Grundlagen der Programmierung für Nicht-Informatiker

Tag 1

Giuseppe Accaputo

g@accaputo.ch

Schön seid ihr heute hier

Über mich

- Ausbildung
  - B.Sc. & M.Sc. ETH in Rechnergestützte Wissenschaften (2011 – 2017)
  - B.Sc. FH in Informatik mit Vertiefung in Software Engineering (2006 – 2009)
  - Berufslehre als Informatiker EFZ Fachrichtung Systemtechnik (2002 – 2006)
- Arbeit
  - Software Entwickler, Nexiot AG (seit April 2018) [Groovy, Java, Kotlin]
  - Wissenschaftlicher Mitarbeiter, ETH Zürich (2017 – 2018) [Bash, C, C++, MATLAB, Python]
  - Übungs- und Kursleiter, ETH Zürich (2014 – 2017) [C++, Java, MATLAB, Pascal]
  - Nachhilfelehrer, selbstständig (2016 – 2018) [C++, Java, R]
  - Software Entwickler, LTV Gelbe Seiten AG (2009 – 2011) [C#, JavaScript]

Wer seid ihr?

- Studiengang / Beschäftigung
- Programmiererfahrung:
  - 0: Keine Erfahrung
  - 1: Wenig Erfahrung
  - 2: Wesentliche Erfahrung
- Ziele für den Kurs

Aufbau Kurs

- Kurs besteht aus mehreren Lerneinheiten
- Pro Lerneinheit:
  - Definition der Lernziele
  - Inhalt der Lerneinheit
  - Passende Übungen
  - Live Coding
  - Kontrolle Lernziele

Fragen während und nach dem Kurs

- Fragen stellen erwünscht und jederzeit erlaubt
- Ihr dürft mich auch sehr gerne nach dem Kurs kontaktieren, falls ihr zu irgendwelchen Themen fragen habt: g@accaputo.ch

Feedback

- Erste Durchführung des Kurses
- Feedback ist natürlich sehr willkommen

Gebäude

- Gebäude wenn möglich nur nach Absprache verlassen (wegen Zutritt)
- Telefonnr. Kursraum: 0446356776

Inhaltsverzeichnis – Gesamter Kurs

Kurs – Lernziele

- Nach diesem Kurs…
  - … kennt ihr einige fundamentale Grundlagen der Programmierung
  - … kennt ihr die wichtigsten Bausteine der Python Programmiersprache
  - … könnt ihr Python Code ausführen
  - … könnt ihr einfache Aufgaben in ein Python Programm abbilden und ausführen

Inhaltsverzeichnis – Heutiger Kurstag

Universität Zürich[UZH]

Zentrale Informatik – IT Fort- und Weiterbildungen

Grundlagen der Programmierung

Lernziele

– Nach dieser Einheit wisst ihr...

1. … was ein Programm ist
2. … warum wir Programmiersprachen benötigen
3. … warum wir uns für Python entschieden haben

Aufgabe 1

[Aufgabe]

1. Zahl 1 hat Wert 2
2. Zahl 2 hat Wert 8
3. Zahl 3 hat Wert 4
4. Gib (Zahl 1 * Zahl 2) + Zahl 3 aus

Aufgabe 2

[Aufgabe]

1. Master-Passwort lautet "test"
2. Benutzer gibt Passwort "tesst" ein
3. Falls eingegebenes Passwort mit Master-Passwort übereinstimmt:
    1. Gib "Login erfolgreich" aus
4. Sonst:
    1. Gib "Falsches Passwort" aus

Aufgabe 3

[Aufgabe]

1. Liste 1 besteht aus den Elementen 1,2,3,4,5, und 6
2. Für jedes Element in Liste 1:
    1. Gib Element auf einer eigenen Zeile aus

Aufgabe 4

[Aufgabe]

1. Liste 2 besteht aus den Elementen 3,5,8, und 1
2. Gib die Summe bestehend aus dem ersten und dritten Element der Liste 2 aus

Aufgabe 5

[Aufgabe]

1. Die Liste von Teams, die an der WM teilnehmen besteht aus den Elementen "Schweiz", "Brasilien", und "Deutschland"
2. Falls "Italien" in der Liste vorkommt:
    1. Gib "Juppi, Italien nimmt an der WM teil!" aus
3. Sonst:
    1. Gib "Damn it, das wird ein langweiliger Sommer für uns Italiener!" aus

Aufgabe 5

[Aufgabe]

1. Verfügbare Kontostände:
    1. "Giusi": 300.-
    2. "Marco": 1000.-
2. Kontobenutzer ist "Giusi "
3. Abzuhebender Betrag ist 150.-
4. Falls Kontobenutzer genug Geld auf Konto hat:
    1. Gib dem Kontobenutzer den Betrag in Noten raus
    2. Zieh den Betrag vom Kontostand des Kontobenutzers ab
5. Sonst:
    1. Gib "Leider ist Kontostand zu niedrig"

Was ist ein Programm?

- Ein Programm ist eine Folge von Anweisungen, um bestimmte Aufgaben oder Probleme mithilfe eines Computers zu bearbeiten oder zu lösen
- Ein Rezept von Befehlen um dem Computer mitzuteilen was dieser in welcher Reihenfolge machen soll

Was ist ein Programm?

- *Eingabe*: Daten kommen von der Tastatur, einer Datei, dem Internet, etc.
- *Ausgabe*: Daten werden auf dem Bildschirm angezeigt, in eine Datei geschrieben, etc.
- *Mathematische Anweisungen*: Führe Addition, Subtraktion, Multiplikation, etc. aus
- *Bedingte Anweisungen*: Überprüfe gewisse Bedingungen und führe basierend darauf spezifische Anweisungen aus
- *Repetition*: Wiederhole gewisse Anweisungen

- Beispielprogramme aus dem Alltag: E-Mail Programm, Microsoft Word

Beispielprogramm von vorhin

1. Zahl 1 hat Wert 2
2. Zahl 2 hat Wert 8
3. Zahl 3 hat Wert 4
4. Gib (Zahl 1 * Zahl 2) + Zahl 3 aus

Wie können wir dem Computer Anweisungen geben?

- Wir haben nun eine konkrete Idee für ein Programm
- Million Dollar Question:
  Wie können wir (in einer *Sprache*, die wir verstehen) dem Computer nun verständlich mitteilen (in einer *Sprache*, die er versteht) welche Anweisungen er ausführen soll?

- Entwurf eines möglichen Programms:

1. Zahl 1 hat Wert 2
2. Zahl 2 hat Wert 8
3. Zahl 3 hat Wert 4
4. Gib (Zahl 1 * Zahl 2) + Zahl 3 aus

Die Programmiersprache

- *Software Entwickler (wir)* schreiben Programme
- Ein Programm beginnt mit dem *Code*, der eine Reihe von Anweisungen für den Computer enthält
- Diese Anweisungen werden mit Hilfe einer Programmiersprache geschrieben
  - Eine formale Sprache zur Formulierung von Anweisungen, die von einem Computer ausgeführt werden können

Die Maschinensprache

– Computer können nur Anweisungen ausführen, welche in Maschinensprache geschrieben sind

– Nächste Million Dollar Question:
Wie können wir nun anhand einer Programmiersprache dem Computer verständlich mitteilen – also in Maschinensprache –, welche Anweisungen er ausführen soll?

