BIO-INFORMATICA

OWE2A PYTHON II

PROGRAMMEREN IN PYTHON II

HAN_UNIVERSITY
OF APPLIED SCIENCES

CURRICULUM OVERZICHT

	Periode 1	Periode 2	Periode 3	Periode 4
1e jaar	Programmeren in Python (1)	Programmeren in Python (2)	Informatiesystemen met Python	Geautomatiseerde identificatie van eiwitten via sequentievergelijking
	Metabolisme	Opsporen van genetische mutaties bij erfelijke ziektes	Vergelijkende genoomanalyse: evolutie van virussen	Sequentie alignment (BLAST) en functionele eiwit analyse
2e jaar	Programmeren in Java	Datastructuren en algoritmen in Java	Analyse en ontwerptechnieken	Webtechnologie en textmining
	Proteomics: eiwitstructuren, eiwitfuncties, scheidingstechnieken en data analyse	Transcriptomics: data analyse mbt regulatie van metabole routes	Genomics: annoteren van genomisch DNA	Moleculaire fylogenie: evolutie, multiple sequence alignment mbt signaaltransductie
3e jaar	Stage of minor		Data Mining en grid computing	RNA seq, biostatistiek,
			Eiwitten: carcinogenese en eiwitstructuren	Workflows
4e jaar	Stage, afstudeerstage of minor		Minor of afstudeerstage	

JAAR 1

Basis van programmeren in Python

Complexe structuren

Databases voor opslag van data

	Periode 1	Periode 2	Periode 3	Periode 4
1e jaar	Programmeren in Python (1)	Programmeren in Python (2)	Informatiesystemen met Python	Geautomatiseerde identificatie van eiwitten via sequentievergelijking
	Metabolisme	Opsporen van genetische mutaties bij erfelijke ziektes	Vergelijkende genoomanalyse: evolutie van virussen	Sequentie alignment (BLAST) en functionele eiwit analyse

Ontwikkelen van applicaties voor biologisch analyse en visualisatie op het web

OVERZICHT OWE 1A INFORMATICA

Week	Onderwerpen	
1	IDE: PyCharm Debuggen en testen PEP richtlijnen	PEP tutorial
2	Grafieken met MatPlotLib	MatPlotLib.org
3	Datastructuren	H9 SowP
4	Regular Expressions	H7 DiP
5	Object Oriëntatie	H10 & H11 SowP
6	Recursie	H12 SowP
7	Graphical User Interface	H13 SowP

AGENDA

1. Datastructuren
2. Dictionaries
3. Sets
4. Serializing objects

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)

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 als je de index niet kent

AGENDA

1. Datastructuren
2. Dictionaries
3. Sets
4. Serializing objects

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

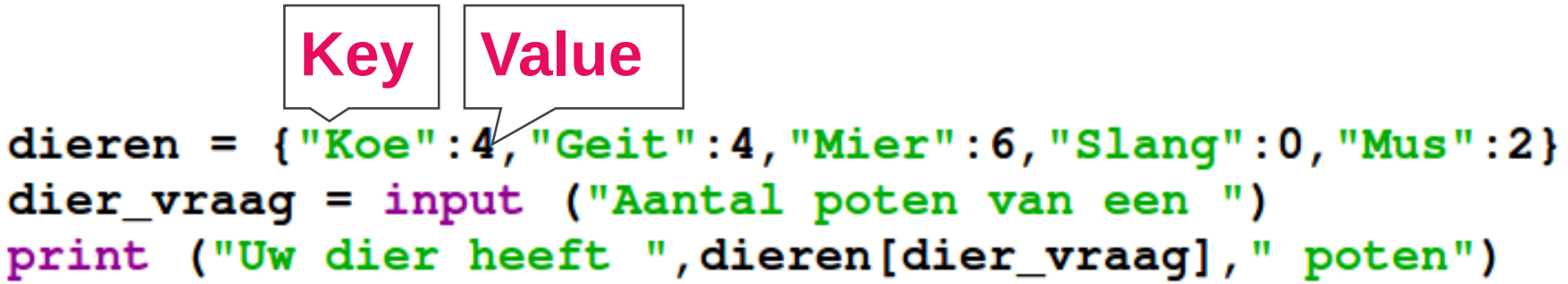
Hoe schrijven we de code om de vraag te beantwoorden hoeveel poten een koe heeft?

```
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*



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 heeft
- 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

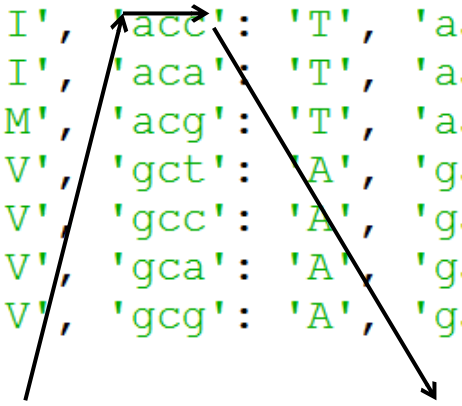
Al onze cellen hebben ook een dictionary…

