Lecture 2

SE1 – Moderne Programmierkonzepte

Prof. Dr. Maximilian Scherer

maximilian.scherer@dhbw-mannheim.de

DHBW
Duale Hochschule
Baden-Württemberg
Mannheim

SoSe 2022

Nachbesprechung

- Programmiersprachen

Nachbesprechung

- Programmiersprachen
- Programmierparadigmen

Nachbesprechung

- Programmiersprachen
- Programmierparadigmen
- Funktionale vs objektorientierte Programmierung

Nachbesprechung

- Programmiersprachen
- Programmierparadigmen
- Funktionale vs objektorientierte Programmierung
- Events / Lambda Ausdrücke

Mini-Projekt Themen

- Gruppen 2-4 Studierende

Mini-Projekt Themen

- Gruppen 2-4 Studierende
- Fokus auf 1-2 Vorlesungsinhalte

Mini-Projekt Themen

- Gruppen 2-4 Studierende
- Fokus auf 1-2 Vorlesungsinhalte
- Im One-Pager darstellen

Mini-Projekt Themen

- Gruppen 2-4 Studierende
- Fokus auf 1-2 Vorlesungsinhalte
- Im One-Pager darstellen
- Vorlesungsbezug (Darstellung und Umsetzung) bei Bewertung hohes Gewicht

Mini-Projekt Themen

- Gruppen 2-4 Studierende
- Fokus auf 1-2 Vorlesungsinhalte
- Im One-Pager darstellen
- Vorlesungsbezug (Darstellung und Umsetzung) bei Bewertung hohes Gewicht
- Eigene Ideen (kurze Absprache) gerne möglich

Mini-Projekt Themen

- Gruppen 2-4 Studierende
- Fokus auf 1-2 Vorlesungsinhalte
- Im One-Pager darstellen
- Vorlesungsbezug (Darstellung und Umsetzung) bei Bewertung hohes Gewicht
- Eigene Ideen (kurze Absprache) gerne möglich
- ideenauswahl.txt

Slides modified from “Python for JAVA Developers”

`https://github.com/akashp1712/awesome-python-cheatsheets` by Akash Panchal

Python for JAVA Developers: Basics V 1.2

Referring python 3.7.1

1 Basic Syntax

1.1 End of Statements

Unlike the Java, to end a statement in Python, we don't have to type in a semicolon, you simply press $\boxed{Enter}$. But semicolons can be used to delimit statements if you wish to put multiple statements on the same line.

```
message1 = 'Hello World!'
message2 = "Python gives no missing semicolon error!"


# Instead of System.out.print, we use print
```

```
print (message1) # print 'Hello World!' on the console output
print ("Hello"); print ("Python!"); # usage of the semicolon
```

1.2 Code Blocks and Indentation

One of the most distinctive features of Python is its use of indentation to mark blocks of code. Consider the following if-statement from our non-zero number checking program:

JAVA

```
if (0 == value) {
   System.out.print("Number is Zero");
} else {
   System.out.print("Number is non-Zero.");
}

System.out.print("All done!");
```

Python

```
if 0 == value:
    print('Number is Zero')
else:
    print('Number is non-Zero.')

print('All done!')
```

To indicate a block of code in Python, you must indent each line of the block by the same amount. The two blocks of code in our example if-statement are both indented **four spaces**, which is a typical amount of indentation for Python.

2 VARIABLES

2.1 Declaration

Variables are created the first time a value is assigned to them. There is no concept of the declaration of the data type in python.

```
number = 11
```

```
string = "This is a string"
```

You declare multiple variables by separating each variable name with a comma.

```
a, b = True, False
```

2.2 Assigning Values

```
a = 300
```

The same value can be assigned to multiple variables at the same time:

```
a = b = c = 1
```

And multiple variables can be assigned different values on a single line:

```
a, b, c = 1, 2, "john"
```

This is the same as:

```
a = 1
b = 2
```

```
c = "john"
```

3 Data Types

Python sets the variable type based on the value that is assigned to it. Unlike JAVA, Python will change the variable type if the variable value is set to another value.

```
var = 123 # This will create a number integer assignment
var = 'john' # the `var` variable is now a string type.
```

3.1 Numbers

Most of the time using the standard Python number type is fine. Python will automatically convert a number from one type to another whenever required. We don't require to use the type casting like JAVA.

