

OWE 3a: Python II



**Geavanceerde concepten
in Python en
programmeren voor
bio-informatica
toepassingen**

Opzet OWE 3a

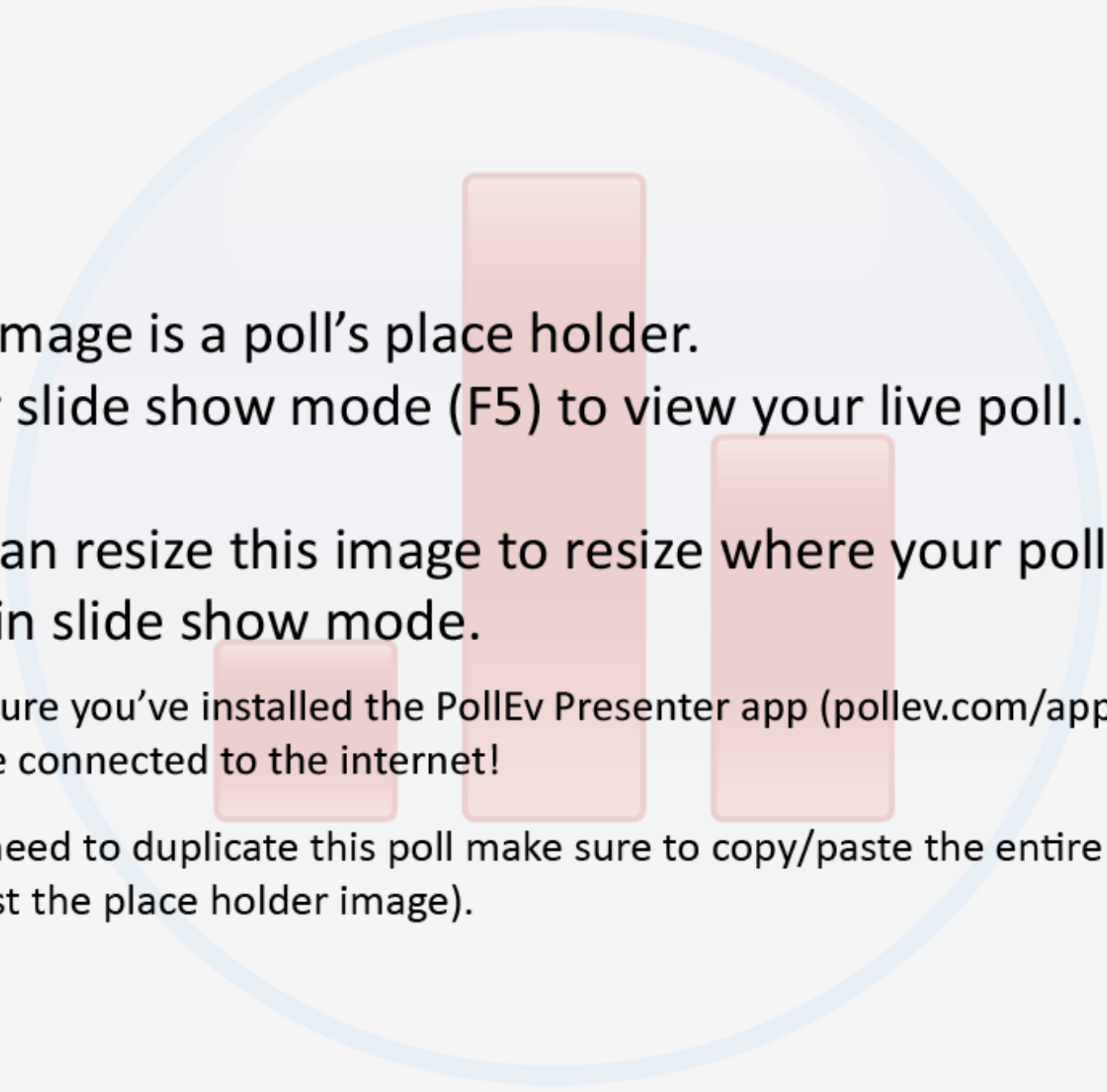
Les	Onderwerp	Theorie	Opgaven
1	<ul style="list-style-type: none"> Review Python I Pseudocode Flowcharts Documenteren en Testen 	H1..8 SowP*	Afvinkopdracht 1
2	<ul style="list-style-type: none"> Graphs Strings 	Matplotlib tutorial H9 More about Strings	Afvinkopdracht 2
3	<ul style="list-style-type: none"> Datastructuren: Dictionaries Sets 	H10 Dictionaries and Sets	Afvinkopdracht 3
4	<ul style="list-style-type: none"> Text and Language Processing Regular Expressions 	H7 DiP**	Afvinkopdracht 4
5	<ul style="list-style-type: none"> Object-Oriented Programming 	H11 Classes and Object-Oriented Programming H12 Inheritance	Afvinkopdracht 5
6	<ul style="list-style-type: none"> Recursion 	H13 Recursion	Afvinkopdracht 6
7	<ul style="list-style-type: none"> GUI Programming 	H14 GUI Programming	Voorbeeld thematoets

*SowP: Starting out with Python

**DiP Dive into Python

Agenda

1. **Dictionaries**
2. **Sets**
3. **Serializing objects**



This image is a poll's place holder.
Enter slide show mode (F5) to view your live poll.