```
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', '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'
       }
```

Eigen werk auteur

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

Eigen werk auteur

DICTIONARY

- Maar om alle codons op te halen voor een aminozuur is het juist erg lastig
- Daarvoor zou je een andere dictionary nodig hebben

DICTIONARY AMINOZUUR → CODON

```
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"]
       }
```

```
['UUA', 'UUG', 'CUU', 'CUC', 'CUA', 'CUG']
```

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

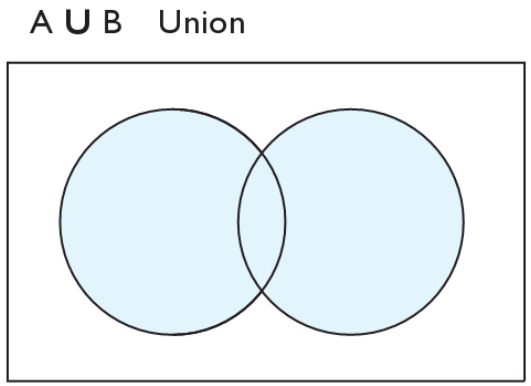
AGENDA

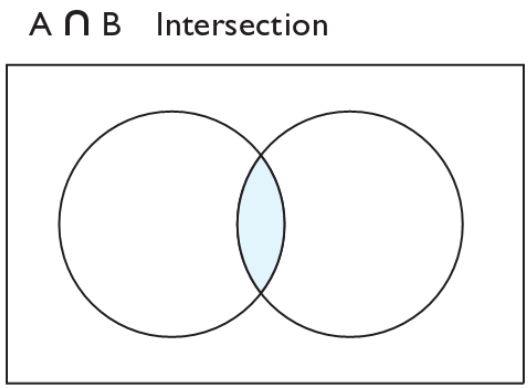
1. Datastructuren
2. Dictionaries
3. Sets
4. Serializing objects

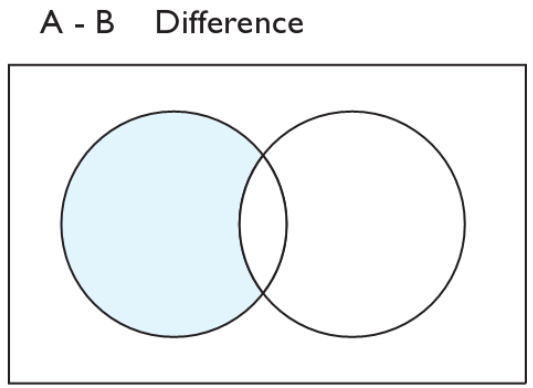
SETS

- We kunnen in Python met verzamelingen werken en met set operatoren bewerkingen op uitvoeren

Python	Betekenis
Union	Samenvoeging
Intersection	Overlap
Difference	Verschil







Eigen werk auteur

CREATIE VAN EEN SET

Een set maak je met het sleutelwoord set

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

Voor het toevoegen van waardes gebruik je add

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

SETS BESTAAN UIT UNIEKE WAARDES

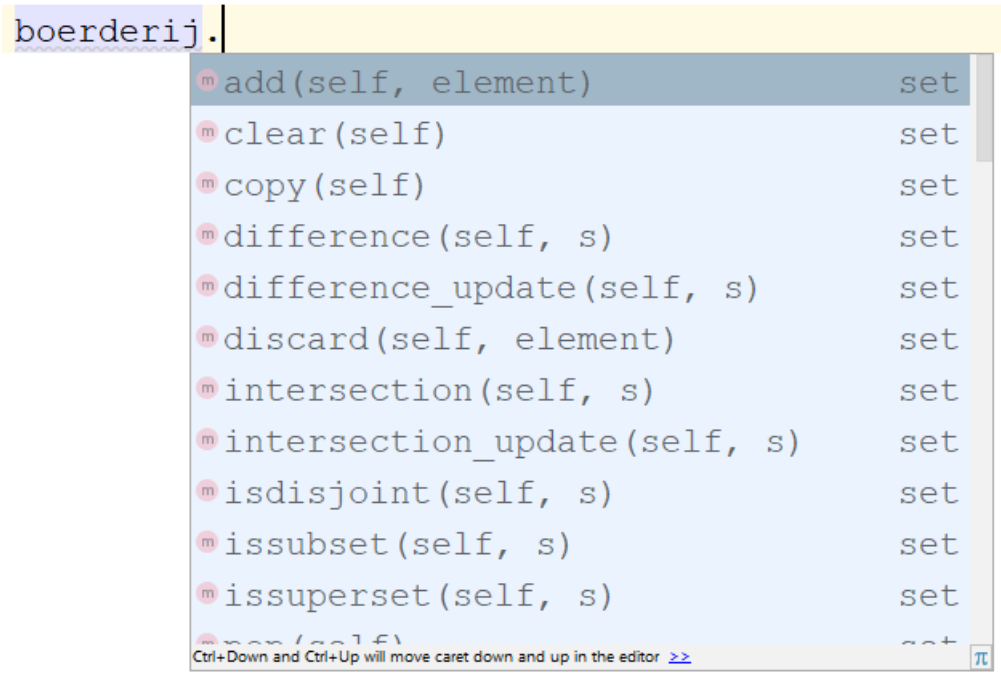
```
boerderij  = set(["Koe","Geit","Varken","Konijn","Konijn"])