Type	Java	Python	Description
int	int a = 11	a = 11	Signed Integer
long	long a = 1712L	a = 1712L	(L) Long integers
float	float a = 19.91	a = 19.91	(.) Floating point values
complex	- - -	a = 3.14J	(J) integer [0 to 255]

3.2 String

Create string variables by enclosing characters in quotes. Python uses single quotes $\boxed{\text{′}}$ double quotes $\boxed{\text{”}}$ and triple quotes $\boxed{\text{””””}}$ to denote literal strings. Only the triple quoted strings $\boxed{\text{””””}}$ will automatically continue across the end of line statement.

```
firstName = ’Jane’
lastName = "Doe"
message = """This is a string that will span across multiple lines. Using
  newline characters and no spaces for the next lines."""
```

Key Methods:

Java	Python
charAt()	find()
indexOf()	index()
length()	len()
replace()	replace()
toString()	str()
trim()	rstrip(), lstrip()

Python String comparison can be performed using equality (==) and comparison (<, >, !=, <=, >=) operators. There are no special methods to compare two strings.

3.3 Boolean

Python provides the boolean type that can be either set to False or True. In python, every object has a boolean value. The following elements are false:

- None

- False
- 0
- Empty collections: “”, (), [],

All other objects are True. **Note** that, In JAVA **null** is not false while in python None is.

3.4 List

The List is one of the most powerful variable type (data structure!) in Python. A list can contain a series of values. Unlike JAVA, the list need not be always homogeneous. A single list can contain strings, integers, as well as objects. Lists are mutable.

List variables are declared by using brackets $\boxed{[]}$ following the variable name.

```
A = [] # This is a blank list variable.
B = [1, 23, 45, 67] # creates an initial list of 4 numbers.
C = [2, 4, 'john'] # can contain different variable types.
D = ["A", B, C] # can contains other list objects as well.
```

Key Methods:

Java ArrayList	Python list
list = new ArrayList()	list=[]
list.add(object)	list.append(object)
list.get(i)	list[i]
list.size()	len(list)
list2= list.clone()	list2=list[:]

3.5 Tuple

A tuple is another useful variable type similar to list and can contain heterogeneous values. Unlike the list, a tuple is **immutable** And is like a static array.

A tuple is **fixed in size** and that is why tuples are replacing array completely as they are more efficient in all parameters.

Tuple can be used as an alternative of **list(python/Java)** OR **Array(Java)** with respect to the use cases. i.e, If you have a dataset which will be assigned only once and its value should not change again, you need a tuple.

Tuple variables are declared by using parentheses $\boxed{()}$ following the variable name.

```
A = () # This is a blank tuple variable.
B = (1, 23, 45, 67) # creates a tuple of 4 numbers.
C = (2, 4, 'john') # can contain different variable types.
D = ("A", B, C) # can contains other tuple objects as well.
```

3.6 Dictionary

Dictionaries in Python are lists of **Key: Value** pairs. This is a very powerful datatype to hold a lot of related information that can be associated through keys. Dictionary in Python can be the Alternative of the Map in JAVA. But again a Dictionary in python can contain **heterogeneous** key as well as value.

```
room_num = {'john': 121, 'tom': 307}
room_num['john'] = 432 # set the value associated with the 'john' key to 432
print (room_num['tom']) # print the value of the 'tom' key.
room_num['isaac'] = 345 # Add a new key 'isaac' with the associated value
```

```
print (room_num.keys()) # print out a list of keys in the dictionary
print ('isaac' in room_num) # test to see if 'issac' is in the dictionary.
  This returns true.

hotel_name = {1: "Taj", "two": "Woodland", "next": 3.14} # this is totally
  valid in python
```

Key Methods:

Java HashMap	Python Dictionary
Map myMap = new HashMap()	my_dict = { }
clear()	clear()
clone()	copy()
containsKey(key)	key **in** my_dict
get(key)	get(key)
keySet()	keys()
put(key, value)	my_dict[key] = value
remove(key)	pop(key)
size()	len(my_dict)

4 OPERATORS

4.0.1 Operator Precedence

Operator Precedence is same as that of JAVA, let's revise it in python. Arithmetic operators are evaluated first, comparison operators are evaluated next, and logical operators are evaluated last.

Arithmetic operators are evaluated in the following order of precedence.