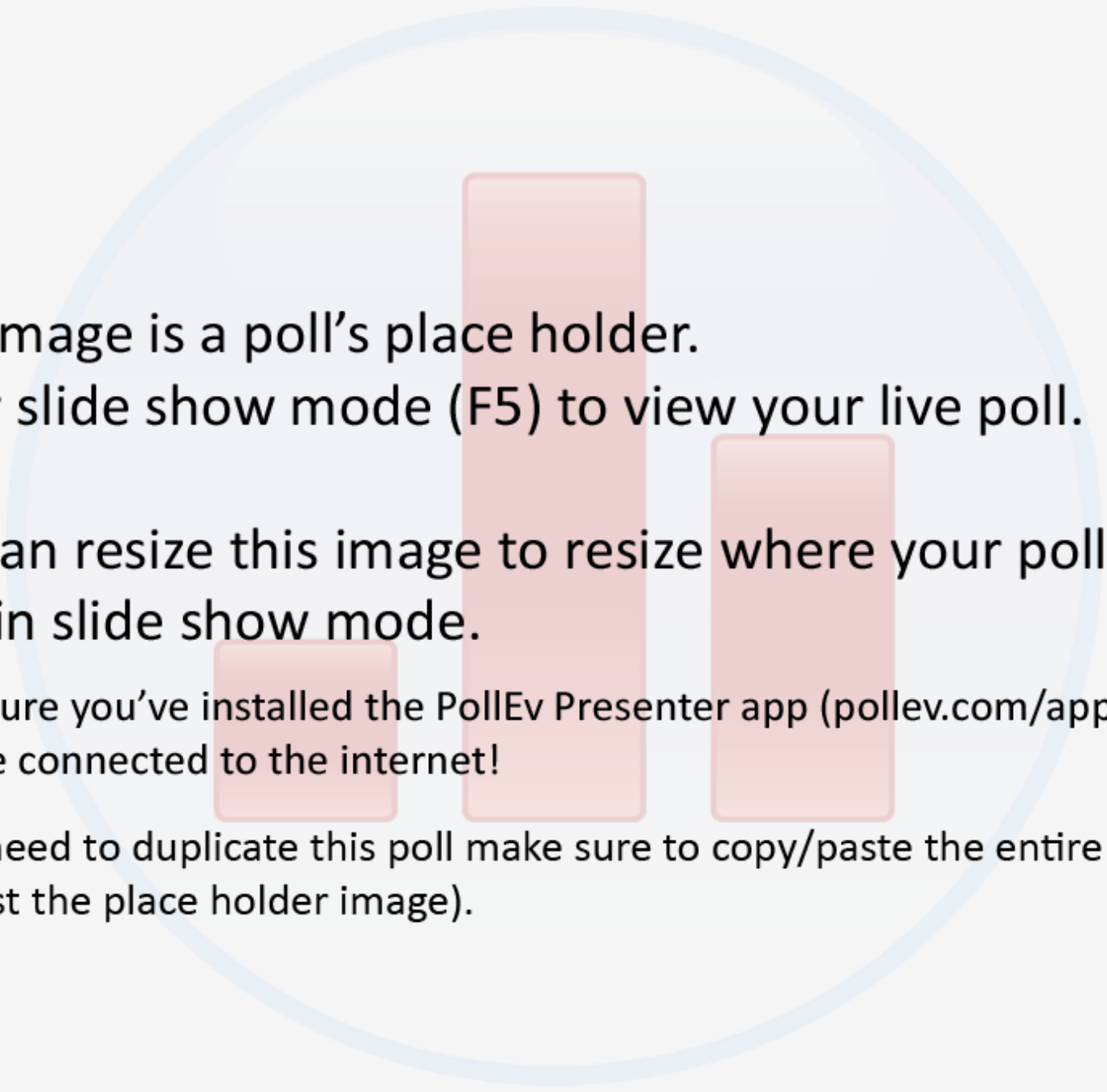
You can resize this image to resize where your poll will
load in slide show mode.

Make sure you've installed the PollEv Presenter app (pollev.com/app)
and are connected to the internet!

If you need to duplicate this poll make sure to copy/paste the entire slide
(not just the place holder image).

Voorbeelden van datastructuren

- String
- List
- Tuples
- Dictionaries
- Sets
- Stack
- Queue
- Deque
- Tree
- Graphs
- BTrees
- Heaps
- Integer
- Float
- Double



This image is a poll's place holder.
Enter slide show mode (F5) to view your live poll.

