

Inleiding Programmeren

Toon Calders

Les 4: dict, set, tuple, files

Wat zagen we vorige keer?

- dict, set en tuple
- Composite data types (samengestelde data types)
- dict, set en list zijn *mutable*
- tuple, string, int, bool zijn *immutable*

Snelle quiz

- Wat is het resultaat van het volgende code-fragment:

```
L=[{1,2},{(3,4),(3,4)},{7}]  
K=L  
K[0]={3,4}  
print(L)
```

- (A) `[{1,2},{(3,4),(3,4)},{7}]`
- (B) `[{3,4},{(3,4)},{7}]`
- (C) `[{1,2},{(3,4)},{7}]`

Snelle quiz

- Wat is het resultaat van het volgende code-fragment:

```
L=[{1,2},{(3,4),(3,4)},{7}]  
K=L  
K[0]={3,4}  
print(L)
```

- (A) `[{1,2},{(3,4),(3,4)},{7}]`
- (B) `[{3,4},{(3,4)},{7}]`**
- (C) `[{1,2},{(3,4)},{7}]`

Snelle quiz

- Wat is het resultaat van het volgende code-fragment:

```
L=[ {1, 2}, { (3, 4) , (3, 4) }, {7} ]  
K=L[:]  
K[0]={3, 4}  
print(L)
```

- (A) [{1,2}, {(3, 4), (3, 4)}, {7}]
- (B) [{3, 4}, {(3, 4)}, {7}]
- (C) [{1, 2}, {(3, 4)}, {7}]

Snelle quiz

- Wat is het resultaat van het volgende code-fragment:

```
L=[ {1, 2}, { (3, 4) , (3, 4) }, {7} ]  
K=L[:]  
K[0]={3, 4}  
print(L)
```

- (A) [{1,2}, {(3, 4), (3, 4)}, {7}]
- (B) [{3, 4}, {(3, 4)}, {7}]
- (C) [{1, 2}, {(3, 4)}, {7}]**

Snelle quiz

- Wat is het resultaat van het volgende code-fragment:

```
L=[ {1,2}, { (3,4) , (3,4) }, {7} ]  
K=L[:]  
K[0].add(2)  
K[0].add(3)  
print(L)
```

- (A) [{1, 2, 2, 3}, {(3, 4), (3, 4)}, {7}]
- (B) [{1, 2}, {(3, 4)}, {7}]
- (C) [{1, 2, 3}, {(3, 4)}, {7}]

