StatsKurs_Uebung_Tag1

November 18, 2019

1 Einführung in die Statistik mit Python - Tag 1

1.1 Python Konventionen

Für Befehle über mehrere Zeilen, benutzt man das Zeichen um Python zu informieren, dass der Befehl auch die nächste Zeile betrifft.

```
In [1]: value = 1 + \
        2 + \
        3
```

Man kann mehrere Befehle auf eine Zeile schreiben, diese werden dann mit einem ; voneinander getrennt

```
In [2]: a = "One"; b = "Two"; c= "Three"
```

Um eine Zeichenkette (String) zu schreiben, verwendet man Anführungszeichen (') oder (''):

```
In [3]: str1 = 'Hello every body'
```

Für einen Zeichenkette über mehrere Zeilen, verwendet man 3 Anführungszeichen (") ohne das Zeichen zu benutzen

```
In [4]: multiLineStr = """This is a paragraph. It is
        made up of multiple lines and sentences."""
```

Kommentare beginnen immer mit einem #

```
In [5]: # First comment
        print("Hello, Python!") # second comment
        # This is a comment.
        # This is a comment, too.
        # This is a comment, too.
        print("Finish")
```

```
Hello, Python!
Finish
```

1.2 Python as Calculator

In [6]:
```
# Typische Mathematische Operationen

# Addition
2 + 3
# Subtraktion
3.3 - 2
# Multiplikation
2 * (3.141 - 2)
# Division
10 / 4
# Division ohne Rest
10 // 4
# Modulo bzw. Rest
10 % 3
# Potenz
5 ** 5
```

Out[6]: 3125

1.3 Hello World

In [7]:
```
print("hello")
print("world")

# Hello World
print("Hellow World.")

# Simple output
print("Hello, I'm Python!")

x = 'hello, python world!'
print(x)
print(x.split(' '))

# Input, assignment
# name = input('What is your name?\n')
# print('Hi, %s.' % name)
```

```
hello
world
Hellow World.
Hello, I'm Python!
hello, python world!
['hello,', 'python', 'world!']
```

1.4 Namensräume in Python

```
In [8]: # Import pi from Math package
        from math import pi

        # Print pi
        print(pi) # 3.14159

        # Define a variable, also called pi
        pi = 2

        # Print pi
        print(pi)
```

```
3.141592653589793
2
```

1.5 Zeichenketten festlegen

```
In [9]: # einfache Anführungszeichen
        x = 'Hallo Welt.'

        # doppelte Anführungszeichen
        y = "Hallo Welt."
        print(x ==y)

        quote1 = "Sie sagte 'Hallo' zu mir."
        quote2 = 'Sie sagte "Hallo" zu mir.'
        print(quote1 == quote2)

        # Eine mehrzeilige Zeichenkette
        z = """Diese Zeichenkette besteht aus mehreren Zeilen.
        Das hier ist die zweite Zeile,
        das ist die dritte."""
```

```
True
False
```

1.6 Methoden von Zeichenketten

```
In [10]: s = "Hallo Welt"

         s.lower()
         ## 'hallo welt'

         s.upper()
         ## 'HALLO WELT'
```

```
s.capitalize()
## "Hallo welt"

# Get help
# ?str.capitalize

# Oder
str.upper(s)

s.startswith("H")
## True

s.endswith("x")
## False

s.isdigit()
## False

str.isdigit("6")
## True

s.split(" ") ## ['Hallo', 'Welt']

"-".join(s.split(" ")) ## 'Hallo-Welt'
```

```
Out[10]: 'Hallo-Welt'
```

1.7 Indexierung von Zeichenketten

```
In [11]: n = "0123456789"

n[0]
## '0'
n[0:3] == n[:3]

## True
n[0:9]
## '012345678

n[0] == n[:]
## '0' ## '0123456789'

n[::2]
## True ## '02468'

n[1::2]
## '012345678' ## '13579'
```

```
n[::-1]
## '9876543210'
```

```
Out[11]: '9876543210'
```

1.8 Listen erstellen

```
In [12]: # leere Listen erstellen
         liste_leer1 = []
         liste_leer2 = list()
         liste_leer1 == liste_leer2

         # Listen erstellen mit unterschiedlichen Datentypen
         liste_inhalt = [1, 2.0, "drei", ["Hallo", "Welt"], {"key":"value"}]
         liste_inhalt

         # list()-Funktion auf Objekte anwenden
         list("Hello")
```

```
Out[12]: ['H', 'e', 'l', 'l', 'o']
```