You can resize this image to resize where your poll will
load in slide show mode.

Make sure you've installed the PollEv Presenter app (pollev.com/app)
and are connected to the internet!

If you need to duplicate this poll make sure to copy/paste the entire slide
(not just the place holder image).

Waarom leren we datastructuren?

- Datastructuren versnellen processen
- Ze zijn intuïtief en werken makkelijker

Wanneer gebruiken we datastructuren?

- Het netwerk van vrienden op Facebook zit in een datastructuur (Graph)
- Het telefoonboek in je telefoon zit in een datastructuur (Dictionary)
- De beschrijving van afstamming (Fylogenetische boom)
- Nederlands-Engels woordenboek (Dictionary)

Lijsten (1/2)

- In Python hebben we list
- Lists zijn makkelijk om lineaire data op te slaan
- In de informatica spreken we meestal over arrays

Lijsten (2/2)

- Lijsten zijn geordend
- Lijsten hebben als nadeel dat het niet heel snel zoeken is naar een item

Dictionaries

- In Python kennen we dictionaries
- Dictionaries heten officieel hashmaps
- Een hashmap maakt het mogelijk heel snel een waarde terug te vinden
- Hashmaps maken gebruik van een sleutel-waarde combinatie

Waarom dictionaries?

```
lijst1 = ["Koe", "Geit", "Mier", "Slang", "Mus"]  
lijst2 = [4, 4, 6, 0, 2]
```

Wat is de betekenis van Lijst2?

```
dieren = ["Koe", "Geit", "Mier", "Slang", "Mus"]  
poten  = [4, 4, 6, 0, 2]
```

Code om vraag te beantwoorden

- Hoe schrijven we de code om de vraag te beantwoorden hoeveel poten een koe heeft?

```
dieren = ["Koe", "Geit", "Mier", "Slang", "Mus"]  
poten  = [4, 4, 6, 0, 2]
```

```
dier_vraag = input ("Aantal poten van een ")  
teller = 0  
for dier in dieren:  
    if dier == dier_vraag:  
        print ("Uw dier heeft ",poten[teller], " poten")  
        teller += 1
```

Met een dictionary

- De vraag hoeveel poten een dier heeft is nu razendsnel te beantwoorden
- Er zijn *key-value pairs*

Key

Value

```
dieren = {"Koe":4, "Geit":4, "Mier":6, "Slang":0, "Mus":2}
dier_vraag = input ("Aantal poten van een ")
print ("Uw dier heeft ",dieren[dier_vraag], " poten")
```

Vraag

- Maar kan ik ook de vraag beantwoorden welke dieren 6 poten hebben?

```
dieren = {"Koe":4, "Geit":4, "Mier":6, "Slang":0, "Mus":2}
dier_vraag = input ("Aantal poten van een ")
print ("Uw dier heeft ",dieren[dier_vraag], " poten")
```

keys() en values()

De keys van een dictionary

```
>>> print (dieren.keys())  
dict_keys(['Mier', 'Geit', 'Koe', 'Mus', 'Slang'])
```

De values van een dictionary

```
>>> print (dieren.values())  
dict_values([6, 4, 4, 2, 0])
```


Welke dieren hebben 6 poten?

- De dictionary is zeer geschikt om te bepalen hoeveel poten een dier
- De dictionary is totaal ongeschikt om te bepalen welke dieren een bepaald aantal poten hebben

```
# Welke dieren hebben 6 poten?
```

```
teller = 0
```

```
lijst = list(dieren.values())
```

```
for aantal in lijst:
```

```
    if aantal == 6:
```

```
        print (list(dieren.keys())[teller])
```

```
teller += 1
```

Dictionaries

- Een woordenboek **Nederlands-Engels** is erg geschikt om het Nederlandse woord **lintworm** op te zoeken
- Een woordenboek **Engels-Nederlands** is totaal ongeschikt om het Engelse woord voor **lintworm** op te zoeken