Dem Computer erfolgreich Anweisungen geben

Schritt 1: Anweisungen in Code erfassen

- Code enthält Anweisungen, welche in der gewünschten Programmiersprache von einem Menschen (im Weiteren *Entwickler* genannt) geschrieben sind
  - Ziel: Anweisungen sollen am Ende dieser Prozedur vom Computer ausgeführt werden
- Wird mittels Texteditor oder Entwicklungsumgebung geschrieben und in einer Textdatei abgespeichert

Schritt 2: Code vom Interpreter übersetzen lassen

- Der Interpreter – oder auch *Übersetzer* genannt – übersetzt den Code in einen maschinennahen Code, welcher vom Computer verstanden wird
  - Der maschinennahe Code ist in einer *Maschinensprache* geschrieben und enthält für den Computer verständliche Anweisungen
- Interpreter ist «Schnittstelle» zwischen Entwickler und Computer

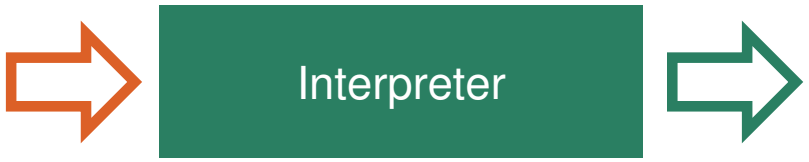
Schritt 3: Die Anweisungen vom Computer ausführen

- Nachdem ein Code erfolgreich zu Maschinencode übersetzt wurde, werden die Anweisungen vom Computer ausgeführt
  - Gängige Programmiersprachen sind äusserst *portabel*, d.h. Code, welcher mit solchen Programmiersprachen erstellt wurde kann auf verschiedenen Betriebssystemen (Windows, Mac OS X, und Linux) übersetzt und ausgeführt werden

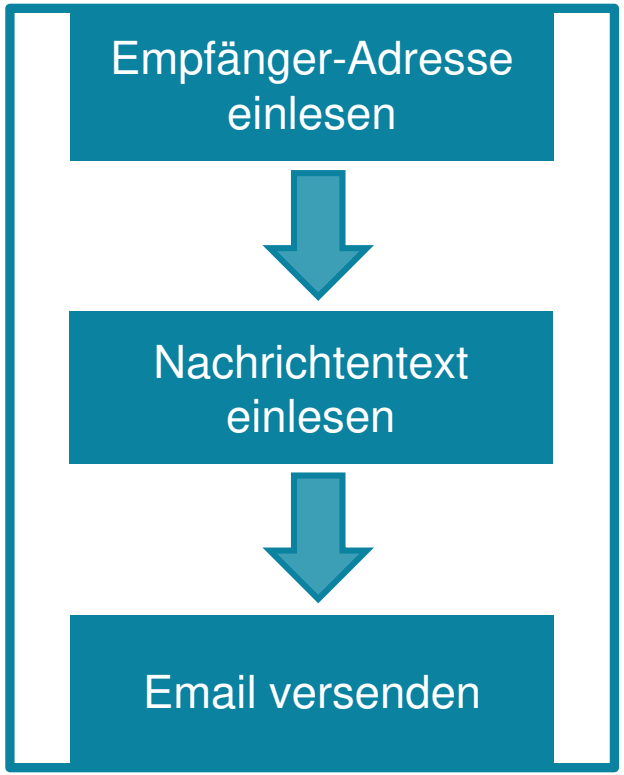
Beispielverlauf

Code (Entwickler):

1. Lies Empfänger-Adresse ein
2. Lies Nachrichtentext ein
3. Versende Email



Ausführung (Computer):



Warum Python?

– Idee für ein Programm:

IDEE

1. Zahl 1 hat Wert 2
2. Zahl 2 hat Wert 8
3. Zahl 3 hat Wert 4
4. Gib (Zahl 1 * Zahl 2) + Zahl 3 aus

– Programm in Python umgesetzt:

CODE

```
zahl_1 = 2
zahl_2 = 8
zahl_3 = 4
print (zahl_1 * zahl_2) + zahl_3
```

Warum Python?

- Einfache Sprache
  - Sehr angenehm bei ersten Schritten in der Programmierung
- Hochsprache
  - Wir müssen uns nicht um «gewisse Details» kümmern; Sprache nimmt uns sehr viel ab (schränkt auch ein, ist aber in unserer Situation überhaupt nicht schlimm)
- Interpretierte Sprache
- Portierbar auf den gängigsten Betriebssystemen
  - Code kann auf Windows, Mac OS X, und Linux interpretiert und ausgeführt werden
- Umfangreiche Bibliotheken
  - Bibliotheken die numerische Methoden anbieten, Graphen generieren können, spezielle Dateien einlesen können, Statistiken berechnen, etc.