1.9 Indexierung von Listen

```
In [13]: # Zuerst erstellen wir eine Liste
         liste = [9, 8, 7, 6, 5, 4]

         # Nun indexieren wir die Liste
         liste[1:3]
         liste[:2]
         liste[3:]
         liste[1::2]
         liste[-1::-1]
```

```
Out[13]: [4, 5, 6, 7, 8, 9]
```

1.10 Methoden von Listen

```
In [14]: liste1 = []
         liste2 = [4, 5]
         liste3 = [4, 3, 2, 5, 1]

         liste1.append(1) # Element anfügen
         liste1.extend(liste2) # Liste erweitern
         liste1.pop() # letztes Element herauslösen
         liste3.sort() # Liste sortieren
         liste3.remove(3) # spezifisches Element entfernen
```

1.11 Listen anwenden

```
In [15]: # List comprehensions
         fruits = ['Banana', 'Apple', 'Lime']
         loud_fruits = [fruit.upper() for fruit in fruits]
         print(loud_fruits)
         ## ['BANANA', 'APPLE', 'LIME']

         # List and the enumerate function
         list(enumerate(fruits))
         ## [(0, 'Banana'), (1, 'Apple'), (2, 'Lime')]
```

```
['BANANA', 'APPLE', 'LIME']
```

```
Out[15]: [(0, 'Banana'), (1, 'Apple'), (2, 'Lime')]
```

1.12 For-Schleife durch eine Liste

```
In [16]: numbers = [2, 4, 6, 8]
         product = 1

         for number in numbers:
           product = product * number
           print('The product is:', product)

         ## ('The product is:', 384)
```

```
The product is: 2
The product is: 8
The product is: 48
The product is: 384
```

1.13 Listen entschachteln

```
In [17]: import itertools

         liste = [[1,2,3], [4, 5, 6]]
         chain1 = list(itertools.chain(*liste))
         chain2 = list(itertools.chain.from_iterable(liste))

         print(chain1 == chain2)
         ## True

         print(chain1)
         ## [1, 2, 3, 4, 5, 6]
```

```
True
[1, 2, 3, 4, 5, 6]
```

1.14 Ein Dictionary-Objekt erstellen

```
In [18]: # Dictionary mit geschweiften Klammern instanziieren
         d = {"key1": "val1", "key2": "val2"}

         # dict indexing
         print(d["key2"])

         # Werte ändern
         d["key2"] = "val2_neu"
         print(d["key2"])

         # Dictionary mit dict-Funktion
         dict(hallo="welt")
         #dict(1="eins")
```

```
val2
val2_neu
```

```
Out[18]: {'hallo': 'welt'}
```

1.15 Ein Dictionary sortieren

```
In [19]: d = {3: "drei", 1: "eins", 2: "zwei"}

         d_sorted = {k:v for (k,v) in sorted(d.items(), key=lambda x: x[0])}
         print(d_sorted)

         ## {1: 'eins', 2: 'zwei', 3: 'drei'}

         # Einfacher Datenabgleich mithilfe von Key-Value-Paaren
         de_en = {"Hallo": "hello", "Welt": "world", "du": "you",
                  "bist": "are", "schön": "beautiful",}

         trans = (de_en.get("Hallo"), de_en.get("Welt"), de_en.get("du"),
                  de_en.get("bist"), de_en.get("schön"),)

         print("{} {}, {} {} {}!".format(*trans)) # tuple unpacking

         ## hello world, you are beautiful!

         # default Parameter ändern
         de_en.get("groartig", "Wort nicht vorhanden")
         ## 'Wort nicht vorhanden'
```

```
{1: 'eins', 2: 'zwei', 3: 'drei'}
hello world, you are beautiful!
```

```
Out[19]: 'Wort nicht vorhanden'
```

1.16 Einem Dictionary neue Werte hinzufügen

```
In [20]: # Key-Value-Paar hinzufügen per Zuweisung
         de_en["groartig"] = "awesome"

         # Key-Value-Paar hinzufügen per update()-Methode
         de_en.update({"groartig": "awesome"})
         de_en.get("groartig", "Wort nicht vorhanden")

         ## 'awesome'
```

```
Out[20]: 'awesome'
```