Dictionary in de bio-informatica

- Al onze cellen hebben ook een dictionary...

```
code = { 'ttt': 'F', 'tct': 'S', 'tat': 'Y', 'tgt': 'C',
        'ttc': 'F', 'tcc': 'S', 'tac': 'Y', 'tgc': 'C',
        'tta': 'L', 'tca': 'S', 'taa': '*', 'tga': '*',
        'ttg': 'L', 'tcg': 'S', 'tag': '*', 'tgg': 'W',
        'ctt': 'L', 'cct': 'P', 'cat': 'H', 'cgt': 'R',
        'ctc': 'L', 'ccc': 'P', 'cac': 'H', 'cgc': 'R',
        'cta': 'L', 'cca': 'P', 'caa': 'Q', 'cga': 'R',
        'ctg': 'L', 'ccg': 'P', 'cag': 'Q', 'cgg': 'R',
        'att': 'I', 'act': 'T', 'aat': 'N', 'agt': 'S',
        'atc': 'I', 'acc': 'T', 'aac': 'N', 'agc': 'S',
        'ata': 'I', 'aca': 'T', 'aaa': 'K', 'aga': 'R',
        'atg': 'M', 'acg': 'T', 'aag': 'K', 'agg': 'R',
        'gtt': 'V', 'gct': 'A', 'gat': 'D', 'ggc': 'G',
        'gtc': 'V', 'gcc': 'A', 'gac': 'D', 'ggc': 'G',
        'gta': 'V', 'gca': 'A', 'gaa': 'E', 'gga': 'G',
        'gtg': 'V', 'gcg': 'A', 'gag': 'E', 'ggg': 'G'
    }
```

Werking dictionary

```
code = { 'ttt': 'F', 'tct': 'S', 'tat': 'Y', 'tgt': 'C',
        'ttc': 'F', 'tcc': 'S', 'tac': 'Y', 'tgc': 'C',
        'tta': 'L', 'tca': 'S', 'taa': '*', 'tga': '*',
        'ttg': 'L', 'tcg': 'S', 'tag': '*', 'tgg': 'W',
        'ctt': 'L', 'cct': 'P', 'cat': 'H', 'cgt': 'R',
        'ctc': 'L', 'ccc': 'P', 'cac': 'H', 'cgc': 'R',
        'cta': 'L', 'cca': 'P', 'caa': 'Q', 'cga': 'R',
        'ctg': 'L', 'ccg': 'P', 'cag': 'Q', 'cgg': 'R',
        'att': 'I', 'act': 'T', 'aat': 'N', 'agt': 'S',
        'atc': 'I', 'acc': 'T', 'aac': 'N', 'agc': 'S',
        'ata': 'I', 'aca': 'T', 'aaa': 'K', 'aga': 'R',
        'atg': 'M', 'acg': 'T', 'aag': 'K', 'agg': 'R',
        'gtt': 'V', 'gct': 'A', 'gat': 'D', 'ggt': 'G',
        'gtc': 'V', 'gcc': 'A', 'gac': 'D', 'ggc': 'G',
        'gta': 'V', 'gca': 'A', 'gaa': 'E', 'gga': 'G',
        'gtg': 'V', 'gcg': 'A', 'gag': 'E', 'ggg': 'G'
    }
```

`code['act']`

T

Kenmerken dictionary

- Een dictionary is ongeordend
- Bevat altijd een sleutel-waarde combinatie
- De waarde mag zelf wel weer een complexe datastructuur zijn als een list, set of dictionary
- De snelheid van het retourneren van een waarde is onafhankelijk van de grootte van de dictionary

Samengevat

- **Dictionaries zijn erg geschikt om alles wat een sleutel-waarde combinatie op te slaan**
- **Het zorgt er voor dat je razendsnel hetgeen je zoekt terug krijgt**

Agenda

1. **Dictionaries**
2. **Sets**
3. **Serializing objects**

Sets

- Python heeft net als SQL sets
- We kunnen in Python ook met verzamelingen werken en met set operatoren bewerkingen op uitvoeren

SQL	Python	Betekenis
Union	Union	Samenvoeging
Intersect	Intersection	Overlap
Minus	Difference	Verschil

Creatie van een set

- Een set maak je met het sleutelwoord **set**

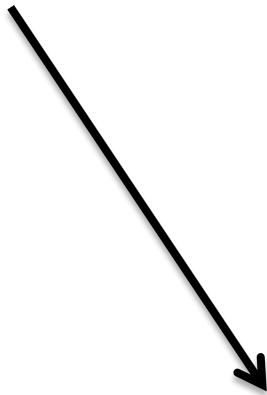
```
mijnSet = set(["apen", "noten"])
```

- Voor het toevoegen van waarden gebruik je **add**

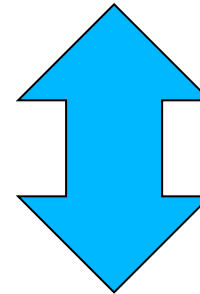
```
mijnSet.add("miezen")
```

Sets bestaan uit unieke waarden

```
boerderij = set(["Koe", "Geit", "Varken", "Konijn", "Konijn"])
dierentuin = set(["Olifant", "Giraffe", "Zebra", "Konijn", "Geit"])
print (boerderij)
print (dierentuin)|
```