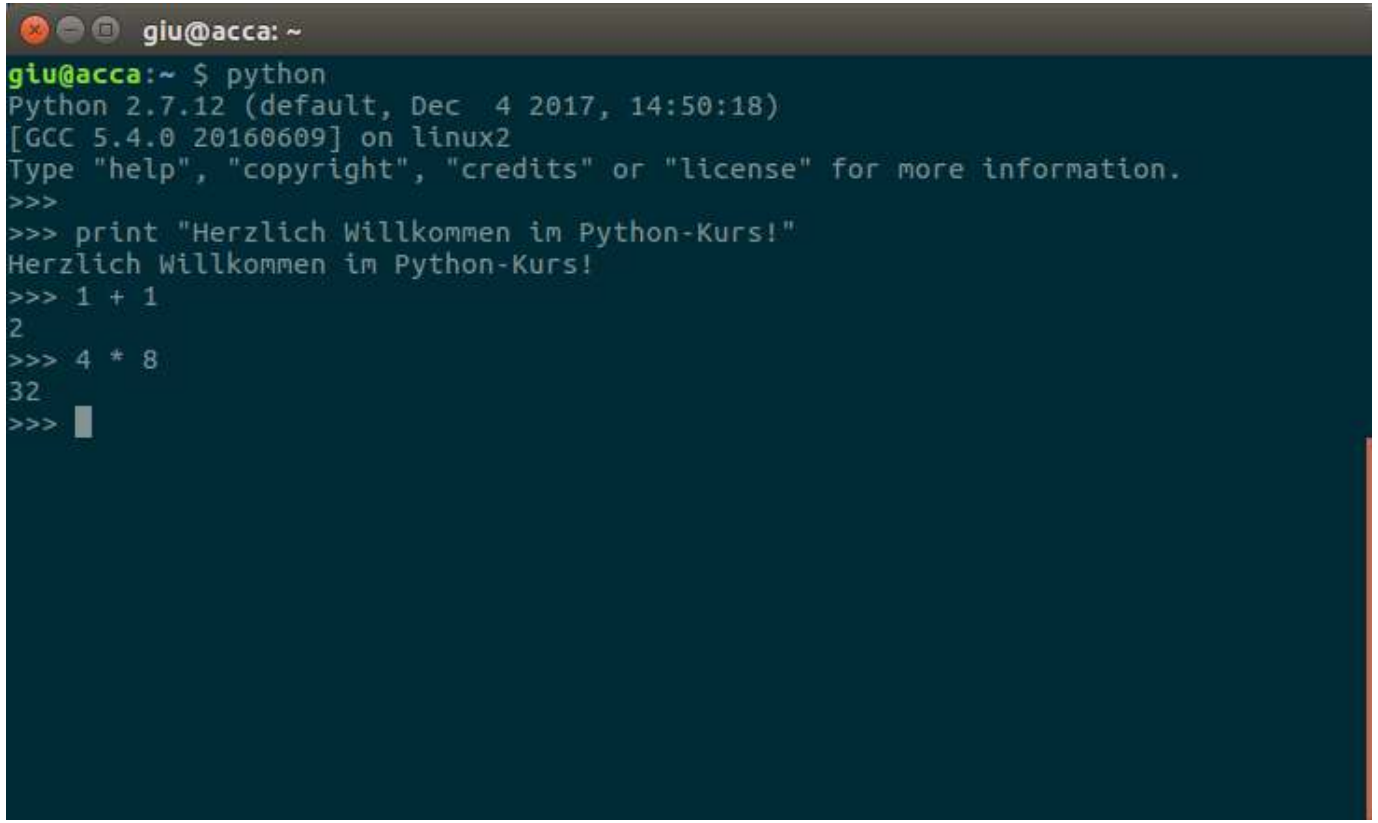
Interaktiver Modus

- Der Python-Interpreter unterstützt einen *interaktiven Modus*
- Anweisungen können direkt am Terminal eingeben werden; die Ergebnisse kann man sofort betrachten

INTERPRETER

```
>>> print 'Ich bin der Interpreter'
Ich bin der Interpreter
```

Interaktive Benutzung



Wie erlernt man das Programmieren?

- «Learning by doing»
  - Viel Programmieren, ausprobieren und in Fehlersituation geraten
    - Fehlersituationen versuchen zu verstehen
  - Grosser Bestandteil des Kurses

Unser erstes Programm – Hallo, integrierte Entwicklungsumgebung!

- Integrierte Entwicklungsumgebung (IE):
  Sammlung von Tools um Softwareentwicklung angenehmer zu gestalten
- PyCharm: IE um Python Programme zu entwickeln
  - Code wird mit Hilfe von PyCharm geschrieben, übersetzt und gleich ausgeführt ohne
  - Ohne IE: Tool um Code zu schreiben, Terminal um Code zu übersetzen und auszuführen

- PyCharm starten:
  Finder öffnen (Apfeltaste + Leertaste drücken) und PyCharm eingeben, mit Enter bestätigen

«Hallo Welt!»

– Unser erstes Programm:

CODE

```
print 'Hallo Welt!'
```

«Hallo Welt!» in anderen Programmiersprachen

– Java

CODE

```
public class HalloWelt{
  public static void main(String args[]){
    System.out.println('Hallo Welt!');
  }
}
```

– Kompilieren: `javac HalloWelt.java`

«Hallo Welt!» in anderen Programmiersprachen

– C++

CODE

```
#include <iostream>
int main(){
  std::cout << 'Hallo Welt!' << std::endl;
  return 0;
}
```

– Kompilieren: `g++ HalloWelt.cpp -o HalloWelt`

Lernziele – Check

– Nach dieser Einheit wisst ihr...
  1. … was ein Programm ist
  2. … warum wir Programmiersprachen benötigen
  3. … warum wir uns für Python entschieden haben

Universität Zürich[UZH]

Zentrale Informatik – IT Fort- und Weiterbildungen

Variablen, Anweisungen, Ausdrücke, und alles dazwischen

Lernziele

– Nach dieser Einheit wissen wir:
  1. … was eine Variable ist
  2. … wie man einer Variable einen Wert zuweist
  3. … wie man eine oder mehrere Variablen ausgibt
  4. … was *Strings*, *Floats*, und *Ints* sind
  5. … wieso wir Typumwandlungen benötigen

Werte und Typen

Werte

- Fundamentale Sache wie z.B. ein Buchstabe oder eine Zahl
  - Beispiel eines Wertes: `2`
  - Weiteres Beispiel eines Wertes: `'Python'`
  - Noch ein Beispiel eines Wertes: `3.14159`

Datentypen

- Ein Wert gehört immer zu einem bestimmten Datentyp
  - Der Wert `2` ist eine *ganze Zahl*
  - Der Wert `'Python'` ist eine *Zeichenkette*
  - Der Wert `3.14159` ist eine *Fliesskommazahl*

Datentypen

Datentyp in Python	Beispiele
Integer (ganze Zahl)	`-2, 1, 0, 1, 2, 3, 4, 5`
Float (Fliesskommazahl)	`-4.5592, -1.0, 0.421, 1.4234, 3.14`
Strings (Zeichenketten)	`'hello', 'Giuseppe', 'Python', '3 Musketiere'`

Datentyp herausfinden

– Mit der Hilfe von `type(wert)` können wir herausfinden, zu welchem Datentyp ein bestimmter Wert gehört

CODE

```
print type('Hello, there!')
print type(3.14)
print type(9000)
print type('200.4123')
```

INTERPRETER

PYCHARM

```
<type 'str'>
<type 'float'>
<type 'int'>
<type 'str'>
```

Variablen

Variablen

- Eine *Variable* ist wie eine beschriftete Box, in welcher wir einen Wert verstauen (*speichern*) können
  - Eine Variable hat also einen *Namen* und einen *Wert*
- Ein Wert kann man mittels dem = Operator in eine Variable speichern / einer Variable zuweisen
  - Beispiel: `variablen_name = 'Ein möglicher Wert'`
- Eine Variable hat auch einen Typ, welchen wir mittels `type(variablen_namen)` herausfinden können

CODE

```
answer = 42
mein_name = 'Giuseppe'

print answer
print mein_name
print type(mein_name)
```

INTERPRETER

PYCHARM

```
42
Giuseppe
<type 'str'>
```

Regeln für Variablennamen

– Regeln für Variablennamen:

1. Variablenname ist ein einzelnes Wort (keine Leerzeichen)
2. Variablenname darf nur Buchstaben, Zahlen und Underscore (_) enthalten
3. Variablenname darf nicht mit einer Zahl beginnen
4. Variablennamen sind «case-sensitive».
   Beispiel: `ein_wert` und `ein_Wert` sind zwei verschiedene Variablen

Gültige Variablennamen	Ungültige Variablennamen
`Mein_name`	`Mein-name`
`meineStadt`	`Meine Stadt`
`_privat`	`5123privat`
`GROSS`	`GROS$`

Wählt aussagekräftige Variablennamen

- Verwendet möglichst aussagekräftige Variablennamen
  - Nicht so aussagekräftig: `string1 = 'Giuseppe'`
  - Aussagekräftig: `mein_name = 'Giuseppe'`

Schlüsselwörter

– Python hat 29 reservierte Wörter, welche einzeln nicht als Variablennamen verwendet werden dürfen

