Koprolkalender

Algemene info

De module `datetime`

De module datetime uit The Python Standard Library definieert een aantal nieuwe gegevenstypes die datums (`datetime.date` objecten) en tijdsintervallen (`datetime.timedelta` objecten) voorstellen. Hieronder alvast enkele voorbeelden.

```
>>> from datetime import date
>>> geboortedatum = date(1990, 10, 3)
>>> geboortedatum = date(day=3, month=10, year=1990)
>>> geboortedatum.day          # day is een eigenschap
3
>>> geboortedatum.month        # month is een eigenschap
10
>>> geboortedatum.year         # year is een eigenschap
1990
>>> geboortedatum.weekday()    # weekday is een methode !!
2
>>> vandaag = date.today()
>>> vandaag
datetime.date(2015, 11, 10)    # uitgevoerd op 11 oktober 2015
>>> from datetime import timedelta
>>> morgen = vandaag + timedelta(1)
>>> morgen
datetime.date(2015, 11, 11)
>>> verschil = morgen - vandaag
>>> type(verschil)
datetime.timedelta
>>> verschil.days
1
```

Specifieke info

In de voorstelling van `datetime.date` objecten worden de maanden chronologisch genummerd van 1 tot en met 12, waarbij januari de eerste maand (maand 1) is en december de laatste maand (maand 12). In deze opgave voegen we daaraan toe dat de dertiende maand terug de maand januari is, de veertiende maand de maand februari, enzoverder. Onderstaande tabel geeft een overzicht van wat we in deze opgave bedoelen met de maanden 12 tot en met 25.

origineel	nieuw	origineel	nieuw
12	12	19	7
13	1	20	8
14	2	21	9
15	3	22	10
16	4	23	11
17	5	24	12
18	6	25	1

Zoals je kunt vaststellen hebben we hier dus terug cyclisch gedrag: na de laatste maand komt terug de eerste maand. Daarbij komt de modulo operator (`%`) altijd handig van pas. Maar aangezien de maanden genummerd worden vanaf 1, en niet vanaf 0, kunnen we de modulo operator niet rechtstreeks gebruiken. Want `12 % 12` is gelijk aan `0`, terwijl de eerste maand (januari) genummerd is als `1`.

De oplossing bestaat er dus in om de nummering 1 tot en met 12 eerst om te zetten naar de nummering 0 tot en met 11, dan de modulo operator toepassen op deze nieuwe nummering, en vervolgens de nieuwe nummering terug omzetten naar de nummering 1 tot en met 12. Dit kan op de volgende manier gebeuren:

```
>>> maand = 15
>>> (maand - 1) % 12 + 1
3
>>> maand = 24
>>> (maand - 1) % 12 + 1
12
```

De miljardjarige oorlog

Specifieke info

Om alle restrictieplaatsen te vinden, kan je eerst itereren over alle mogelijke startposities, waarna je voor elke mogelijke startpositie itereert over alle mogelijke stoppositie. Voor elke combinatie van een start- en stoppositie kan je dan nagaan of de tussenliggende sequentie palindromisch is en voldoet aan de lengtevereisten.

Foutdetectie

Algemene info

De module `random`

De module random uit The Python Standard Library kan gebruikt worden om willekeur toe te voegen aan je Python code. Deze module bevat onder andere de volgende functies.

functie	korte omschrijving
`random()`	genereert een willekeurig reëel getal uit het interval $[0, 1[$
`randint(a, b)`	genereert een willekeurig geheel getal uit het interval $[a, b]$
`choice(s)`	kiest een willekeurig element uit de sequentie `s`
`sample(s, k)`	kiest willekeurig `k` verschillende elementen uit de sequentie `s`
`shuffle(l)`	schudt de elementen van de lijst l willekeurig door elkaar

Hieronder staan alvast enkele voorbeelden.

```
>>> import random

>>> random.random()
0.954131645221452
>>> random.random()
0.3548429482674793
```

```
>>> random.randint(3, 10)
5
>>> random.randint(3, 10)
8

>>> lijst = ['a', 'b', 'c']
>>> random.choice(lijst)
'b'
>>> random.choice(lijst)
'a'
>>> lijst
['a', 'b', 'c']

>>> random.sample(lijst, 2)
['a', 'c']
>>> random.sample(lijst, 2)
['b', 'a']
>>> lijst
['a', 'b', 'c']

>>> random.shuffle(lijst)
>>> lijst
['c', 'a', 'b']
```

Veranderlijke argumenten doorgeven aan functies

Als je een veranderlijk (*mutable*) object doorgeeft aan een functie, dan kan die functie het object *in place* wijzigen. Dat kan expliciet de bedoeling zijn, maar soms is het niet wenselijk dat waarden die aan een functie doorgegeven worden, gewijzigd worden tijdens het uitvoeren van de functie.

Stel bijvoorbeeld dat we een lijst doorgeven aan een functie. Wat we dan eigenlijk doorgeven aan de functie is een verwijzing naar die lijst en geen kopie van de lijst (*call by reference* in plaats van *call by value*). Hierdoor wordt de parameter waaraan de lijst wordt toegekend een alias voor de lijst, en kan de functie dus de lijst zelf wijzigen (lijsten zijn veranderlijke datastructuren).

```
>>> def aanpassen(lijst, element):
...     lijst.append(element)
...     return lijst
...
>>> lijst = ['a', 'b']
>>> aangepast = aanpassen(lijst, 'c')
>>> aangepast
['a', 'b', 'c']
>>> lijst
['a', 'b', 'c']
```

In onderstaand voorbeeld maken we eerst een kopie van de lijst die aan de functie werd doorgegeven. Daarna passen we de kopie, maar dus niet de originele lijst aan. Een kopie maken van een lijst kan je bijvoorbeeld met behulp van *slicing* (`lijst[:]`) of aan de hand van de ingebouwde functie `list` (`list(lijst)`).

```
>>> def aanpassen(lijst, element):
...     kopie = lijst[:]
...     kopie.append(element)
```

```
...     return kopie
...
>>> lijst = ['a', 'b']
>>> aangepast = aanpassen(lijst, 'c')
>>> aangepast
['a', 'b', 'c']
>>> lijst
['a', 'b']
```

De Python Tutor geeft je een grafische voorstelling van het verschil tussen bovenstaande voorbeelden:

- voorbeeld zonder kopie
- voorbeeld met kopie

Omdat we de verwijzing naar de originele lijst die aan de functie werd doorgegeven niet meer nodig hebben, kunnen we de functie `aanpassen` uit bovenstaand voorbeeld ook op de volgende manier hierschrijven.

```
def aanpassen(lijst, element):
    lijst = lijst[:]
    lijst.append(element)
    return lijst
```

Hierbij wordt de verwijzing van de variabele `lijst` naar de originele lijst die aan de functie `aanpassen` wordt doorgegeven, vervangen door een verwijzing naar een kopie van de lijst. Deze vervanging is enkel zichtbaar binnen de functie, omdat de variabele `lijst` een lokale variabele is van de functie `aanpassen`. Ook dit voorbeeld kan je bekijken aan de hand van de Python Tutor.