dierentuin = set(["Olifant","Giraffe","Zebra", "Konijn","Geit"])
print (boerderij)
print (dierentuin)
```

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

Eigen werk auteur

SETS HEBBEN VEEL EIGEN METHODES

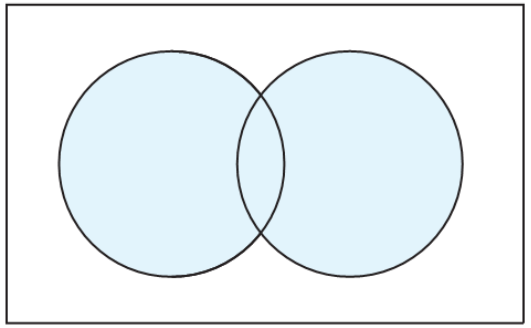
```
boerderij  = set(["Koe","Geit","Varken","Konijn","Konijn"])
dierentuin = set(["Olifant","Giraffe","Zebra", "Konijn","Geit"])
print (boerderij)
print (dierentuin)
```



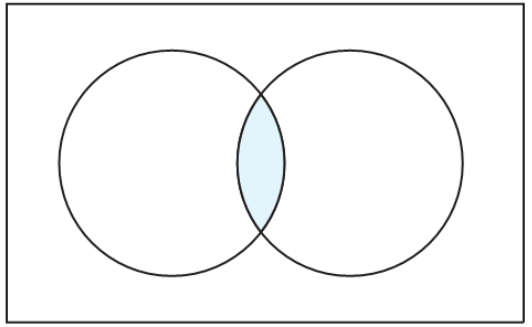
Eigen werk auteur

SET OPERATIONS

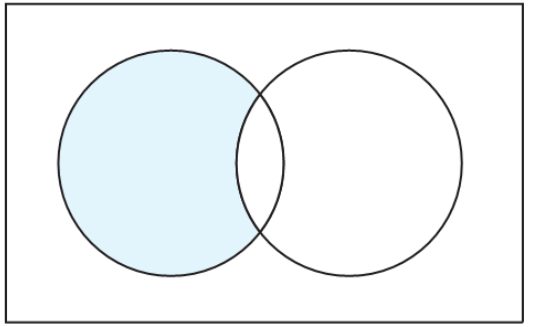
A ∪ B Union



A ∩ B Intersection



A - B Difference



Statement	Actie
s1.union(s2)	Retourneert een set alle items in s1 en s2
s1.intersection(s2)	Retourneert een set van items die zowel in s1 als s2 voorkomen
s1.difference(s2)	Retourneert een set van items in s1 die niet in s2 voorkomen

SETS EN OPERATOREN

Vul deze tabel in

	Union	Intersection	Difference
A: 12 5 17 6 B: 42 17 6	12 5 42 17 6	17 6	12 5
A: 21 76 10 3 9 B:	21 76 10 3 9		21 76 10 3 9
A: 87 B: 22 87 23	22 87 23	87	

VERSCHIL

Welke dieren komen wel voor op de boerderij maar niet in de dierentuin?

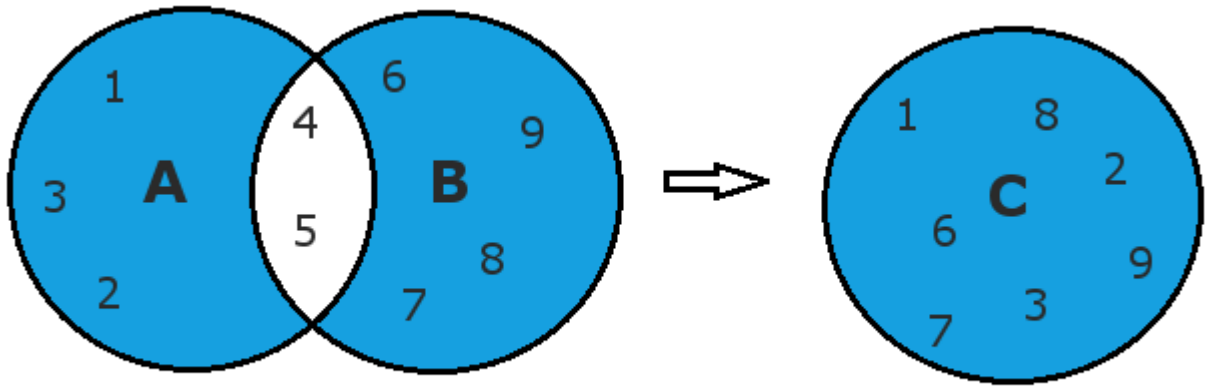
```
boerderij  = set(["Koe","Geit","Varken","Konijn","Konijn"])
dierentuin = set(["Olifant","Giraffe","Zebra", "Konijn","Geit"])

