen van een string en een getal wel. Kun je het resultaat verklaren?

   (f) Maak twee variabelen met floating-point getallen en zet deze met behulp van `print` en `.format` netjes op het scherm. Maak het eerste getal 4 vakjes breed met 1 getal achter de komma en het tweede getal 10 vakjes breed met 6 plaatsen achter de komma.

   (g) Gegeven de string `s = "een twee drie vier vijf zes zeven acht"`. Maak met behulp van de *slice* notatie (`s[x:y]`) expressies die de woorden "twee", "vier" en "zeven" isoleren. Voorbeeld: `s[:3]` isoleert het woord "een".

   (h) Om de interpreter af te sluiten gebruik je het statement `exit()`.

Zoals je hebt gezien is de interactieve interpreter een handig hulpmiddel. Bij het schrijven van een groot Python programma kun je korte statements en expressies dus in de interactieve interpreter snel uitproberen om te kijken of deze het beoogde resultaat geven.

Voor de volgende opdrachten is het de bedoeling dat deze worden gemaakt door een programma te schrijven in een editor en het daarna te draaien met de Python interpreter. Probeer het eerst een keer op de handmatige manier: maak in je favoriete editor (bijvoorbeeld `gedit` of `kate`) een nieuw bestand: `programma.py` (of welke bestandsnaam je maar wilt). Als je wilt kun je ook kijken of de editor is ingesteld om "net" Python te schrijven: gebruik altijd spaties in plaats van tabs en spring in met 4 spaties. Schrijf vervolgens je programma. Om het programma te draaien geef je het volgende commando in het terminalvenster: `python3 programma.py`.

Heb je goed door hoe het werkt? Dan kun je ook de programma's schrijven en direct draaien binnen de "Spyder" omgeving. Deze is op te starten met het commando `spyder3 &`.

2. Vraag de gebruiker om een string in te voeren (gebruik hiervoor `input()`) die bestaat uit een lijst gehele getallen gescheiden door spaties. Vang de invoer op in een string `s`. Je kunt de ingelezen string als volgt omzetten in een lijst van gehele getallen:

   ```
   # Splits string op spaties en pas op elk element de functie int() toe.
   l = map(int, s.split(" "))
   ```

   Schrijf een loop die alle elementen van deze lijst `l` bezoekt en alle getallen tussen 3 en 44 of deelbaar zijn door 6 afdrukt.

3. *(Zie ook Programmeermethoden opgavenbundel 2.a.)* Schrijf een functie die een gegeven temperatuur in graden Fahrenheit omrekent in graden Celsius. Gebruik hiervoor

$$\text{Temp(in Celsius)} = \frac{5}{9} \times (\text{Temp(in Fahrenheit)} - 32)$$

Vervolgens gebruik je deze functie om een temperatuurschaal af te drukken (maak gebruik van een `for`-loop!):

```
Graden F:    0.0  20.0  40.0  60.0  80.0 100.0 120.0
Graden C:  -17.8  -6.7   4.4  15.6  26.7  37.8  48.9
```

4. Schrijf een functie `def sorteer(lijst):` die de gegeven lijst sorteert met Bubblesort. *Tip: je kunt makkelijk omdraaien met `lijst[j+1], lijst[j] = lijst[j], lijst[j+1].`* Definieer na de functie-definitie de volgende lijst:

```
l = [43, 45, 32, 26, 42, 4, 33, 16, 40, 38, 39, 44, 14, 7, 23]
```

en gebruik een aanroep naar de functie `sorteer` om deze lijst te sorteren. Let op: bij een functie-aanroep met een lijst wordt in feite een pointer naar de lijst `l` doorgegeven (call by reference). Dus aanpassingen gemaakt aan `lijst` binnen de functie `sorteer` zijn na afloop van de functie te zien in `l`. Druk `l` na het sorteren af om het resultaat te controleren.

5. Deze opgave mag je maken met Python of iPython (voor de "Pylab" modus met NumPy al geactiveerd, gebruik `ipython3 --pylab` of gebruik de iPython-console binnen `spyder3`). Schrijf expressies om de volgende arrays te maken. Probeer tot een zo kort mogelijke expressie te komen. For-loops gebruiken is **niet** toegestaan. Ook mag je geen toekenningen doen aan individuele array elementen (maar wel aan slices).

```
(a) [4 4 4 4 4 4]

(b) [ 1  4  7 10 13 16 19 22 25 28]

(c) [[ 1.  0.  0.  0.]
     [ 0.  1.  0.  0.]
     [ 0.  0.  1.  0.]
     [ 0.  0.  0.  1.]]

(d) [[ 9.  0.  0.]
     [ 0.  9.  0.]
     [ 0.  0.  9.]]

(e) [[1 2 3]
     [1 2 3]
     [1 2 3]]

(f) [[ 0.  0.  0.  0.  0.]
     [ 0.  0.  1.  0.  0.]
     [ 0.  1.  2.  3.  4.]
     [ 0.  0.  3.  0.  0.]
     [ 0.  0.  4.  0.  0.]]
```

Hints: (e): je mag ook een lijst of array van elementen als waarde geven aan `np.tile()`, (f): je mag toekenningen aan slices gebruiken.