JAVA	Python	Description
NOT AVAILABLE	**	Exponentiation
-	-	Unary negation
*	*	Multiplication
/	/	Division
%	%	Modulus arithmetic
+	+	Addition
-	-	Subtraction

Logical operators are evaluated in the following order of precedence.

JAVA	**Python**	**Description**
!	not	Logical negation
&	&	Logical conjunction
|	|	Logical dis-junction
∧	∧	Logical exclusion
&&	and	Conditional AND
||	or	Conditional OR

Comparison operators all have equal precedence; that is, they are evaluated in the left-to-right order in which they appear.

JAVA	Python	Description
<	<	Less than
>	>	Greater than
<=	<=	Less than or equal to
>=	>=	Greater than or equal to
==	==	Equality
!=	!=	Inequality
equals	is	Object equivalence

Note: == in python is actually .equals() of Java.

5 Conditionals

5.1 if

```
var1 = 250
if 250 == var1:
   print ("The value of the variable is 250")
```

5.2 if..else

```
var1 = 250
if 0 == var1:
   MyLayerColor = 'vbRed'
   MyObjectColor = 'vbBlue'
else :
   MyLayerColor = 'vbGreen'
   MyObjectColor = 'vbBlack'

print (MyLayerColor)
```

5.3 if..elif..elif..else

```
var1 = 0
```

```
if 0 <= var1:
   print ("This is the first " + str(var1))
elif 1 == var1:
   print ("This is the second " + str(var1))
elif 2 >= var1:
   print ("This is the third " + str(var1))
else:
   print ("Value out of range!")
```

5.4 Multiple conditions

```
skill1 = "java"
skill2 = "python"

if skill1 == "java" and skill2 == "python":
   print ("Both the condition satisfy")

if skill1 == "java" or skill2 == "python":
    print ("At least One condition satisfies")
```

6 Looping

6.1 For Loop

Python will automatically increments the counter (x) variable by 1 after coming to end of the execution block.

```
for x in range(0, 5):
    print ("It's a loop: " + str(x))
```

for x in range(10)

Increase or decrease the counter variable by the value you specify.

```
# the counter variable j is incremented by 2 each time the loop repeats
for j in range(0, 10, 2):
    print ("We're on loop " + str(j))

# the counter variable j is decreased by 2 each time the loop repeats
for j in range(10, 0, -2):
    print ("We're on loop " + str(j))
```

You can exit any for loop before the counter reaches its end value by using the $\boxed{break}$ statement.

6.2 While Loop

Simple example of while loop.

```
var1 = 3
while var1 < 37:
    var1 = var1 * 2
    print (var1)
print ("Exited while loop.")
```

Another example of while loop with break statement.

```
while True:
    n = raw_input("Please enter 'hello':")
    if n.strip() == 'hello':
```

```
    break
```

7 Iterations

7.1 Iterating a List

```
friends = ['Huey', 'Dewey', 'Louie']
for friend in friends:
   print ('Hello ', friend)
print ('Done!')
```

7.2 Iterating a Tuple

```
tup = ('alpha', 'beta', 'omega')
for val in tup:
   print (val)
```

7.3 Iterating a Dictionary

```
codes = {'INDIA': 'in', 'USA': 'us', 'UK': 'gb'}
for key in codes:
   print (key, 'corresponds to', codes[key])

# Note: key is just a variable name.
```

The above will simply loop over the keys in the dictionary, rather than the keys and values.

To loop over both key and value you can use the following:

```
codes = {'INDIA': 'in', 'USA': 'us', 'UK': 'gb'}
for key, value in codes.iteritems():
   print (key, 'corresponds to', value)

# Note: key, value are just the variable names.
```

Packages

- import random

Packages

- import random
- import numpy as np

Packages

- import random
- import numpy as np
- from os import listdir

Live-Coding

Python Basics

`https://www.jdoodle.com/python3-programming-online/`

Übungsaufgabe

Fibonacci Generator

- `https://www.youtube.com/watch?v=SjSHVDfXHQ4` The magic of Fibonacci numbers | Arthur Benjamin SjSHVDfXHQ4

Übungsaufgabe

Fibonacci Generator

- `https://www.youtube.com/watch?v=SjSHVDfXHQ4` The magic of Fibonacci numbers | Arthur Benjamin SjSHVDfXHQ4
- Eingabe (Konsole): Natürliche Zahl N