print (boerderij.difference(dierentuin))

{'Koe', 'Varken'}
```

Eigen werk auteur

SYMMETRIC DIFFERENCE



A Symmetric Difference B : C

SYMMETRIC DIFFERENCE

Bij een symmetric_difference verkrijg je van beide sets wat niet in de overlap zit

Statement	Actie
s1.difference(s2)	Retourneert een set van items in s1 die niet in s2 voorkomen
s1.symmetric_difference(s2)	Retourneert een set van items die niet in s1 en niet in s2 voorkomen

Hoe toon je de overeenkomst tussen een dataScientist en dataEngineer?

```
dataScientist = set(['Python', 'R', 'SQL', 'Git',
'Tableau', 'SAS’])
dataEngineer = set(['Python', 'Java', 'Scala', 'Git',
'SQL', 'Hadoop'])
```

```
>>>
dataScientist.intersection(dataEngineer)
{'SQL', 'Git', 'Python'}

>>>
dataEngineer.intersection(dataScientist)
{'SQL', 'Git', 'Python'}
```

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
>>>
```

Eigen werk auteur

SETS: SUBSET EN SUPERSET

Vul de volgende tabel in

	A issubset B	A issuperset B
A: 12 5 17 6 B: 42 17 6	False	False
A: 19 21 88 76 B: 21 76	False	True
A: 87 B: 22 87 23	True	False

KENMERKEN SETS

- Sets bevatten altijd unieke waardes
- Sets zijn ongeordend: de volgorde waarin waardes 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 waardes
- Ze zijn erg geschikt om verschillen en overeenkomsten tussen groepen te bepalen