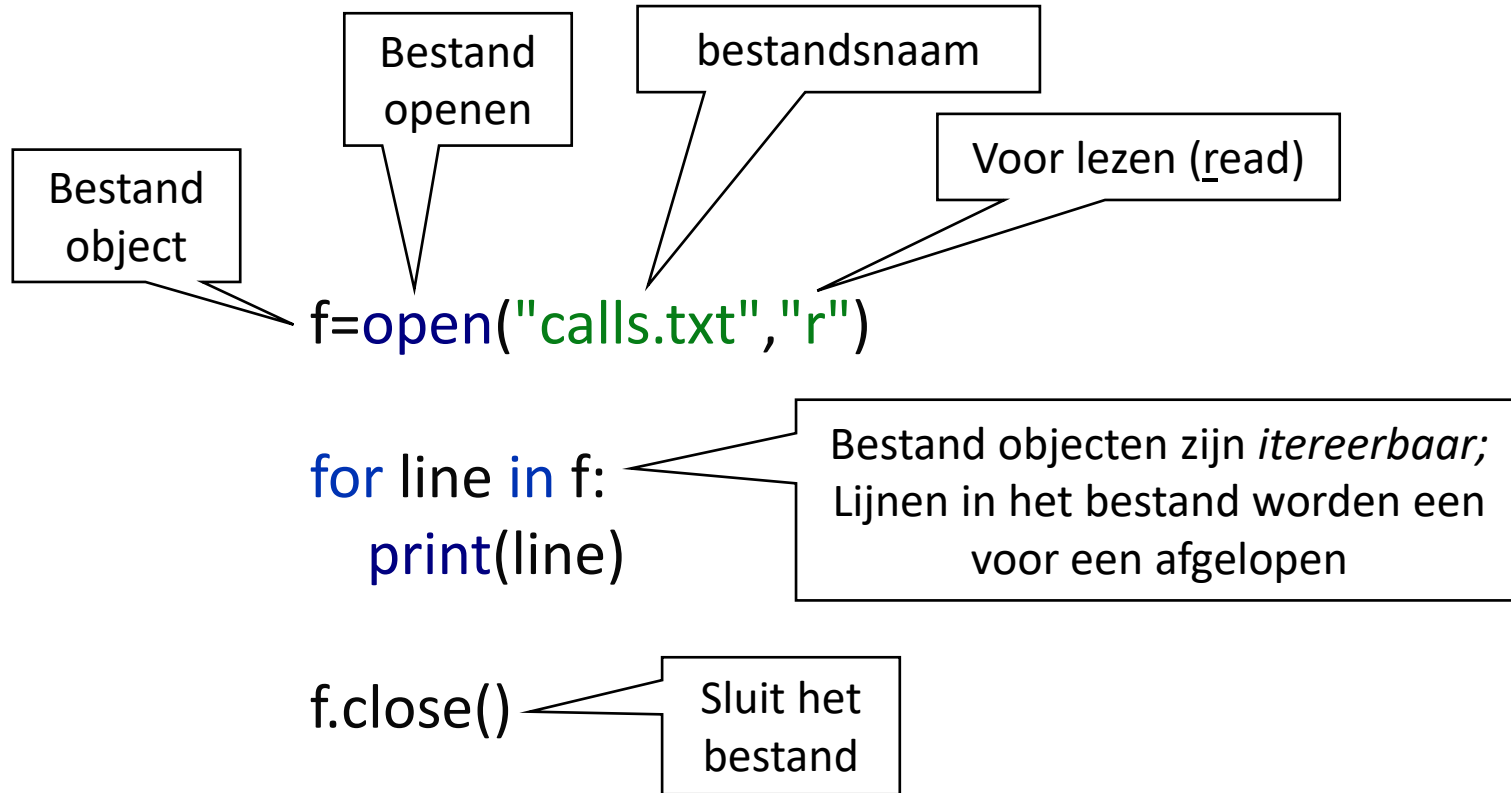
Snelle quiz

- Wat is het resultaat van het volgende code-fragment:

```
L=[ {1,2} , { (3,4) , (3,4) } , {7} ]  
K=L[:]  
K[0].add(2)  
K[0].add(3)  
print(L)
```

- (A) [{1, 2, 2, 3}, {(3, 4), (3, 4)}, {7}]
- (B) [{1, 2}, {(3, 4)}, {7}]
- (C) [{1, 2, 3}, {(3, 4)}, {7}]**

Bestand lezen



Bestand schrijven

```
f=open("calls.txt","r")  
f2=open("calls2.txt","w")
```

Voor schrijven (write)

```
for line in f:  
    f2.write(line)
```

```
f.close()  
f2.close()
```

Schrijf weg naar een bestand

Ingelezen lijn interpreteren

```
f=open("calls.txt","r")
```

De meeste lijnen eindigen met een newline karakter; enkel de allerlaatste lijn in het bestand mogelijk niet

```
for line in f:
```

```
    line=line.strip()
```

```
    record=line.split()
```

```
    print((record[0],record[1]),
```

Dat newline karakter halen we weg

string.split() splitst een string op basis van witruimte (spatie, tab) en geeft een lijst van strings terug

```
f.close()
```

We gaan ervan uit dat er twee strings gescheiden door witruimte per lijn staan; we maken een tuple met deze twee strings

Uitvoer:

Jawad Isabelle
Omar Dayenne
Thibault Anass
Lune Stephen
Thibault Arthur
...

calls.txt

```
f=open("calls.txt","r")  
  
for line in f:  
    line=line.strip()  
    record=line.split()  
  
    print((record[0],record[1]))  
  
f.close()
```

('Jawad', 'Isabelle')
('Omar', 'Dayenne')
('Thibault', 'Anass')
('Lune', 'Stephen')
('Thibault', 'Arthur')

Vragen

- Hoeveel personen zijn er in totaal?
- Wat is het totale aantal gesprekken en wat is het aantal unieke gesprekken?

