Creative Data Mining

uncover and evaluate

06.05.2017

Lecture 3

Dr. Daniel Zünd
Danielle Griego
Artem Chirkin

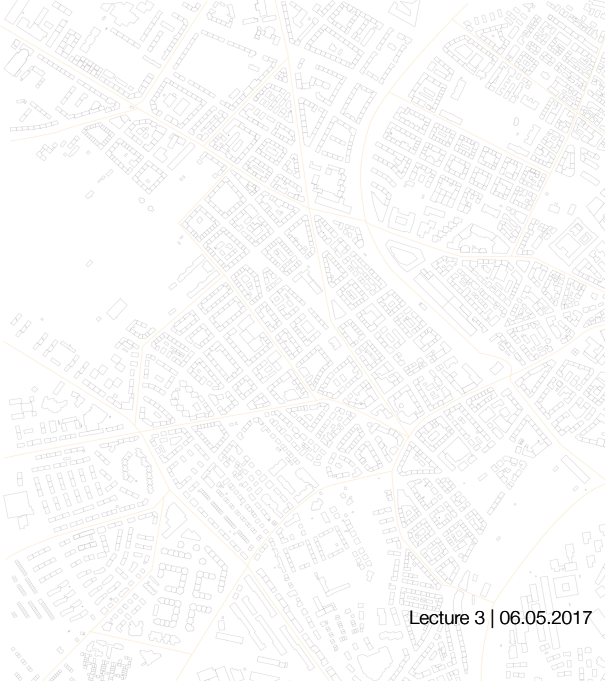
ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

DARCH

iA Chair of Information Architecture

What we'll cover today

– Recap of last week
– Functions
– Modules
– Reading & Writing Files



Programming?

Programming is about solving problems and puzzles. You precisely describe a set of instructions which the computer strictly follows.

For example sorting: (9,4,10,6) → (4,6,9,10)

Types and Values

Values are for example: *1*, *2.0*, *True* or *"Hello world!"*

Each value is of a certain **type**. The numbers are **numerics**, *True* is a **boolean**, and *"Hello world!"* is a **string**.

Types and Values

Booleans

Booleans only have two values or states. They are either *True* or *False*.

Types and Values

Numerics

Numerics are numbers. There are two main types, **Integer** or **Float** numbers.

Integers are **whole numbers**.

Examples: -1, 42, 23, 12

Floating point numbers are **real numbers**.

Examples: 2.7182, -3.1415, 3.0

Types and Values

Strings

Strings are lists of characters.

Examples: ”Hello world!”, ”0”, ”bar”

Operators

Operators represent a value manipulation. It is **important** that you check that the data type is correct, otherwise unexpected behavior might occur.

Example: 4/5 vs. 4.0/5.0

Data Structures

There are three basic data structures in Python.

– Lists
– Tuples
– Dictionaries

Data Structures

Lists

A **list** is a sequence of values of any type.

Example: foo = [42, [1,2,3], ”Hello world!”, 23]

Lists provide several functionalities.

```
foo[2]          # Accesses the third(!) value.
foo.remove(42)  # Removes the first occurence of 42.
foo.append(42)  # Adds 42 to the end of the list.
…
```

Data Structures

Tuples

A **tuple** is a sequence of values of any type, but can not be altered after initialisation.

Example:
foo = (42, [1,2,3], ”Hello world!”, 23)
vs.
foo = [42, [1,2,3], ”Hello world!”, 23]

Data Structures

Dictionaries

Dictionaries store the data as key-value pairs. Each key has an associated value.

Example:
foo = {”answer”:42, ”who?”:”brown fox”}
vs.
foo = (42, [1,2,3], ”Hello world!”, 23)
vs.
foo = [42, [1,2,3], ”Hello world!”, 23]

Values can be accessed and set with the keys.

foo[”answer”] # Returns 42.
foo[42] = ”answer” # Adds a new element with the key 42 and the value ”answer”.