1.17 Wert-Mapping

Text in Zahlen umwandeln:

```
In [21]: antworten = ["gut", "ok", "schlecht", "sehr gut",
                      "sehr schlecht", "wei nicht", "so lala",]
         mapping = {"sehr schlecht": -2, "schlecht": -1, "ok": 0,
                    "gut": 1, "sehr gut": 2, "wei nicht": 8,}

         numerisch = [mapping.get(antwort, 99) for antwort in antworten]
```

Mapping erstellen aus zwei Listen:

```
In [22]: labels = ["negativ", "neutral", "positiv"]
         number = [-1, 0, 1]
         mapping = {n: l for (n, l) in zip(number, labels)}
```

1.18 Tuple

```
In [23]: t = ("Hallo", "du", "schöne", "Welt")

         # tuple unpacking
         x, *y, z = t

         print(x); print(y)

         # list unpacking
         liste = ["Hallo", "du", "schöne", "Welt"]
         a, *b, c = liste

         a == x
```

```
Hallo
['du', 'schöne']
```

```
Out[23]: True
```

1.19 Sets und ausgewählte Methoden

```
In [24]: s1 = {1, 2, 3, 4}
         liste = [1, 3, 5, 7, 7, 3, 1]
         s2 = set(liste)

         s1.union(s2) ## set([1, 2, 3, 4, 5, 7])
         s2.union(s1) ## set([1, 2, 3, 4, 5, 7])

         s1.intersection(s2) ## set([1, 3])
         s2.intersection(s1) ## set([1, 3])

         s1.difference(s2) ## set([2, 4])
         s2.difference(s1) ## set([5, 7])

         s1.add(8)
         s1

Out[24]: {1, 2, 3, 4, 8}
```

Frozensets are unmutable:

```
In [25]: # Thus adding attributes does not work
         #fset = frozenset(liste)
         #fset.add(8)
```

1.20 None, True, False

```
In [26]: False == 0

         True == 1

         None == False

         not None == True

         #if True:
         #  do_this(stuff)
         #  else:
         #    do_that(stuff)

         x = []
         if x:
           print("Die Liste ist gefüllt.")
         else:
           print("Die Liste ist leer.")

Die Liste ist leer.
```

In [27]:
```
# Vergleiche vornehmen

antwort = [1, 2, 3, 2, 4, 1, 8]

if antwort and not 8 in antwort:
  print("Die Antworten sind vollständig")
else:
  print("Antworten unvollständig.")
## Antworten unvollständig.
```

```
Antworten unvollständig.
```

1.21 Prüfung auf Gleichheit und Identität von Werten

In [28]:
```
# x == y # x gleich y

# x != y # x ungleich y

# x > y # x gröer y

# x < y # x kleiner y

# x < y < z # x kleiner y, y kleiner z

# x <= y # x kleiner oder gleich y

# x >= y # x gröer oder gleich y

# x & y, x and y # x und y -> bitwise, logical

# x | y, x or y # x oder y -> bitwise, logical
```

In [29]:
```
x = 4
y = 4.0

# Prüfung auf Gleichheit / Gleichwertigkeit
x == y # True

# Prüfung auf Selbigkeit / Identität
x is y # False

x = 4
y = 4.0
z = x

print(id(x), id(y), id(z))
```

```
94857246319744 140456190789936 94857246319744
```

1.22 Unterschied zwischen Referenz und Kopie

```
In [30]: # Referenz -> dieselbe ID
         x = [1, 2, 3]
         y = x
         y.pop()
         print(x)

         # Kopie -> unterschiedliche ID
         x = [1, 2, 3]
         y = x.copy()
         y.pop()
         print(x, y)
```

```
[1, 2]
[1, 2, 3] [1, 2]
```

1.23 Abfragen mit while-Schleifen

```
In [31]: i = 1
         while i < 4:
           print(i, end=" ")
           i += 1 # kurz für: i = i + 1
```

```
1 2 3
```

```
In [32]: ##allgemeines Funktionsprinzip einer while-Schleife

         #process = True
         #while process:
         #  if some_condition:
         #    # ... rufe die Funktion auf.
         #    do_something_with(stuff)
         #else:
         #  process = False
```