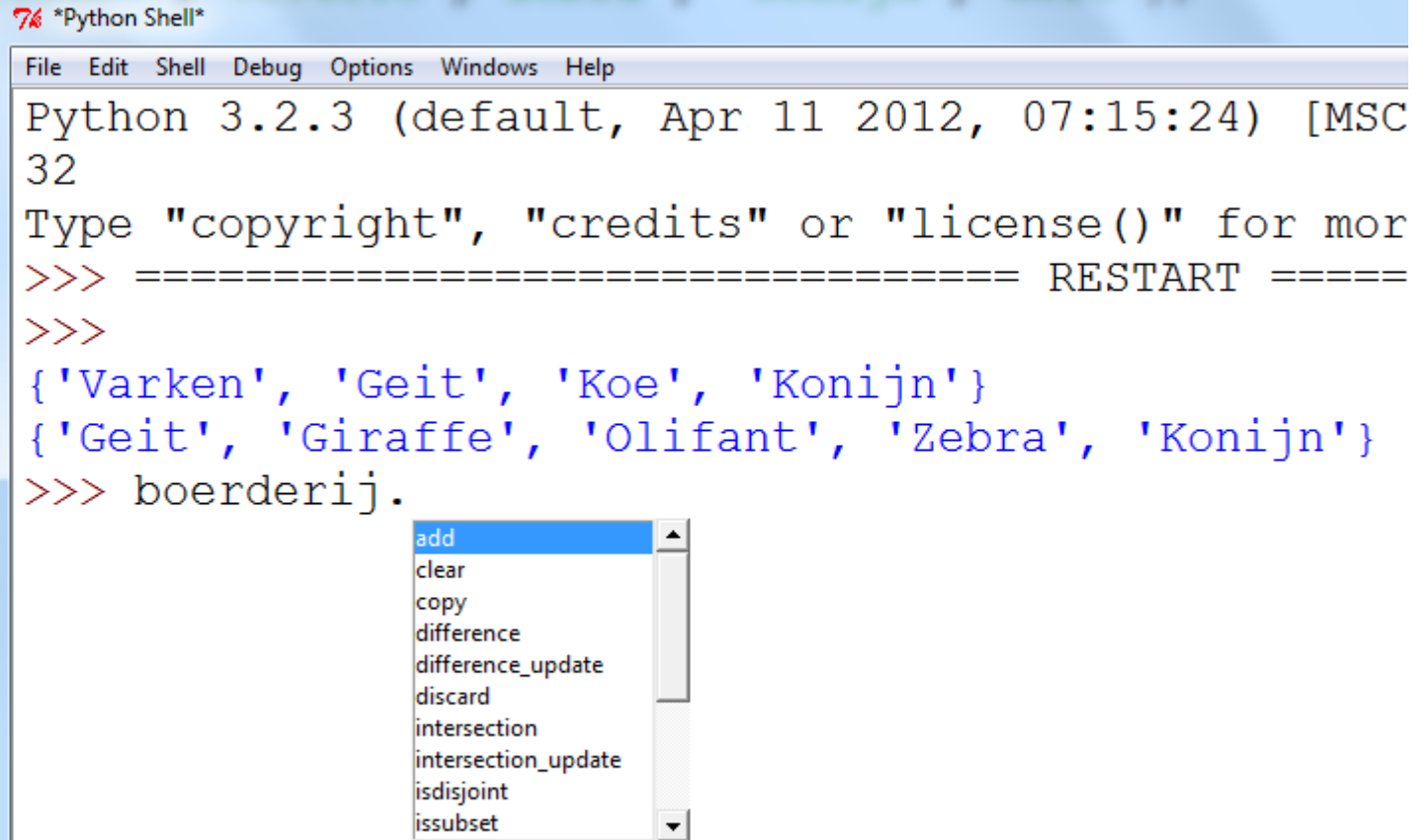


```
{'Varken', 'Geit', 'Koe', 'Konijn'}
{'Geit', 'Giraffe', 'Olifant', 'Zebra', 'Konijn'}
```



Sets hebben veel eigen methodes

```
boerderij = set(["Koe", "Geit", "Varken", "Konijn", "Konijn"])
dierentuin = set(["Olifant", "Giraffe", "Zebra", "Konijn", "Geit"])
print (boerderij)
print (dierentuin)
```



```
*Python Shell*
File Edit Shell Debug Options Windows Help
Python 3.2.3 (default, Apr 11 2012, 07:15:24) [MSC
32
Type "copyright", "credits" or "license()" for mor
>>> ===== RESTART =====
>>>
{'Varken', 'Geit', 'Koe', 'Konijn'}
{'Geit', 'Giraffe', 'Olifant', 'Zebra', 'Konijn'}
>>> boerderij.
```

- add
- clear
- copy
- difference
- difference_update
- discard
- intersection
- intersection_update
- isdisjoint
- issubset

Verschil

```
boerderij = set(["Koe", "Geit", "Varken", "Konijn", "Konijn"])  
dierentuin = set(["Olifant", "Giraffe", "Zebra", "Konijn", "Geit"])
```

```
>>> dierentuin.difference (boerderij)  
{'Giraffe', 'Olifant', 'Zebra'}
```