6. Deze opgave mag je maken met Python of iPython. Definieer de volgende array:

```
A = np.arange(25).reshape(5, 5)
```

Schrijf nu voor elk van de volgende slices van `A` de bijbehorende slice-expressie:

```
(a) 13                      (b) [0 1 2 3 4]

(c) [1 3]                   (d) [[ 0  4]
                                 [ 5  9]
                                 [10 14]
                                 [15 19]
                                 [20 24]]

(e) [[ 6  7  8]             (f) [[ 6  8]
     [11 12 13]                  [16 18]]
     [16 17 18]]
```

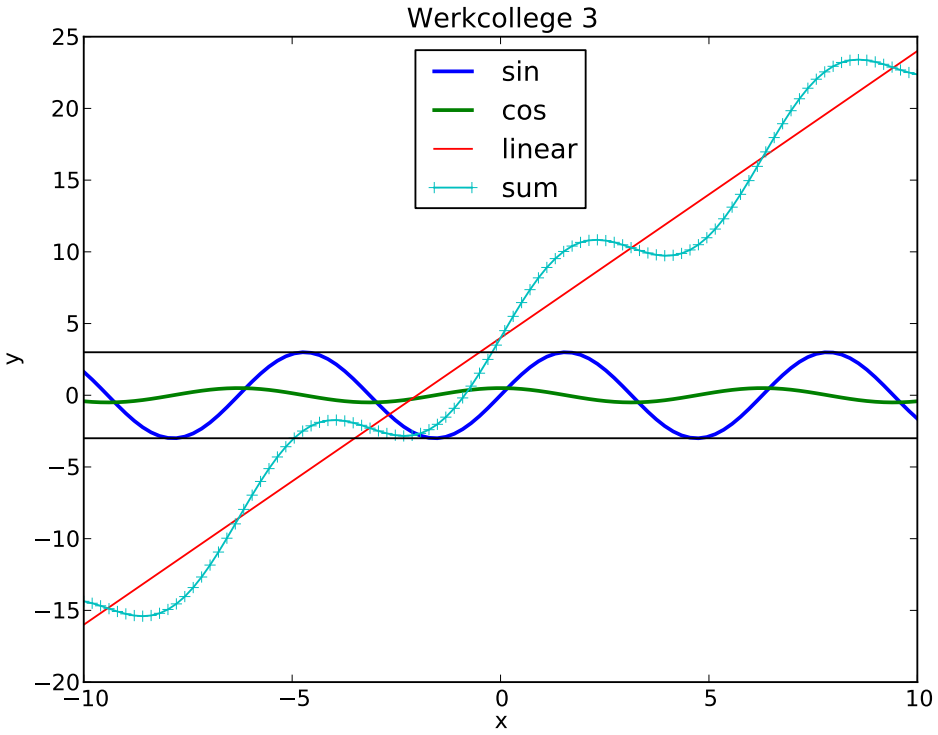
7. Schrijf een programma dat een plot maakt van de volgende functies op een interval $[-10, 10]$ met $N = 100$ stappen (hint: gebruik `np.linspace`):

$$\begin{aligned} y_1 &= 3\sin(x) \\ y_2 &= \frac{1}{2}\cos(x) \\ y_3 &= 2x + 4 \\ y_4 &= y_1 + y_3 \end{aligned}$$

Zorg ervoor dat je de verschillende lijnen goed kunt onderscheiden: maak gebruik van verschillende kleuren, markers, lijndiktes, enz. Geef ook elke lijn een label zodat je een legenda kunt maken.

Plot ook twee **zwarte** lijnen die het bereik van y_1 (voor het interval waarvoor y_1 is geplot) laten zien. *Hint: maak een array met behulp van* `np.tile` *met als parameters de uitkomst van* `np.amin` *en de waarde* N.

Maak de plot netjes op: geef het een titel, plaats aslabels en een legenda. De plot mag of op het scherm worden getoond, of naar een bestand worden geschreven. Voorbeeld:



8. Maak als eerste een *.txt*-bestand waarin op elke regel twee getallen staan gescheiden door een spatie, bijvoorbeeld:

```
65 12
23 42
9 3
43 4
13 95
16 48
36 90
28 5
46 19
22 47
```

   Deze twee kolommen stellen ieder een getallenreeks voor. Schrijf een programma dat dit bestand inleest en beide kolommen in een NumPy-array plaatst, stel `a` en `b`. Tip: lees het bestand regel-voor-regel in, voeg de elementen toe aan een lijst en maak, zodra het hele bestand is gelezen, NumPy-arrays op basis van de lijsten.

   Maak nog twee arrays en sla in de ene array de som van `a` en `b` op en in de andere het verschil. Maak hier gebruik van de element-gewijze NumPy operators (en dus *niet* van een for-loop).

   In plaats van aparte NumPy arrays mag je ook een twee-dimensionale array maken en daar alle data in plaatsen. Experimenteer gerust!

   Nu we vier arrays met getallen hebben, willen we deze graag plotten. We zetten elke getallenreeks uit tegen de x-waarden 0 t/m 9. Maak een aparte array met deze x-waarden en plot iedere getallenreeks.

   (Tip voor de toekomst: NumPy heeft een ingebouwde functie om eenvoudige bestanden bestaande uit getallen in te lezen: `np.loadtxt()`*).*

9. **(+)** Tot nu toe moesten we altijd ons programma draaien door Python expliciet aan te roepen: `python3 programma.py`. We kunnen er ook een echt uitvoerbaar programma van maken, net als C++ programma's. Dit gaat als volgt: in `programma.py`, voeg als de **allereerste** regel in het programma toe:

```
#!/usr/bin/env python3
```

   Dit heet de *she-bang regel*. Sla het programma op. Vervolgens gebruik je de terminal om de "executable-bit" te zetten op je programma: `chmod +x programma.py`. Vanaf nu kun je je programma draaien net als een C++ programma: `./programma.py`.