Logical Operators

Logical Operators are expressions that evaluate to either *True* or *False*.
They are normally used in **conditional statements**.
Relational operators: ==, !=, <, >, <=, >=, and, or, not
They stand for: equal, not equal, smaller, greater, smaller equal, greater equal, and, or, not

Examples:
x < 2 or x >= 0
x == 0 or x != 0
not (x == 0 or x > 100)

Conditional statements

Depending on a conditional statement, you can execute different parts of the code.

Example:

```
if aValue < 2:
        print(”Hello world!”)
elif aValue == 2:
        print(”Hello again!”)
else:
        print(”Goodbye”)
```

Loops

Loops make it possible to repeat certain sets of instructions. There are two different kinds of loops: *for* and *while* loops.

Loops

for loops iterate through a list and make it possible to do something with each element.

Example:

```
for a in [1, 32, 4, 2]:
    b = a * a
    print(b)
```

while loops repeat a certain set of instructions, until a condition is not met anymore.

Example:

```
a = 1
while a < 16:
    a = a + a
    print(a)
```

Loops

useful keywords

– **break**: terminates the loop in any case.
– **continue**: terminates the current iteration and continues the loop.

Putting it all together

```
for a in [0,1,2,3,4,5,6,7,8,9]:
    if a < 3:
        print(a * 100)
    elif a == 5:
        continue
    elif a > 7:
        break
    else:
        print(a/2)
```

Putting it all together

```
for a in [0,1,2,3,4,5,6,7,8,9]:
    if a < 3:
        print(a * 100)
    elif a == 5:
        continue
    elif a > 7:
        break
    else:
        print(a/2)
```

Output:

```
0
100
200
1
2
3
3
```

Indentation

Indentation is **important**!

```
for a in [1,3]:
    a = a * a
    print(a)
```

Output:
1
9

```
for a in [1,3]:
    a = a * a
print(a)
```

Output:
9

Functions

If you do not want to retype the same code over and over again, you can define a **function** that you can call over and over again.

def functionName*(*parameters*):*
    instructions
    return variable

Example:

```
def square(a):
    a = a * a
    return a
```

To get the square of 3, for example, you can now call *square(3)*.

Functions

def functionName*(*parameters*):*
    instructions
    return variable

Also multiple parameters are possible.

Example:

```
def square(a):
    a = a * a
    return a
```

To get the square of 3, for example, you can now call *square(3)*.

Example:

```
def multiply(a, b):
    result = a * b
    return result
```

To multiply 3 with 4, for example, you can now call *multiply(3,4)*.

Modules

Often, more complicated code is already implemented and can be used as **modules**. They help to be more efficient, because the programmer does not need to implement everything from scratch.

When installed, **modules** can be integrated to the code using the keyword *import*. To import, for example, the *Pandas* module, write *import Pandas* before you want to use it.

Working with Files

There exist many modules that can process many different data formats. All the data used in this course has the CSV format.

Example CSV file content:

Name, Age
Hans, 23
Jack, 42
Jean, 37
Juan, 8

CSV Files

Reading file

age.csv content:

```
Name, Age
Hans, 23
Jack, 42
Jean, 37
Juan, 8
```

```
import csv
names = []
ages = []
with open('age.csv', 'r') as f:
    content = csv.reader(f)
    firstRow = True
    for row in content:
        if firstRow:
            firstRow = False
            continue
        names.append(row[0])
        ages.append(int(row[1]))
print(names)
print(ages)
```

CSV Files

Writing file

```
import csv
names = ['Hans', 'Jack', 'Jean', 'Juan']
ages = [23, 42, 37, 8]
with open('agePlus1.csv', 'w') as f:
    fileWriter = csv.writer(f)
    fileWriter.writerow(['Name','Age'])
    for i in range(len(names)):
        currRow = [names[i], ages[i]+1]
        fileWriter.writerow(currRow)
```

```
import csv
names = []
ages = []
with open('age.csv', 'r') as f:
    content = csv.reader(f)
    firstRow = True
    for row in content:
        if firstRow:
            firstRow = False
            continue
        names.append(row[0])
        ages.append(int(row[1]))
print(names)
print(ages)
```

Hint for the Exercise

Modulo (%) is a useful operator when you want to test for even or odd numbers. It calculates the rest of a division.

Examples: *5%2* returns 1
6%2 returns 0
13%5 returns 3