Subset en superset

- Met `issubset` kunnen we bepalen of het een deelverzameling betreft
- Met `issuperset` kunnen we bepalen of het de ander set ook omvat

```
>>> set1 = set([1, 2, 3, 4])
>>> set2 = set([2, 3])
>>> set1.issubset (set2)
False
>>> set2.issubset (set1)
True
>>> set1.issuperset (set2)
True
>>> set2.issuperset (set1)
False
>>>
```

Kenmerken sets

- **Sets bevatten altijd unieke waarden**
- **Sets zijn ongeordend: de volgorde waarin waarden voorkomen is willekeurig**
- **Sets mogen zowel getallen als strings bevatten**
- **Sets zijn zeer geschikt voor het vergelijken van verzameling**

Samenvattend

- **Sets zijn ongeordende verzamelingen van unieke waarden**
- **Ze zijn erg geschikt om verschillen en overeenkomsten tussen groepen te bepalen**
- **Sets zijn gebaseerd op de verzamelingenleer net als de database taal SQL**

Agenda

1. Dictionaries
2. Sets
3. **Serializing objects**

Wat is serialization?

- Met serialization *bevries* je objecten (datastructuren) en kun je deze opslaan in een bestand
- Serialization heeft tot doel objecten tijdelijk op te slaan en later weer te kunnen gebruiken



This image is a poll's place holder.
Enter slide show mode (F5) to view your live poll.

You can resize this image to resize where your poll will
load in slide show mode.

Make sure you've installed the PollEv Presenter app (pollev.com/app)
and are connected to the internet!

If you need to duplicate this poll make sure to copy/paste the entire slide
(not just the place holder image).

Kenmerken serialization

- Objecten en datastructuren worden tijdelijk opgeslagen voor gebruik later
- In Python wordt niet gesproken over serialization maar over pickling (op zuur zetten)

```
import pickle

bestandsnaam = "studenten.dat"

def bepaalBi():
    meer = 'j'
    try:
        input_file = open(bestandsnaam, 'rb')
        bil = pickle.load(input_file)
        input_file.close()
    except IOError:
        bil = set()
    while meer.lower() == 'j':
        print("Studenten in Bil: ", bil)
        student = input('Student: ')
        bil.add(student)
        meer = input("Doorgaan (j/n)")
    output_file = open(bestandsnaam, 'wb')
    pickle.dump(bil, output_file)
    output_file.close()
```

Read van een Binary bestand

Laden van een pickle bestand

Als bestand niet bestaat: maak lege set

Write van een Binary bestand

Dump van een pickle bestand

```
def main():
    bepaalBi()
```

```
main()
```

Samenvatting serialization

- **Serialization maakt het mogelijk complexe datastructuren op te slaan**
- **Gebruik in Python daarvoor de module pickle**

Samenvatting datastructuren

- **Datastructure**
 - HashTable (Dictionaries)
 - Sets
- **Serializing objects**

Voorbeeld vragen



Voorbeeldvraag

- Bij welke datastructuur is de data niet muteerbaar na initialisatie?
 - a. List
 - b. Tuple
 - c. Dictionary
 - d. Stack
 - e. Sets

Voorbeeldvraag

Welke datastructuren heeft deze code?

```
aa3 = {"Ala": ["GCU", "GCC", "GCA", "GCG"],
      "Arg": ["CGU", "CGC", "CGA", "CGG", "AGA", "AGG"],
      "Asn": ["AAU", "AAC"],
      "Asp": ["GAU", "GAC"],
      "Cys": ["UGU", "UGC"],
      "Gln": ["CAA", "CAG"],
      "Glu": ["GAA", "GAG"],
      "Gly": ["GGU", "GGC", "GGA", "GGG"],
      "His": ["CAU", "CAC"],
      "Ile": ["AUU", "AUC", "AUA"],
      "Leu": ["UUA", "UUG", "CUU", "CUC", "CUA", "CUG"],
      "Lys": ["AAA", "AAG"],
      "Met": ["AUG"],
      "Phe": ["UUU", "UUC"],
      "Pro": ["CCU", "CCC", "CCA", "CCG"],
      "Ser": ["UCU", "UCC", "UCA", "UCG", "AGU", "AGC"],
      "Thr": ["ACU", "ACC", "ACA", "ACG"],
      "Trp": ["UGG"],
      "Tyr": ["UAU", "UAC"],
      "Val": ["GUU", "GUC", "GUA", "GUG"],
      "Start": ["AUG", "CUG", "UUG", "GUG", "AUU"],
      "Stop" : ["UAG", "UGA", "UAA"]}
}
```

