Loonbrief

Inlezen van meerdere regels

Als je op voorhand weet hoeveel regels er moeten ingelezen worden, dan kan je gebruik maken van de volgende constructie

```
>>> aantal_regels = 4
>>> for _ in range(aantal_regels):
...     regel = input()
...     # doe iets met de regel
...
```

Als je niet op voorhand weet hoeveel regels er zijn, maar je weet dat de laatste regel bijvoorbeeld leeg is, dan kan je gebruik maken van de volgende constructie

```
>>> regel = input()
>>> while regel:
...
...     # doe iets met de regel
...
...     regel = input()
...
```

Conan the Bacterium

Specifieke info

Het eerste experiment

We kunnen het probleem vertalen naar een wiskundige formule. Laat ons het aantal bacteriën na t seconden z_t noemen. Uit de opgave leiden we nu af dat het aantal bacteriën in de proefbuis zich zal gedragen als:

$$z_0 = 1$$
$$z_{t+1} = az_t + b$$

In Python-termen zal dit een lus zijn waarin elke keer `z = a * z + b` uitgevoerd wordt. Beslis zelf of je hier beter een `for`-lus of `while`-lus gebruikt.

Het tweede experiment

Voor het tweede experiment hebben we een gelijkaardige formule:

$$z_0 = t$$
$$z_{t+1} = az_t + b$$

Er zijn hier twee verschillen: de beginwaarde z_0 is anders en we mogen stoppen wanneer de waarde groter is dan bij het eerste experiment. Beslis opnieuw zelf of je hier beter een `for`-lus of `while`-lus gebruikt.

Hittegolf

Lussen vroegtijdig afbreken

In Python kan je gebruik maken van de statements `break` en `continue` om een lus vroegtijdig af te breken. Deze statements worden echter algemeen aanzien als slechte programmeerstijl.

Een situatie waarin je een lus vroegtijdig zou willen afbreken, doet zich bijvoorbeeld voor als je op zoek moet gaan naar een oplossing door een aantal mogelijke gevallen uit te proberen, en te stoppen van zodra je één oplossing gevonden hebt. In plaats van te werken met `break` en `continue` kan je in dit geval beter werken met een variabele die aangeeft of de oplossing reeds gevonden werd

```
>>> gevonden = False
>>> while not gevonden:
...     if (oplossing gevonden): # "oplossing gevonden" stelt voorwaarde voor
...         gevonden = True
...
```

Van zodra de oplossing gevonden werd (hier aangegeven door het feit dat de voorwaarde *oplossing gevonden* de waarde `True` aanneemt), wordt de variabele `gevonden` op `True` gezet, waardoor uit de `while`-lus zal gesprongen wanneer de `while`-voorwaarde opnieuw geëvalueerd wordt na de huidige iteratiestap.

Specifieke info

Bij deze opgave is het een goed idee om bij te houden hoeveel opeenvolgende zomerse dagen je hebt zien passeren, en hoeveel tropische dagen er binnen die afgelopen periode van zomerse dagen werden waargenomen.

Algemeen

Oneindige lussen

Let op voor oneindige lussen. Een oneindige lus is een lus die eindeloos blijft uitgevoerd worden: in de meeste gevallen gaat het om een `while`-lus waarbij de statements binnen de lus er nooit voor zorgen dat de voorwaarde uiteindelijk `False` wordt. Bekijk bij wijze van bijvoorbeeld eens het volgend stuk code

```
>>> i = 0
>>> a = 0
>>> while i < 4:
...     a += 1
```

Omdat het statement `a += 1` er nooit zal voor zorgen dat de initiële waarde van de variabele `i` groter dan of gelijk aan 4 wordt, zal de voorwaarde `i < 4` altijd de waarde `True` opleveren.

Tip: Als je werkt met Pycharm, dan kan je nagaan dat je programma aan het uitvoeren is, door het rode vierkantje links van de Console of rechtsonder de menubalk. Als je op dit vierkantje klikt, dan forceer je om de uitvoering van het programma te stoppen.

Tellen vanaf nul

Informatici beginnen standaard te tellen vanaf 0, niet vanaf 1, en Python volgt deze traditie op heel wat verschillende plaatsen. Zo genereert de ingebouwde functie `range` een reeks opeenvolgende getallen die begint bij 0, als je slechts één enkel argument meegeeft aan de functie. Zo tel je bijvoorbeeld standaard tot 5 in Python

```
>>> for i in range(6):
...     print(i)
...
0
```

```
1
2
3
4
5
```

Als je niet wil beginnen tellen vanaf 0, dan kan je de startwaarde meegeven als een tweede argument aan de `range` functie.

```
>>> for i in range(1, 6):
...     print(i)
...
1
2
3
4
5
```

Het wordt echter als meer *pythonic* beschouwd om het voorgaande op de volgende manier uit te schrijven

```
>>> for i in range(5):
...     print(i + 1)
...
1
2
3
4
5
```