1.24 for-Schleifen schreiben

```
In [33]: names = ["Markus", "Julia", "Klaus"]

         n = len(names) # n = 3

         # als Schleife mit Index-Wert
         for i in range(n):
           name = names[i] # names[0] == "Markus"
```

```
    print(name)

# auf Elemente direkt zugreifen
for names in names:
    print(names)
```

```
Markus
Julia
Klaus
Markus
Julia
Klaus
```

1.25 List-Comprehensions

```
In [34]: # Liste mit for-loop erstellen
         numbers_for = []

         for i in range(10): numbers_for.append(i)

         # Beispiel einer einfachen List-Comprehension
         numbers_comp = [i for i in range(10)]
         numbers_for == numbers_comp

         # Eine Liste mit Listen entschachteln
         list_of_lists = [[0, 1, 2], [3, 4, 5], [6, 7, 8]]

         # als for-loop
         single_numbers = []
         for num_list in list_of_lists:
             for num in num_list:
                 single_numbers.append(num)

         # als list comp
         single_numbers = [num for num_list in list_of_lists for num in num_list]
```

1.26 List Comprehensions mit if-Bedingung

```
In [35]: # als for-loop
         even = []
         for i in range(10):
             if i % 2 == 0:
                 even.append(i)

         # als list comp
         even = [i for i in range(10) if i % 2 == 0]
         print(even)
```

```
[0, 2, 4, 6, 8]
```

1.27 Dictionary Comprehensions

```
In [36]: numbers = [1, 2, 3]
         labels = ["eins", "zwei", "drei"]
         d = {k:v for (k, v) in zip(numbers, labels)}
```

1.28 Fehlerbehebung: Try und Except

```
In [37]: #nummern = [1, 2, "3.14", "Hallo. Wie geht's?", 4.0]

         ## Code ohne try und except Schlagwörtern
         #for num in nummern:
         #  if type(num) == str:
         #    if "." in num:
         #      num = float(num)
         #    else:
         #      num = int(num)
         #      print(num * 3)

         ## ValueError: could not convert string to float: 'Hallo. Wie geht es dir?'
```

```
In [38]: nummern = [1, 2, "3.14", "Hallo. Wie geht's?", 4.0]

         # Code mit try und except Schlagwörtern
         for num in nummern:
           try:
             if type(num) == str:
               if "." in num:
                 num = float(num)
               else:
                 num = int(num)
             print(num * 3)
           except ValueError as e:
             print(e)
```

```
3
6
9.42
could not convert string to float: "Hallo. Wie geht's?"
12.0
```

1.29 Eigene Funktionen definieren

```
In [39]: # Funktion definieren
         def hello(name):
```

```
    return (f"Hallo, {name}! Wie geht es dir?")

# Funktion aufrufen
hello("John")
```

```
Out[39]: 'Hallo, John! Wie geht es dir?'
```

```
In [40]: ## Funktion zur Berechnung des Quadrats
         def squared(x):
           return x**2

         for ii in range(6):
           print(ii, squared(ii))
```

```
0 0
1 1
2 4
3 9
4 16
5 25
```

```
In [41]: # Funktion mit Keyword-Parameter
         def hello(name, anrede="dir"):
           return f"Hallo, {name}! Wie geht es {anrede}?"

         print(hello("James", "ihnen"))

         print(hello("ihnen", "James"))
```

```
Hallo, James! Wie geht es ihnen?
Hallo, ihnen! Wie geht es James?
```

1.30 Daten einlesen und exportieren

```
In [42]: ## Einlesen allgemein
         # pd.read_<format>(fpath, **params)

         ## Einlesen einer Datei mit Tabulator getrennten Werten
         # data = pd.read_csv("my_tsv_data.csv", sep="\t")

         ## Export allgemein
         # df.to_<format>(fpath, **params)

         ## Export einer Datei mit Pipe getrennten Werten
         # df.to_csv(fpath, sep="|")
```

1.31 Beispiel-Datensatz generieren

In [43]:
```
# Libraries importieren
import numpy as np
import pandas as pd

# Seed festlegen, zur Reproduktion des Codes
np.random.seed(42)

# Daten und Index generieren
data = np.random.random((10, 4))
names = ["f1", "f2", "f3", "f4"]
idx = range(4, 14)

# DataFrame erstellen
df = pd.DataFrame(data, columns=names, index=idx)

# Attribute: Zeilen, Spalten und Werte
df.index
df.columns
df.values

# Indexierung einer Spalte
df["f1"] == df.f1
#out: True
```