Voorbeeldvraag

Wat print `aa3["Leu"][2][2]`

```
aa3 = {"Ala": ["GCU", "GCC", "GCA", "GCG"],
      "Arg": ["CGU", "CGC", "CGA", "CGG", "AGA", "AGG"],
      "Asn": ["AAU", "AAC"],
      "Asp": ["GAU", "GAC"],
      "Cys": ["UGU", "UGC"],
      "Gln": ["CAA", "CAG"],
      "Glu": ["GAA", "GAG"],
      "Gly": ["GGU", "GGC", "GGA", "GGG"],
      "His": ["CAU", "CAC"],
      "Ile": ["AUU", "AUC", "AUA"],
      "Leu": ["UUA", "UUG", "CUU", "CUC", "CUA", "CUG"],
      "Lys": ["AAA", "AAG"],
      "Met": ["AUG"],
      "Phe": ["UUU", "UUC"],
      "Pro": ["CCU", "CCC", "CCA", "CCG"],
      "Ser": ["UCU", "UCC", "UCA", "UCG", "AGU", "AGC"],
      "Thr": ["ACU", "ACC", "ACA", "ACG"],
      "Trp": ["UGG"],
      "Tyr": ["UAU", "UAC"],
      "Val": ["GUU", "GUC", "GUA", "GUG"],
      "Start": ["AUG", "CUG", "UUG", "GUG", "AUU"],
      "Stop": ["UAG", "UGA", "UAA"]}
}
```

Voorbeeldvraag

Wat print `aa3.keys()`

```
aa3 = {"Ala": ["GCU", "GCC", "GCA", "GCG"],
      "Arg": ["CGU", "CGC", "CGA", "CGG", "AGA", "AGG"],
      "Asn": ["AAU", "AAC"],
      "Asp": ["GAU", "GAC"],
      "Cys": ["UGU", "UGC"],
      "Gln": ["CAA", "CAG"],
      "Glu": ["GAA", "GAG"],
      "Gly": ["GGU", "GGC", "GGA", "GGG"],
      "His": ["CAU", "CAC"],
      "Ile": ["AUU", "AUC", "AUA"],
      "Leu": ["UUA", "UUG", "CUU", "CUC", "CUA", "CUG"],
      "Lys": ["AAA", "AAG"],
      "Met": ["AUG"],
      "Phe": ["UUU", "UUC"],
      "Pro": ["CCU", "CCC", "CCA", "CCG"],
      "Ser": ["UCU", "UCC", "UCA", "UCG", "AGU", "AGC"],
      "Thr": ["ACU", "ACC", "ACA", "ACG"],
      "Trp": ["UGG"],
      "Tyr": ["UAU", "UAC"],
      "Val": ["GUU", "GUC", "GUA", "GUG"],
      "Start": ["AUG", "CUG", "UUG", "GUG", "AUU"],
      "Stop": ["UAG", "UGA", "UAA"]}
}
```

Voorbeeldvraag

Wat print `aa3.keys()[2]`

```
aa3 = {"Ala": ["GCU", "GCC", "GCA", "GCG"],
      "Arg": ["CGU", "CGC", "CGA", "CGG", "AGA", "AGG"],
      "Asn": ["AAU", "AAC"],
      "Asp": ["GAU", "GAC"],
      "Cys": ["UGU", "UGC"],
      "Gln": ["CAA", "CAG"],
      "Glu": ["GAA", "GAG"],
      "Gly": ["GGU", "GGC", "GGA", "GGG"],
      "His": ["CAU", "CAC"],
      "Ile": ["AUU", "AUC", "AUA"],
      "Leu": ["UUA", "UUG", "CUU", "CUC", "CUA", "CUG"],
      "Lys": ["AAA", "AAG"],
      "Met": ["AUG"],
      "Phe": ["UUU", "UUC"],
      "Pro": ["CCU", "CCC", "CCA", "CCG"],
      "Ser": ["UCU", "UCC", "UCA", "UCG", "AGU", "AGC"],
      "Thr": ["ACU", "ACC", "ACA", "ACG"],
      "Trp": ["UGG"],
      "Tyr": ["UAU", "UAC"],
      "Val": ["GUU", "GUC", "GUA", "GUG"],
      "Start": ["AUG", "CUG", "UUG", "GUG", "AUU"],
      "Stop": ["UAG", "UGA", "UAA"]}
}
```