Vragen

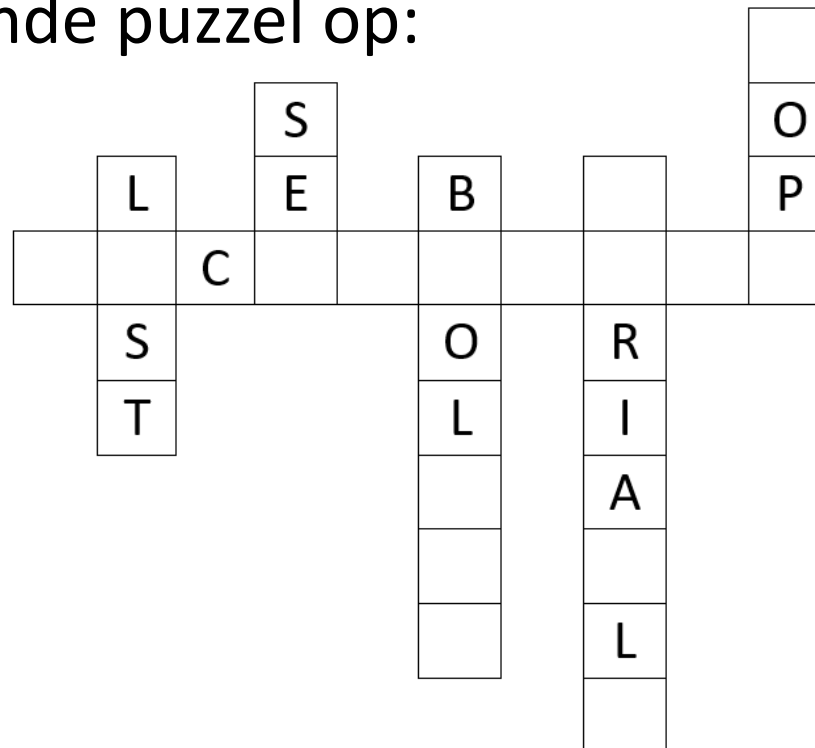
- Welke personen waren bij het grootste aantal gesprekken betrokken (als beller of als opgebelde) en bij hoeveel gesprekken is dat ?
- Welke personen hadden contact met het grootste aantal andere mensen?

Vragen

- Welke paren van mensen hadden het meeste contact met elkaar?

Uitdaging

- Los volgende puzzel op:



```
from words import woordenBoek
print(match("p??h??", woordenBoek))
```


Deze les

- Recursieve functies = functies die zichzelf aanroepen
- Wederzijdse recursie (mutual recursion) = twee of meerdere functies die elkaar aanroepen
- Vergeet niet een basisgeval op te stellen!
- Recursie brengt veel overhead met zich mee: opbouwen en afbreken stack frames
- Als f zichzelf oproept, dan hebben de twee uitvoeringen geen toegang tot elkaars variabelen!

Voorbeeld: recursieve functie

- Bij het oplopen van een trap kan je telkens 1 of 2 treden tegelijk nemen; op hoeveel manieren kan je een trap met n treden oplopen?
- We maken een functie $\text{manieren}(n)$
 - We starten met 1 trede: $\text{manieren}(n-1)$ voor de rest
 - We starten met 2 treden: $\text{manieren}(n-2)$ voor de rest
- Basisgeval vergeten?
 - Oneindige recursie

Stack Frame

```
def manieren(n):
    if n<=1:
        return 1
    return manieren(n-1)+manieren(n-2)

print(manieren(3))
```

- Recursieve aanroepen zijn onafhankelijk van elkaar

```
manieren(3)                                n=3
    if n<=1:
        return 1
    return manieren(n-1)+manieren(n-2)
        manieren(2)                        n=2
            if n<=1:
                return 1
            return manieren(n-1)+manieren(n-2)
                manieren(1)                n=1
                    if n<=1:
                        return 1
                    return manieren(n-1)+manieren(n-2)
```