Out[43]:
```
4     True
5     True
6     True
7     True
8     True
9     True
10    True
11    True
12    True
13    True
Name: f1, dtype: bool
```

1.32 Einem DataFrame neue Spalten hinzufügen

In [44]:
```
# neue Werte hinzufügen
np.random.seed(42)
cluster_values = np.random.randint(1, 4, 10)
df["cluster"] = cluster_values

mapping = {
    1: "Sport",
    2: "Wirtschaft",
    3: "Kultur",
```

```
}
df["label"] = df.cluster.map(mapping)
```

1.33 Werte durch Berechnungen hinzufügen und löschen

```
In [45]: # neue Werte durch Berechnungen hinzufügen
         df["sum_f1_f2"] = df.f1 + df.f2
         df["mean_f1_f2"] = (df.f1 + df.f2) / 2

         # neue Werte durch for-Loop hinzufügen
         for column in df.columns[0:4]:
             name = f"{column}_mn_diff"
             df[name] = df[column] - df[column].mean()

         # Spalten löschen
         df.drop("sum_f1_f2", axis=1, inplace=True)
```

1.34 Auf Werte eines DataFrame zugreifen

```
In [46]: # Auf Einzelwerte zugreifen: [Zeile, Spalte]
         df.at[4, "f1"]
         df.iat[4, 0]
         df.at[4, "f1"] == df.iat[0, 0]

         # Einzelwerte festlegen
         df.at[4, "f1"] = 1

         # Slicing mit Namen
         df.loc[5:9, ["f1", "f3"]]

         # Indexierung mit Position
         df.iloc[1:6, [0, 2]]

         # einfaches boolean Indexing
         df[df.f1 > 0.4]
         df[df.label == "Kultur"]

         # komplexes boolean Indexing
         df[(df.label == "Kultur") | (df.cluster == 2)]
         df[(df.label == "Kultur") & (df.f3 >= 0.3)]
```

```
Out[46]:          f1        f2        f3        f4  cluster   label  mean_f1_f2  \
         4   1.000000  0.950714  0.731994  0.598658        3  Kultur    0.662627
         12  0.065052  0.948886  0.965632  0.808397        3  Kultur    0.506969
         13  0.304614  0.097672  0.684233  0.440152        3  Kultur    0.201143

             f1_mn_diff  f2_mn_diff  f3_mn_diff  f4_mn_diff
         4    -0.055296    0.493759    0.314628    0.077754
```

```
12   -0.364785    0.491930    0.548266    0.287493
13   -0.125222   -0.359283    0.266867   -0.080752
```

1.35 Werte eines DataFrame sortieren

```
In [47]: df.sort_index(inplace=True)
         df.sort_values("f1", ascending=False, inplace=True)
         df.sort_values(["f2", "f1"], ascending=False, inplace=True)
```

1.36 Überblick über die Daten beschaffen

```
In [48]: df.dtypes
         df.head(2)
         df.tail(7)
         df.shape

Out[48]: (10, 11)
```

1.37 Daten mit eigenen Funktionen bearbeiten

```
In [49]: def multi_x(x, *args):
           for arg in args:
             x *= arg
           return int(x)

         # xi mit den args-Werten multiplizieren [i: 0 ... n]
         df.f1.apply(multi_x, args=(3, 2, 4))

         # 0, wenn xi kleiner oder gleich 0.5, sonst 1 [i: 0 ... n]
         df.f2.apply(lambda x: 0 if x <= 0.5 else 1)

Out[49]: 4     1
         12    1
         10    1
         6     1
         8     1
         7     0
         5     0
         9     0
         13    0
         11    0
         Name: f2, dtype: int64
```

1.38 Do it yourself

Liest den Datensatz **size.csv** in Python ein

```
In [50]: import pandas as pd
         size = pd.read_csv("/home/matt/Documents/Github/pyStats/data/size.csv")
```

Verschafft euch einen Überblick über die Daten (dtypes, head, tail)

```
In [51]: # Look at data type
         size.dtypes

         # Look at head
         size.head(2)
         pd.DataFrame.head(size)
         size.tail()

         # Look at shape
         size.shape
```

```
Out[51]: (30, 4)
```