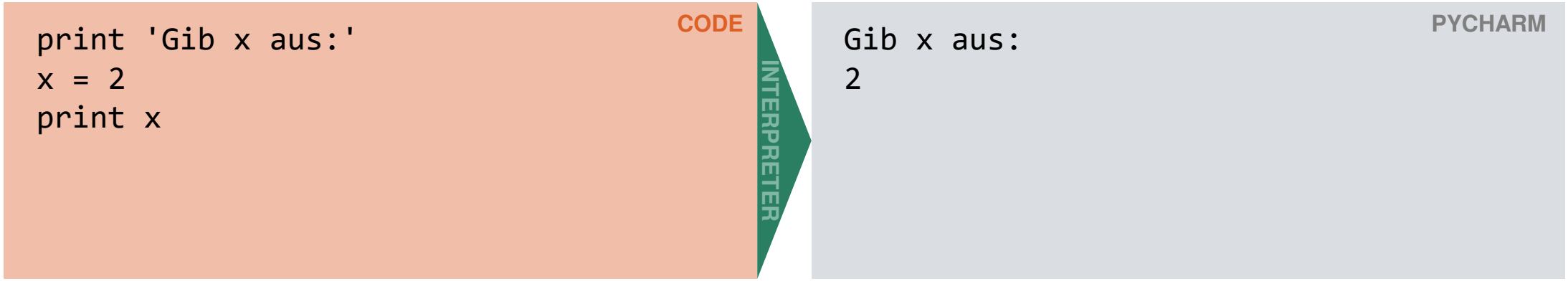
Reservierte Namen («keywords»)					
`and`	`def`	`exec`	`if`	`not`	`return`
`assert`	`del`	`finally`	`import`	`or`	`try`
`break`	`elif`	`for`	`in`	`pass`	`while`
`class`	`else`	`from`	`is`	`print`	`yield`
`continue`	`except`	`global`	`lambda`	`raise`	

– Beispiel: Der Variablenname `finally_exec` ist z.B. gültig; der Variablenname `pass` hingegen nicht

Anweisungen und Ausdrücke

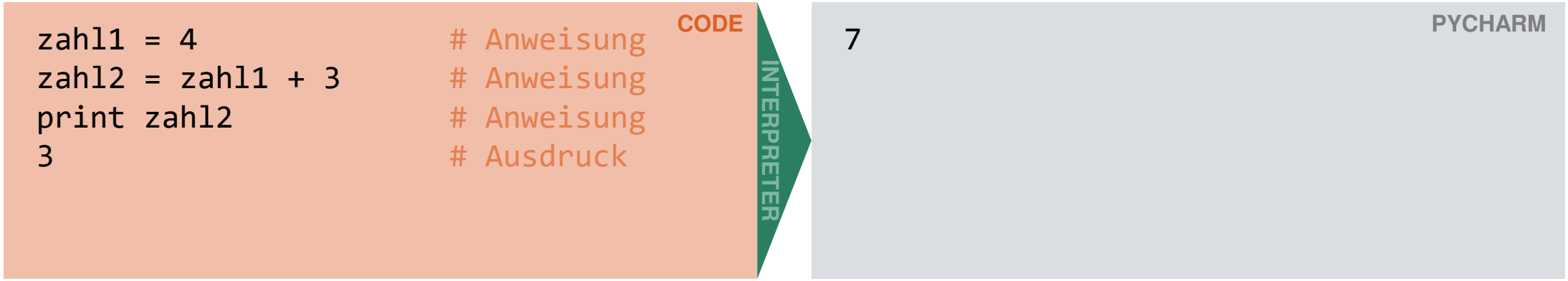
Anweisungen («Statements»)

- Eine *Anweisung* ist eine Instruktion (oder Befehl), welche der Python-Interpreter ausführen kann
  - Beispiel: Die Wertzuweisung `sprache = 'Python'` ist eine Anweisung
- Ein Code kann eine Folge von Anweisungen enthalten;
  der Python-Interpreter führt dabei jede Zeile von oben nach unten einzeln aus



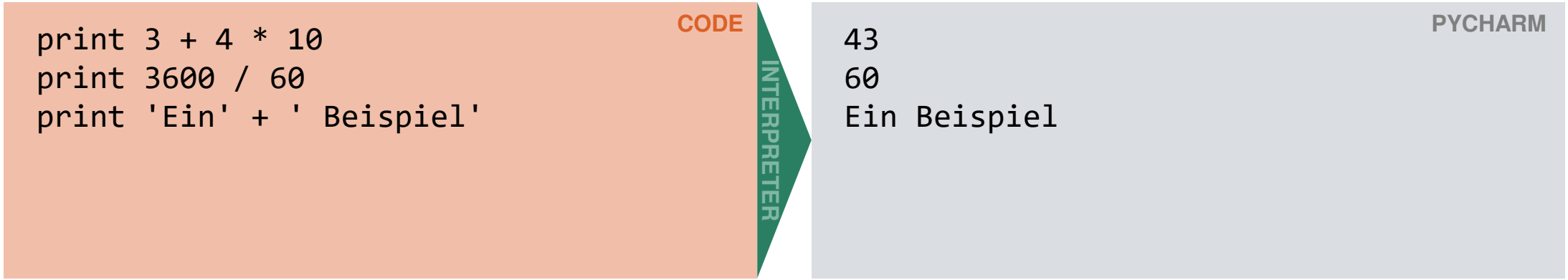
Die Auswertung von Ausdrücken («Expressions»)

- Ein *Ausdruck* ist eine Kombination von Werten, Variablen, und Operatoren
- Ein Ausdruck kann immer ausgewertet werden
- In einem Code ist ein alleinstehender Ausdruck eine legale Anweisung



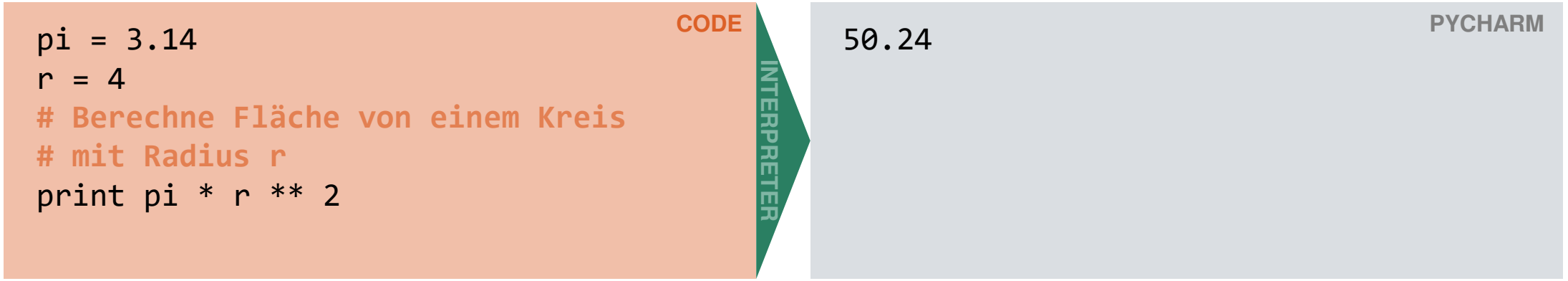
Operatoren

- *Operatoren* sind spezielle Symbole, die z.B. Berechnungen wie die Addition oder Multiplikation darstellen
- Werte, die durch Operatoren verknüpft werden, heissen *Operanden*
- Die Bedeutung eines Operators hängt vom Datentyp der Operanden ab
  - Z.B. Der + Operator angewendet auf zwei Zahlen addiert diese zusammen; der + Operator angewendet auf zwei Strings verkettet sie hingegen