Voorbeeldvraag

Wat print `list(set(aa3.keys()))[2]`

```
aa3 = {"Ala": ["GCU", "GCC", "GCA", "GCG"],
      "Arg": ["CGU", "CGC", "CGA", "CGG", "AGA", "AGG"],
      "Asn": ["AAU", "AAC"],
      "Asp": ["GAU", "GAC"],
      "Cys": ["UGU", "UGC"],
      "Gln": ["CAA", "CAG"],
      "Glu": ["GAA", "GAG"],
      "Gly": ["GGU", "GGC", "GGA", "GGG"],
      "His": ["CAU", "CAC"],
      "Ile": ["AUU", "AUC", "AUA"],
      "Leu": ["UUA", "UUG", "CUU", "CUC", "CUA", "CUG"],
      "Lys": ["AAA", "AAG"],
      "Met": ["AUG"],
      "Phe": ["UUU", "UUC"],
      "Pro": ["CCU", "CCC", "CCA", "CCG"],
      "Ser": ["UCU", "UCC", "UCA", "UCG", "AGU", "AGC"],
      "Thr": ["ACU", "ACC", "ACA", "ACG"],
      "Trp": ["UGG"],
      "Tyr": ["UAU", "UAC"],
      "Val": ["GUU", "GUC", "GUA", "GUG"],
      "Start": ["AUG", "CUG", "UUG", "GUG", "AUU"],
      "Stop": ["UAG", "UGA", "UAA"]}
}
```

Voorbeeldvraag

Wat print `aa3.values()`

```
aa3 = {"Ala": ["GCU", "GCC", "GCA", "GCG"],
      "Arg": ["CGU", "CGC", "CGA", "CGG", "AGA", "AGG"],
      "Asn": ["AAU", "AAC"],
      "Asp": ["GAU", "GAC"],
      "Cys": ["UGU", "UGC"],
      "Gln": ["CAA", "CAG"],
      "Glu": ["GAA", "GAG"],
      "Gly": ["GGU", "GGC", "GGA", "GGG"],
      "His": ["CAU", "CAC"],
      "Ile": ["AUU", "AUC", "AUA"],
      "Leu": ["UUA", "UUG", "CUU", "CUC", "CUA", "CUG"],
      "Lys": ["AAA", "AAG"],
      "Met": ["AUG"],
      "Phe": ["UUU", "UUC"],
      "Pro": ["CCU", "CCC", "CCA", "CCG"],
      "Ser": ["UCU", "UCC", "UCA", "UCG", "AGU", "AGC"],
      "Thr": ["ACU", "ACC", "ACA", "ACG"],
      "Trp": ["UGG"],
      "Tyr": ["UAU", "UAC"],
      "Val": ["GUU", "GUC", "GUA", "GUG"],
      "Start": ["AUG", "CUG", "UUG", "GUG", "AUU"],
      "Stop": ["UAG", "UGA", "UAA"]}
}
```

Welke datastructuur?

Datastructuren en recursie (40 punten)

GeneLists.py

```
def getList (b):  
    l = []  
    for r in open (b):  
        l += [r.replace('\n', '')]  
    return l  
  
l1 = getList (r"human.csv")  
l2 = getList (r"mouse.csv")  
l3 = getList (r"monkey.csv")
```

3. Gegeven is bovenstaande code, de code levert drie lijsten van ieder ongeveer 30000 genen. Schrijf de aanvullende code om zonder iteratie de lijst op te leveren die alle genen weergeeft die alleen in de mens voor komen en niet in de muis of in de aap. (20 pt)

Appendix



**Tellen van codons en
aminozuren**

Bepaal aantal aminozuren

```
# Bepaal per aminozuur aantal in een sequentie
seq = "MTSSRLWFSLLLAAAFAGRATALWPWPQNFQTS DQRYVLYPNNFQFYDV"
A = seq.count("A")
B = seq.count("B")
C = seq.count("C")
D = seq.count("D")
G = seq.count("G")
H = seq.count("H")
K = seq.count("K")
M = seq.count("M")
N = seq.count("N")
R = seq.count("R")
S = seq.count("S")
T = seq.count("T")
V = seq.count("V")
W = seq.count("W")
Y = seq.count("Y")
print ("A =", A, "B =", B, "C =", C, "D =", D, "G =", G, "H =", H, "K =", K, "M
```

Bepaal aantal aminozuren

- Met een dictionary is het veel eenvoudiger tellen

```
aantal = {}  
for letter in "MTSSRLWFSLLLAAAFAGRATALWPWPQNFQTSDQRYVLYPNNFQFQYDV":  
    aantal[letter] = aantal.count(letter)  
print (aantal)
```

Bepaal het aantal codons

```
seq = "TCTCCAAGACGCATCCCAGTG"
codon_aantal = {}
for i in range(0, len(seq) - len(seq)%3, 3):
    codon = seq[i:i+3]
    codon_aantal[codon] = 0
    codon_aantal[codon] = codon_aantal[codon] + 1
print (codon_aantal)
```

```
{ 'ATC': 1, 'GTG': 1, 'TCT': 1, 'AGA': 1, 'CCA': 1, 'CGC': 1 }
```

Verantwoording

- In deze uitgave is géén auteursrechtelijk beschermd werk opgenomen
- Alle teksten © Martijn van der Bruggen/HAN tenzij expliciet externe bronnen zijn aangegeven
- Screenshots op basis van eigen werk auteur en/of vernoemde sites
- Eventuele images zijn opgenomen met vermelding van bron