Algoritme van Euclides voor GCD

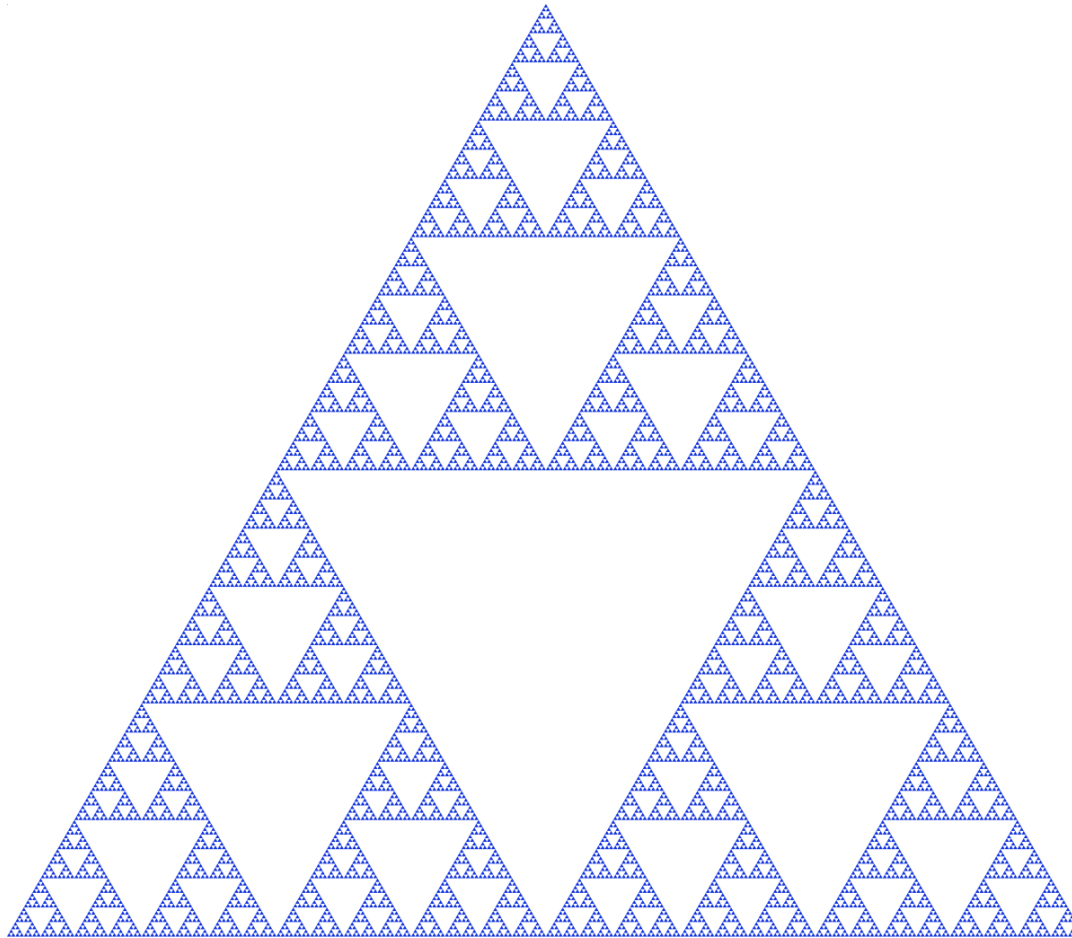
$$\text{gcd}(x, y) = \begin{cases} x & \text{if } y = 0 \\ \text{gcd}(y, \text{remainder}(x, y)) & \text{if } y > 0 \end{cases}$$

Pseudocode (recursive):

```
function gcd is:  
  input: integer x, integer y such that x > 0 and y >= 0  
  
    1. if y is 0, return x  
    2. otherwise, return [ gcd( y, (remainder of x/y) ) ]  
  
end gcd
```

[https://en.wikipedia.org/wiki/Recursion_\(computer_science\)](https://en.wikipedia.org/wiki/Recursion_(computer_science))

Sierpinski driehoek



Conclusie

- Recursieve functie = functie die zichzelf oproept
 - Elke aanroep heeft zijn eigen variabelen (eigen frame)
 - Vergeet niet om een basis geval op te stellen!
- Ideaal om problemen op te lossen die bestaan uit kleinere versies van hetzelfde probleem