Übungsaufgabe

Fibonacci Generator

- `https://www.youtube.com/watch?v=SjSHVDfXHQ4` The magic of Fibonacci numbers | Arthur Benjamin SjSHVDfXHQ4
- Eingabe (Konsole): Natürliche Zahl N
- Ausgabe (Konsole): Alle Fibonaccizahlen kleiner als N

Übungsaufgabe

Fibonacci Generator

- `https://www.youtube.com/watch?v=SjSHVDfXHQ4` The magic of Fibonacci numbers | Arthur Benjamin SjSHVDfXHQ4
- Eingabe (Konsole): Natürliche Zahl N
- Ausgabe (Konsole): Alle Fibonaccizahlen kleiner als N
- Fibonacci-Sequenz $f_n = f_{n-1} + f_{n-2}$ mit $f_0 = f_1 = 1$

Übungsaufgabe

Fibonacci Generator

- `https://www.youtube.com/watch?v=SjSHVDfXHQ4` The magic of Fibonacci numbers | Arthur Benjamin SjSHVDfXHQ4
- Eingabe (Konsole): Natürliche Zahl N
- Ausgabe (Konsole): Alle Fibonaccizahlen kleiner als N
- Fibonacci-Sequenz $f_n = f_{n-1} + f_{n-2}$ mit $f_0 = f_1 = 1$
- In Java und Python implementieren

Programmierparadigmen

- Imperative (structured, procedural)

Programmierparadigmen

- Imperative (structured, procedural)
- Object-oriented

Programmierparadigmen

- Imperative (structured, procedural)
- Object-oriented
- Declarative / Functional

Programmierparadigmen

- Imperative (structured, procedural)
- Object-oriented
- Declarative / Functional
- Event-driven

Programmierparadigmen

- Imperative (structured, procedural)
- Object-oriented
- Declarative / Functional
- Event-driven
- Data-driven

Data-driven Programming

- Keine Werte hardcoden

Data-driven Programming

- Keine Werte hardcoden
- Alle Werte in “Dateien” auslagern

Data-driven Programming

- Keine Werte hardcoden
- Alle Werte in “Dateien” auslagern
    - .ini Konfigurationsdateien

Data-driven Programming

- Keine Werte hardcoden
- Alle Werte in “Dateien” auslagern
    - .ini Konfigurationsdateien
    - csv / xlsx Tabellen (Flach)

Data-driven Programming

- Keine Werte hardcoden
- Alle Werte in “Dateien” auslagern
  - .ini Konfigurationsdateien
  - csv / xlsx Tabellen (Flach)
  - xml, json, yaml, toml

Data-driven Programming

- Keine Werte hardcoden
- Alle Werte in “Dateien” auslagern
  - .ini Konfigurationsdateien
  - csv / xlsx Tabellen (Flach)
  - xml, json, yaml, toml
  - sql / nosql Datenbanken

Data-driven Programming

- Kein kontext-spezifisches Verhalten hardcoden

Data-driven Programming

- Kein kontext-spezifisches Verhalten hardcoden
- Insbesondere im game-development / modding sehr relevant

Data-driven Programming

- Kein kontext-spezifisches Verhalten hardcoden
- Insbesondere im game-development / modding sehr relevant
- Scripting in Dateien auslagern

Data-driven Programming

- Kein kontext-spezifisches Verhalten hardcoden
- Insbesondere im game-development / modding sehr relevant
- Scripting in Dateien auslagern
  - eigene Scriptsprache

Data-driven Programming

- Kein kontext-spezifisches Verhalten hardcoden
- Insbesondere im game-development / modding sehr relevant
- Scripting in Dateien auslagern
  - eigene Scriptsprache
  - lua / js / python

Data-driven Programming

- Kein kontext-spezifisches Verhalten hardcoden
- Insbesondere im game-development / modding sehr relevant
- Scripting in Dateien auslagern
  - eigene Scriptsprache
  - lua / js / python
  - JIT compiling (C#)

Data-driven Programming

- Kein kontext-spezifisches Verhalten hardcoden
- Insbesondere im game-development / modding sehr relevant
- Scripting in Dateien auslagern
  - eigene Scriptsprache
  - lua / js / python
  - JIT compiling (C#)
- Achtung: Sicherheitsrisiko / Sandboxing benötigt

Live Coding

Programmier-Paradigmen