AGENDA

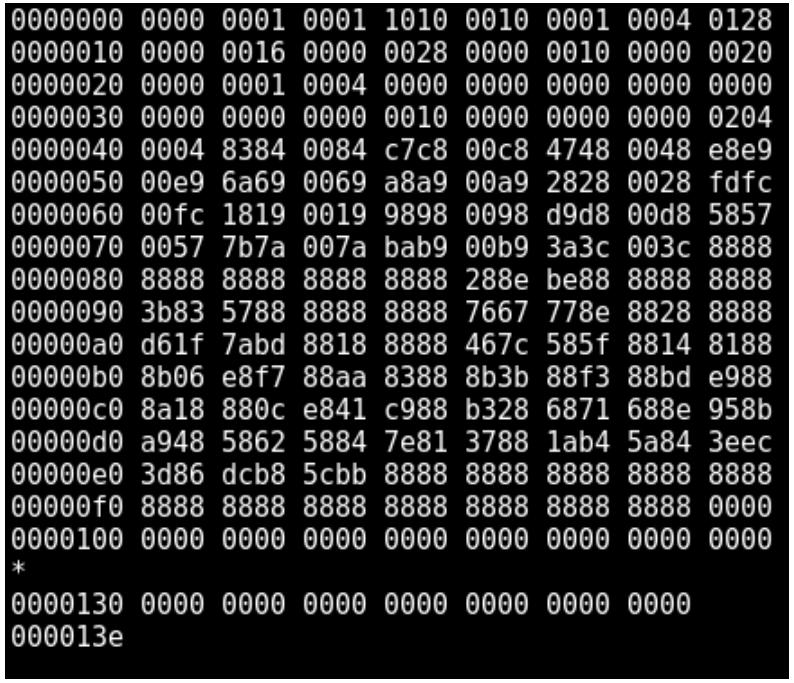
1. Datastructuren
2. Dictionaries
3. Sets
4. Serializing objects

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

WAT IS EEN BINARY FILE?

- Een binary file is een computer readable file ofwel niet human readable



https://en.wikipedia.org/wiki/Binary_file 22-feb-2017

WAT IS HET NUT VAN SERIALIZATION?

KENMERKEN SERIALIZATION

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

```python
import pickle


def bepaalBi(bestandsnaam):
    """Bepaal de bi1 klassen

    input:
    bestandsnaam, str, naam bestand

    return
    """
    meer = 'j'
    try:
        imput_file = open(bestandsnaam, 'rb')
        bi1 = pickle.load(input_file)
        input_file.close()
    except IOError:
        bi1 = set()
    while meer.lower()=="j":
        print("Studenten in Bi1: ", bi1)
        student = input("Student: ")
        bi1.add(student)
        meer = input("Doorgaan (j/n)")
    output_file = open(bestandsnaam, 'wb')
    pickle.dump(bi1, output_file)
    output_file.close()
    return


def main():
    bestandsnaam = "studenten.dat"
    bepaalBi(bestandsnaam)


main()
```

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

MET OPEN WITH

```
import pickle


def bepaalBi(bestandsnaam):
    """Bepaal de bi1 klassen

    input:
    bestandsnaam, str, naam bestand

    return
    """
    meer = 'j'
    try:
        with open(bestandsnaam, 'rb') as input_file:
            bi1 = pickle.load(input_file)
    except IOError:
        bi1 = set()
    while meer.lower()=="j":
        print("Studenten in Bi1: ", bi1)
        student = input("Student: ")
        bi1.add(student)
        meer = input("Doorgaan (j/n)")
    with open(bestandsnaam, 'wb') as output_file:
        pickle.dump(bi1, output_file)
    return


def main():
    bestandsnaam = "studenten.dat"
    bepaalBi(bestandsnaam)


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

APPENDIX

Tellen van codons en aminozuren

BEPAAL AANTAL AMINOZUREN

Bepaal het aantal aminozuren in een sequentie:

- Met dictionaries
- Zonder dictionaries

```
# Bepaal per aminozuur aantal in een sequentie
seq = "MTSSRLWFSLLLAAAFAGRATALWPWPQNFQTSDQRYVLYPNNFQFQYDV"
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
```

HAN_UNIVERSITY OF APPLIED SCIENCES

Eigen werk auteur

BEPAAL AANTAL AMINOZUREN

Met een dictionary is het veel eenvoudiger tellen

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

Eigen werk auteur

BEPAAL HET AANTAL CODONS

Maak een dictionary met het aantal als waarde en de codon als key. Gebruik hiervoor een willekeurige DNA sequentie

```
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}
```

Eigen werk auteur

VERANTWOORDING

- In deze uitgave is géén auteursrechtelijk beschermd werk opgenomen
- Alle teksten © Teuntje Peeters/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