Kommentare

- Grössere Programmierprojekte können aus mehreren tausenden Zeilen Code bestehen
  - Code wird immer komplizierter zu verstehen / lesen
- Kommentare helfen, den Code verständlicher darzustellen / zu erklären wo nötig
- Kommentare werden mit dem # Symbol markiert, und werden vom Interpreter ignoriert



Aufgaben

[Aufgabe]

– Versucht die Aufgaben 1.1, 1.2, und 1.3 zu lösen

Zeichenketten / Strings

Wiederholung von Strings

– Verwende den * Operator um einen String zu wiederholen

CODE

```
sehr_str = 'sehr '
cool_str = 'cool'

print 2 * sehr_str + cool_str
```

INTERPRETER

PYCHARM

```
sehr sehr cool
```

Verkettung von mehreren Strings

– Verwende den + Operator um Strings zu verketten

CODE

```
string1 = 'Hallo'
string2 = ', Welt'
string3 = '!'

print string1 + string2 + string3
```

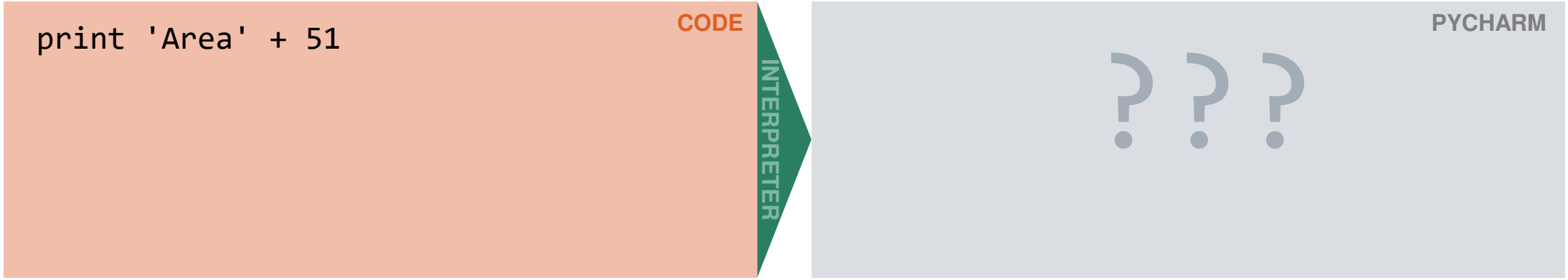
INTERPRETER

PYCHARM

```
Hallo, Welt!
```

Verkettung von Strings mit Zahlen

– Was geschieht, wenn wir z.B. 'Area' mit 51 verketten möchten?



Verkettung von Strings mit Zahlen

– Was geschieht, wenn wir z.B. 'Area' mit 51 verketten möchten?

CODE

```
print 'Area' + 51
```

INTERPRETER

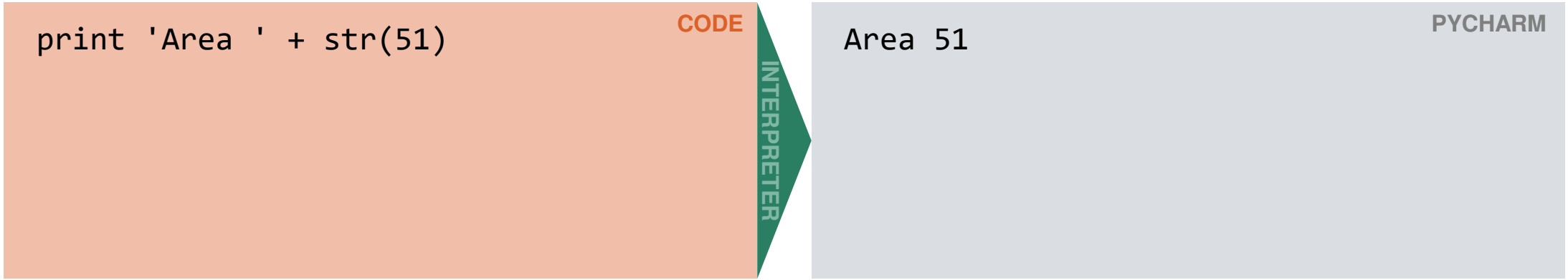
PYCHARM

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: cannot concatenate
                'str' and 'int' objects
```

– *Tipp*: Operatoren nur auf Operanden, die von ähnlichen Typen (z.B. Zahlen) stammen anwenden
  – Im obigen Code versuchen wir den Operator + auf Operanden von komplett verschiedenen Typen anzuwenden (Strings und ganze Zahlen)

Typumwandlungen

- Python bietet die Möglichkeit an, den Typ von Variablen zu ändern
- Wenn ich z.B. eine Variable zahl die eine ganze Zahl ist, so kann ich sie mit str(zahl) in einen String umwandeln



«errare humanum est» – Die drei Fehlerarten

1. Syntaktische Fehler
   - Werden vom Interpreter erkannt («Sprachfehler»)
   - z.B. `print(Hallo Welt)` statt `print('Hallo Welt!')`
2. Laufzeit Fehler
   - Fehler, die erst bei der Ausführung des Programms auftreten. Programm bricht ab.
   - z.B. Division mit 0
3. Semantische (logische) Fehler
   - Programm bricht nach Ausführung nicht ab.
   - z.B. Programm verkauft Alkohol an Jugendliche unter 18 Jahren
     (wir erhalten keine Fehlermeldung)

«errare humanum est» – Debugging zur Hilfe

- Fehlersuche
- Eine der herausforderndsten Teile des Programmierens

Aufgabe

[Aufgabe]

– Versucht die Aufgabe 1.4 zu lösen

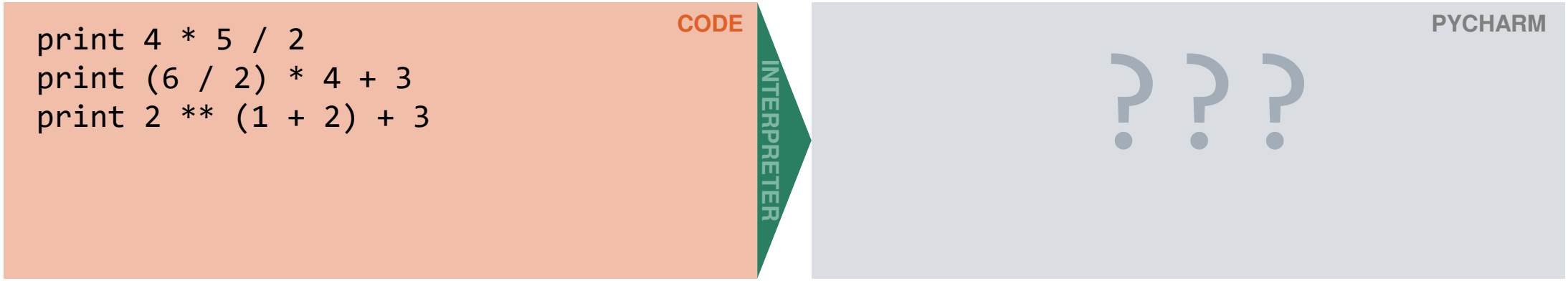
Mathematische Datentypen – ganze Zahlen und Fliesskommazahlen

Mathematische Operatoren und die Vorrangregeln

- Eine Anweisung kann aus mehreren mathematischen Operatoren bestehen
- Reihenfolge der Auswertung hängt von den folgenden Vorrangregeln ab (höchster Vorrang oben):

Operator	Operation	Beispiel	Resultat
(…)	Klammern	`(2 * 3) ** 2`	`36`
**	Exponent	`2 ** 3`	`8`
%	Modulus	`22 % 10`	`2`
/	Division	`12 / 4`	`3`
*	Multiplikation	`10 * 2`	`20`
-	Subtraktion	`18 - 8`	`10`
+	Addition	`1 + 1`	`2`

Mathematische Operatoren und die Vorrangregeln



Addition einer Fliesskommazahl mit einer ganzen Zahl

CODE