Sortiert alle Daten nach GröSSe

```
In [52]: size.sort_values(by="groesse_cm")

         # Von klein zu gro
         size = size.sort_values("groesse_cm")
         # Alternative
         size.sort_values("groesse_cm", inplace=True)

         # Von gro zu klein
         size.sort_values("groesse_cm", ascending=False, inplace=True)
         size.head()
```

```
Out[52]:    geschlecht  groesse_cm  schuhgroesse  teilnehmer
         7           m         200            41           0
         19          m         198            40           0
         6           m         195            43           0
         18          m         194            46           0
         28          m         190            46           1
```

Wie kann man den Datensatz nach zwei Variablen sortieren?

```
In [53]: # Hierzu schaut man am Besten in der Hilfe
         # ?size.sort_values

         # Und kommt dann zu folgendem Ergebniss
         size.sort_values(by=["geschlecht", "schuhgroesse"], inplace=True)
         size.head()
```

```
Out[53]:    geschlecht  groesse_cm  schuhgroesse  teilnehmer
         19          m         198            40           0
         7           m         200            41           0
         2           m         180            41           0
         23          m         179            42           0
         12          m         179            42           0
```

Extrahiert alle Einträge mit einer Groesse über 180 cm

```
In [54]: size[size.groesse_cm > 180]
```

```
Out[54]:    geschlecht  groesse_cm  schuhgroesse  teilnehmer
         19          m         198            40           0
         7           m         200            41           0
         6           m         195            43           0
         0           m         189            43           0
         24          m         189            43           1
         11          m         184            43           0
         25          m         189            45           1
         17          m         188            45           0
         18          m         194            46           0
         28          m         190            46           1
         5           m         190            46           0
         20          w         187            36           0
         9           w         190            37           0
         8           w         185            39           0
         13          w         183            39           0
         21          w         184            40           0
```

Nun extrahiert alle Einträge für weibliche Personen mit einer Groesse von 180 cm und mehr:

```
In [55]: size[(size.geschlecht == "w") & (size.groesse_cm >= 180)]
```

```
Out[55]:    geschlecht  groesse_cm  schuhgroesse  teilnehmer
         20          w         187            36           0
         9           w         190            37           0
         8           w         185            39           0
         13          w         183            39           0
         21          w         184            40           0
```

und für alle männlichen Personen mit einer Groesse unter 190 cm und einer Schuhgroesse von 43:

```
In [56]: size[(size.geschlecht == "m") & (size.groesse_cm < 190) & (size.schuhgroesse == 43)]
```

```
Out[56]:    geschlecht  groesse_cm  schuhgroesse  teilnehmer
         0           m         189            43           0
         24          m         189            43           1
         11          m         184            43           0
         14          m         178            43           0
```

Aller Männer unter 180 cm oder Männer mit SchuhgröSSe 40

```
In [57]: size[(size.geschlecht == "m") & (size.groesse_cm < 180) | (size.geschlecht == "m") &

         # Oder so
         size[(size.geschlecht == "m") & ((size.groesse_cm < 180) | (size.schuhgroesse == 40))]
```

```
Out[57]:    geschlecht  groesse_cm  schuhgroesse  teilnehmer
         19          m         198            40           0
         23          m         179            42           0
         12          m         179            42           0
         26          m         173            42           1
         14          m         178            43           0
```

Extrahiert alle Einträge für weibliche Personen mit einer Groesse von 190 cm oder einer Schuhgroesse von 40.

```
In [58]: size[(size.geschlecht == "w") & ((size.groesse_cm == 190) | (size.schuhgroesse == 40))]
```

```
Out[58]:    geschlecht  groesse_cm  schuhgroesse  teilnehmer
         9           w         190            37           0
         21          w         184            40           0
         10          w         168            40           0
         1           w         160            40           0
```

Alle Männer unter 180 und alle Frauen mit Schuhgroesse 40

```
In [59]: size[((size.geschlecht == "m") & (size.groesse_cm < 180)) | ((size.geschlecht == "w") &
```

```
Out[59]:    geschlecht  groesse_cm  schuhgroesse  teilnehmer
         23          m         179            42           0
         12          m         179            42           0
         26          m         173            42           1
         14          m         178            43           0
         21          w         184            40           0
         10          w         168            40           0
         1           w         160            40           0
```