```
gnz_zahl = 1000
flkom_zahl = 2.4813
summe = gnz_zahl + flkom_zahl

print type(summe)
print summe
```

INTERPRETER

PYCHARM

```
<type 'float'>
1002.4813
```

Addition zweier ganzen Zahlen

CODE

```
gnz_zahl1 = 1000
gnz_zahl2 = 3000
summe = gnz_zahl1 + gnz_zahl2

print type(summe)
print summe
```

INTERPRETER

PYCHARM

```
<type 'int'>
gnz_zahl1
```

Multiplikation einer Fliesskommazahl mit einer ganzen Zahl

CODE

```
gnz_zahl = 3
flkom_zahl = 4.5
mult = gnz_zahl * flkom_zahl

print type(mult)
print mult
```

INTERPRETER

PYCHARM

```
<type 'float'>
13.5
```

Multiplikation zweier ganzen Zahlen

CODE

```
gnz_zahl = 4
flkom_zahl = 5
mult = gnz_zahl * flkom_zahl

print type(mult)
print mult
```

INTERPRETER

PYCHARM

```
<type 'int'>
20
```

Aufgabe

[Aufgabe]

– Wir versuchen die Aufgabe 1.5 gemeinsam zu lösen

Division zweier Zahlen – Regeln

- Wenn zwei Zahlen miteinander dividiert werden…
  - …und beides sind ganze Zahlen, so ist das Ergebnis auch eine ganze Zahl (Nachkommastellen werden einfach abgeschnitten)
  - …und eine davon eine Fliesskommazahl ist, so ist das Ergebnis eine Fliesskommazahl

CODE

```
div1 = 1 / 2
div2 = 5 / 2.0

print type(div1)
print div1

print type(div2)
print div2
```

INTERPRETER

PYCHARM

```
<type 'int'>
0
<type ‘float'>
2.5
```

Typumwandlung

- Python bietet die Möglichkeit an, den Typ von Variablen zu ändern
- `int(zahl)`: zahl wird in ganze Zahl umgewandelt
- `float(zahl)`: zahl wird zu einer Fliesskommazahl umgewandelt

CODE

```
flkom_zahl = 1.482392
gnz_zahl = int(flkom_zahl)

print type(gnz_zahl)
print gnz_zahl
```

INTERPRETER

PYCHARM

```
<type 'int'>
1
```

Lernziele – Check

– Nach dieser Einheit wissen wir:
  1. … was eine Variable ist
  2. … wie man einer Variable einen Wert zuweist
  3. … wie man eine oder mehrere Variablen ausgibt
  4. … was *Strings*, *Floats*, und *Ints* sind
  5. … wieso wir Typumwandlungen benötigen

Aufgabe

[Aufgabe]

– Versucht die Aufgabe 1.6 zu lösen

Universität Zürich[UZH]

Zentrale Informatik – IT Fort- und Weiterbildungen

Funktionen Teil 1 – Ein Einstieg

Lernziele

– Nach dieser Einheit wissen wir:
  1. … was eine Funktion ist
  2. … wie wir eine Funktion definieren können
  3. … welchen Vorteil Funktionen mit sich bringen
  4. … wie wir Funktionen um Argumente erweitern können

Funktionen

- Wir haben bereits die `print()` Funktion kennengelernt
- Eine Funktion ist wie ein Miniprogramm im Programm selbst

CODE

```
def hello():
  print 'Hello!'

hello()
hello()
```

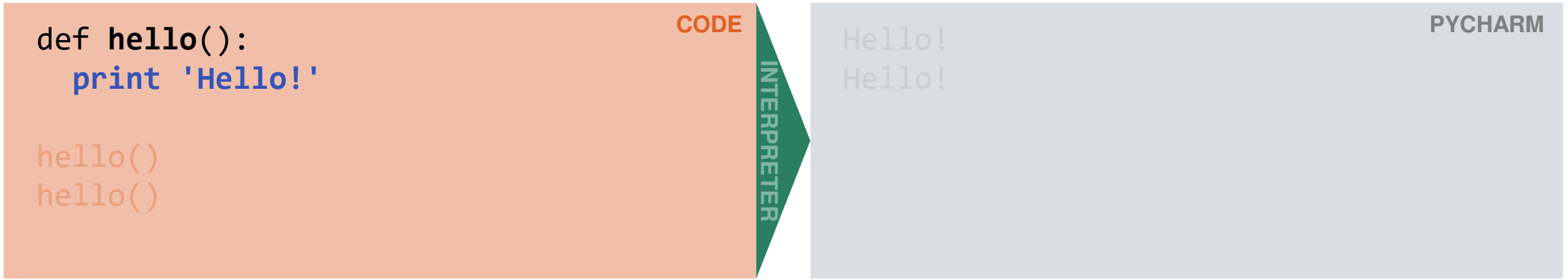
INTERPRETER

PYCHARM

```
Hello!
Hello!
```

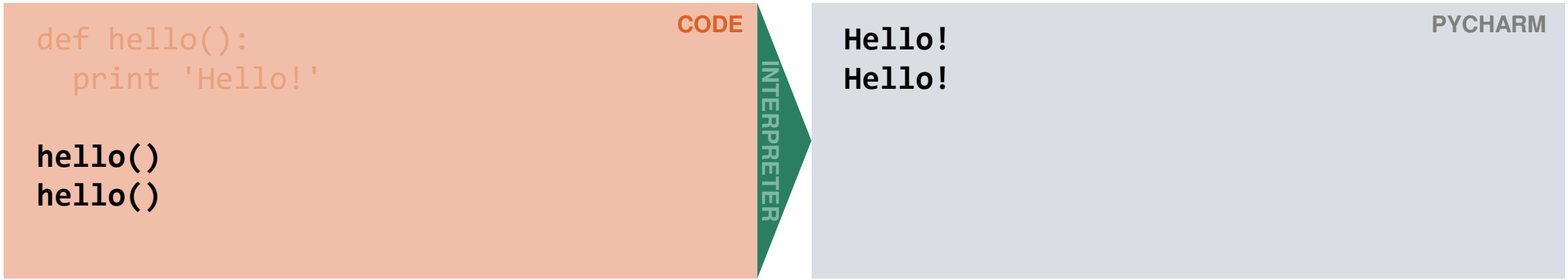
Eine Funktion definieren

- Mittels `def` Keyword kann man eine Funktion definieren
  - Verlangt wird dabei der **Funktionsname** und der **Funktionskörper**
  - Der Code im Funktionskörper wird erst ausgeführt, wenn die Funktion aufgerufen wird



Eine Funktion aufrufen

– Eine selbst-definierte Funktion kann man aufrufen, indem man den Funktionsnamen gefolgt von einem Klammerpaar im Code tippt:



Warum Funktionen?

– Wir möchten unter anderem vermeiden, Code zu duplizieren

CODE

```
print 'Hello!'
print 'Hello!'
print 'Hello!'
print 'Yes?'
print 'Hello!'
print 'Hello!'
print 'Hello!'
```

INTERPRETER

PYCHARM

```
Hello!
Hello!
Hello!
Yes!
Hello!
Hello!
Hello!
```

Warum Funktionen?

– Wir möchten unter anderem vermeiden, Code zu duplizieren

CODE

```
def say_it_3_times():
  print 'Hello!'
  print 'Hello!'
  print 'Hello!'

say_it_3_times()
print 'Yes?'
say_it_3_times()
```

INTERPRETER

PYCHARM

```
Hello!
Hello!
Hello!
Yes!
Hello!
Hello!
Hello!
```

Warum Funktionen?

– Vorteil: Wenn wir nun statt `Hello!` lieber `Hallo!` ausgeben möchten, so müssen wir diese Anpassung lediglich bei der Funktion `say_it_3_times` vornehmen

CODE

```
def say_it_3_times():
  print 'Hallo!'
  print 'Hallo!'
  print 'Hallo!'

say_it_3_times()
print 'Yes?'
say_it_3_times()
```

INTERPRETER

PYCHARM

```
Hallo!
Hallo!
Hallo!
Yes!
Hallo!
Hallo!
Hallo!
```

Aufgabe

[Aufgabe]

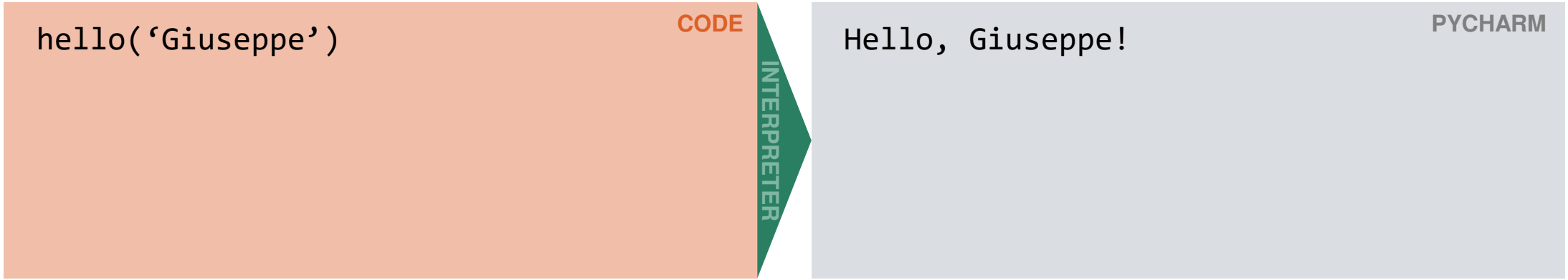
– Wir versuchen die Aufgabe 2.1 gemeinsam zu lösen

Einer Funktion ein Argument übergeben

- Einer Funktion kann man *Werte* mitgeben. Diese nennt man *Argumente* einer Funktion
  - Beispiel: `print('Hello')`
    - Wir übergeben der Funktion `print` den Wert (oder das Argument) `'Hello'`; dieser wird anschliessend ausgegeben
    - Wir möchten also der `print` Funktion mitteilen, welchen Namen sie ausgeben soll

Einer Funktion ein Argument übergeben

- Wir wollen nun ermöglichen, dass der Funktion `hello` der Wert `'Giuseppe'` übergeben werden kann, und danach `'Hello, Giuseppe!'` ausgegeben wird
- Wie erreichen wir das? Unser Ziel:



Funktion um einen Parameter erweitern

CODE

```
def hello(name):
  print 'Hello,' + name + '!'

hello('Giuseppe')
```

INTERPRETER

PYCHARM

```
Hello, Giuseppe!
```

- Die Funktion `hello` wird um den Parameter `name` erweitert
- Der Parameter ist dabei eine Variable, in welcher das Argument (z.B. 'Giuseppe') gespeichert wird
- Wenn wir also `hello('Giuseppe')` aufrufen, so wird beim Eintritt in die Funktion `hello` der Variable `name` den Wert `'Giuseppe'` zugewiesen («`name = 'Giuseppe'`»)

Gültigkeitsbereich («Scope») von Parametern

CODE

```
def hello(name):
  print 'Hello,' + name + '!'

hello('Giuseppe')
print name # nicht möglich!
```

INTERPRETER

PYCHARM

```
Hello, Giuseppe!
```

– Die Variable `name` ist nur in der Funktion `hello` gültig und kann von ausserhalb nicht verwendet werden

Funktion um mehrere Parameter erweitern

CODE

```
def hello(first, last):
  print 'Hello,' + first + ' '
        + last + '!'

hello('Giuseppe', 'Accaputo')
```

INTERPRETER

PYCHARM

```
Hello, Giuseppe Accaputo!
```

- Die Funktion `hello` kann nun mit zwei Parameter aufgerufen werden
- Wir können auch Funktionen mit mehr als zwei Parameter definieren

Mathematische Funktionen

– Das `math` Modul enthält eine Sammlung der gängigsten Funktionen aus der Mathematik

CODE

```
import math

log_e = math.log(10.0)
pi_genauer = math.pi
wurzel = math.sqrt(25)
winkel = 45
sinus = math.sin(winkel)
```

– Übersicht der verfügbaren Funktionen: https://docs.python.org/2.7/library/math.html

Verknüpfung

– Ausdrücke wie Werte und Variablen darf man auch als Teil anderer Ausdrücke (z.B. Funktionen) verwenden

CODE

```
import math

winkel = 45
x = math.sin(winkel + math.pi / 2)

y = math.exp(math.log(10))
```

Lernziele – Check

– Nach dieser Einheit wissen wir:
  1. … was eine Funktion ist
  2. … wie wir eine Funktion definieren können
  3. … welchen Vorteil Funktionen mit sich bringen
  4. … wie wir Funktionen um Argumente erweitern können

Aufgaben

[Aufgabe]

- Versucht die Augabe 2.2 und 2.3 zu lösen
- Überlegt euch, was für Ausgaben von den jeweiligen Funktionen erwartet werden (Funktionen testen!)

Bedingte Anweisungen («Conditionals»)

Lernziele

– Nach dieser Einheit wissen wir:

1. … was der Boolesche Datentyp darstellt
2. … was bedingte Anweisungen sind (und, dass Zweige nicht nur an Bäumen zu finden sind)
3. … wie eine Bedingung aussehen kann bei bedingten Anweisungen
4. … wie wir mit logischen Operatoren mehrere Bedingungen verknüpfen können

Der Boolesche Datentyp – Wahr oder falsch

Boolescher Datentyp

– Der Boolesche Datentyp erlaubt nur zwei Werte, nämlich `True` oder `False` (und nichts dazwischen)

CODE

```
licht_ist_an = True
tuere_ist_offen = False

print(licht_ist_an)
print(tuere_ist_offen)
print type(licht_ist_an)
```

INTERPRETER

PYCHARM

```
True
False
<type 'bool'>
```

Boolescher Ausdruck

- Ein Boolescher Ausdruck (bestehend aus Werten, Variablen, und Operatoren) evaluiert entweder zu `True` oder `False`
  - Der == Operator ist ein Boolescher Operator der zwei Ausdrücke miteinander vergleicht
  - a == b kann als «Ist a gleich b?» interpretiert werden.
    - `1 == 5` evaluiert also zu `False`
    - `5 == 5` evaluiert also zu `True`

Vergleichsoperatoren

Ausdruck	Bedeutung
x != y	Ist x ungleich y?
x > y	Ist x grösser als y?
x < y	Ist x kleiner als y?
x >= y	Ist x grösser oder gleich y?
x <= y	Ist x kleiner oder gleich y?

Vergleichsoperatoren

CODE

```
zahl1 = 1
zahl2 = 4
print zahl1 > zahl2
```

INTERPRETER

PYCHARM

```
False
```

Logische Operatoren

- Es gibt in Python drei logische Operatoren: `and`, `or`, und `not`
- `and`:
  - Der Ausdruck `(x > 0) and (x < 10)` ist nur dann `True`, wenn beide Ausdrücke erfüllt sind, also wenn `x` grösser als 0 *und* kleiner als 10 ist, sonst ist er `False`
- `or`
  - Der Ausdruck `(y < 0) or (x < 10)` ist dann `True`, wenn *mindestens* eine der beiden Bedingungen `True` ist, also wenn entweder `y` kleiner 0, `x` grösser 10 oder beide Ausdrücke `True` sind
- not: eine Bedingung / einen Booleschen Ausdruck negieren
  - Der Ausdruck `not(x > y)` ist dann `True`, wenn `x > y` `False` ist (Negation)

Modulo Operator

- `a % b`:
  Ermittelt den Rest bei der Division des ersten Operanden `a` durch den zweiten Operanden `b`
- Im folgenden stellen wir fest, dass 10 dividiert durch 3 das Resultat 3 Rest 1 ergibt:

CODE

```
division = 10 / 3
print division
rest = 10 % 3
print rest
```

INTERPRETER

PYCHARM

```
3
1
```

Modulo Operator

– Mit dem Modulo Operator können wir auch feststellen, ob eine Zahl x durch eine andere Zahl y teilbar ist

CODE

```
zahl = 1020540

if (zahl % 2)== 0:
  print 'Zahl ist gerade'
else:
  print 'Zahl ist ungerade'
```

INTERPRETER

PYCHARM

```
Zahl ist gerade
```

Vorrangregeln aktualisiert

Operator	Operation
`**`	Exponent
`%`	Modulus
`/`	Division
`*`	Multiplikation
`-`	Subtraktion
`+`	Addition
`<, <=, >, >=, !=, ==`	Vergleichsoperatoren
`not`	Negation
`and`	UND Verknüpfung
`or`	ODER Verknüpfung

Bedingte Anweisungen

Bedingte Anweisungen

- Wir möchten gewisse Sachen nur dann ausführen, wenn eine bestimmte Bedingung erfüllt ist
  - Beispiel: Nur volljährige Personen in sollen in Klub Eintritt erhalten
- Einfachste Form: die `if`-Anweisung

CODE

```
alter = 19

if alter >= 18:
  print 'Kunde ist volljährig'
```

INTERPRETER

PYCHARM

```
Kunde ist volljährig
```

- Boolescher Ausdruck nach dem `if` Schlüsselwort nennt man die *Bedingung*

Bedingte Anweisungen

- `if`-Anweisung besteht aus einem *Kopf* und einem *Anweisungsblock*
- Anweisungsblock wird eingerückt und kann mehrere Anweisungen enthalten
- Anweisungsblock wird nur ausgeführt, wenn Bedingung erfüllt ist

CODE

```
alter = 19

if alter >= 18:                          # Kopf, enthält Bedingung
  print 'Herzlich Willkommen'            # Anweisungsblock
  preis_fuer_einlass_verlangen()         # Anweisungsblock
  stempfel_geben()                       # Anweisungsblock
```

Alternative Ausführung – Einfache Verzweigung

- Zweite Form der `if`-Anweisung ist die alternative Ausführung
- Wenn Bedingung nun falsch ist, wird der zweite Block ausgeführt
  - Alternative Blöcke nennt man *Zweige*

CODE

```
alter = 19

if alter >= 18:
  print 'Herzlich Willkommen'          # Erster Block
  preis_fuer_einlass_verlangen()       # Erster Block
  stempfel_geben()                     # Erster Block
else:
  print 'Eintritt verweigert'          # Zweiter Block
```

Alternative Ausführung – Mehrfache Verzweigung

– Wenn es mehr als zwei Möglichkeiten gibt benötigen wir mehr als zwei Zweige

CODE

```
if x < y:
  print x + ' ist kleiner als ' + y
elif x > y:
  print x + ' ist grösser als ' + y
else:
  print x + ' und ' + y + ' sind gleich gross'
```

– `elif` ist Abkürzung für «else if»

– Es wird wieder nur ein Zweig ausgeführt und Bedingungen werden der Reihe nach überprüft

Alternative Ausführung – Verschachtelte Verzweigung

– Wir können Verzweigungsanweisungen auch ineinander verschachteln:

CODE

```
if x < y:
  print x + ' ist kleiner als ' + y
else:
  if x > y:
    print x + ' ist grösser als ' + y
  else:
    print x + ' und ' + y + ' sind gleich gross'
```

– Verschachtelte Verzweigungen können sehr schnell schwer lesbar werden

Aufgabe

[Aufgabe]

– Versucht die Aufgabe 3.1 zu lösen

Alternative Ausführung – Logische Operatoren

– Logische Operatoren ermöglichen es oftmals geschachtelte Verzweigungen zu vereinfachen:

CODE

```
if x > 0:
  if x < 10:
    print 'x eine positive Zahl mit nur einer Ziffer'
```

CODE

```
if x > 0 and x < 10:
    print 'x eine positive Zahl mit nur einer Ziffer'
```

Funktionsausführung terminieren – Die `return` Anweisung

- `return` Anweisung ermöglicht es, eine Funktion frühzeitig zu beenden
- Möglicher Grund: eintreten einer Fehlerbedingung

CODE

```
def wurzel(x):
  if x < 0:
    print 'Nur positive Zahlen und die 0 sind erlaubt'
    return

  resultat = sqrt(x)
  print 'wurzel(x) = ' + str(resultat)
```

Lernziele – Check

– Nach dieser Einheit wissen wir:
  1. … was der Boolesche Datentyp darstellt
  2. … was bedingte Anweisungen sind (und, dass Zweige nicht nur an Bäumen zu finden sind)
  3. … wie eine Bedingung aussehen kann bei bedingten Anweisunge
  4. … wie wir mit logischen Operatoren mehrere Bedingungen verknüpfen können

Aufgaben

[Aufgabe]

Versucht die Aufgaben 3.2 und 3.3 zu lösen (Bonus-Aufgabe: 3.4)

Nächste Woche

- Funktionen Teil 2 – Rückgabewerte, Wiederverwendbarkeit, und mehr
- Iterationen – Wiederholungen und Durchwanderungen
- Datenstrukturen – Listen, Strings, Tuppel, und Dictionaries
- Dateien

Nächste Woche

– Treffpunkt wieder Samstag, 08:50 Uhr vor dem Gebäude Y10