Python - Grundlagen der Programmierung

Tag 1

Giuseppe Accaputo

g@accaputo.ch

Schön seid ihr heute hier

Wer seid ihr?

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

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]

Aufbau Kurs

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

Python 3.0 Spickzettel

Python 3.0 Spickzettel Giuseppe Accaputo g@accaputo Kurs: Grundlagen der Programmierung für Nicht-Informatiker, HS18	
Variablen	
`variablen_name = <wert>`	
`typ_der_variable = type(variable)`	
Datentypen	
Integer (Int)	`-25, 2, 14, 100, -20`
Float	`2.4123, -1.1312, 4.14123`
String	`'Hallo 1', 'Hallo 2'`
Boolean	`True, False`
List	`[1, 1.2423, 'zwei']`
Tupel	`(1, 2, 3, 'vier')`
Dictionary	`{'key1': wert1, 'key2': wert2}`
Eingabe	
`eingabe = input('Bitte Name eingeben:')`	
`eingabe = int(input('Bitte Zahl eingeben:'))`	
`eingabe = float(input('Bitte Zahl eingeben:'))`	
Ausgabe	
`print('Wert der Variable:', variable)`	
`print('Typ der Variable', type(variable))`	
Strings – Teil 1	
`ein_string = 'Hallo, Welt!'`	Variable vom Typ String definieren
`ein_string[1:5]`	Segment von Index 1 bis und mit Index 4 selektieren
`ein_string[-1]`	Auf das letzte Zeichen zugreifen
`ein_string.lower()`	In Kleinbuchstaben umwandeln
`ein_string.upper()`	In Grossbuchstaben umwandeln
`ein_string.replace(alt, neu)`	`alt` durch `neu` in `ein_string` ersetzen
`string1 == string2`	Ist `string1` gleich `string2`?
`zahl_als_string = str(1.234)`	Typumwandlung zu String

`hallo = 'Hallo, '`
`welt = 'Welt!'`
`hallo_welt = hallo + welt`
	`+` Operator: Zwei Strings verknüpfen

`area = 'Area '`
`area_zahl = 51`
`area_51 = area + str(area_zahl)`
	`+` Operator und `str()`: String und Zahl verknüpfen

Zahlen	
`int(variable)`	Typumwandlung zu `Int`
`float(variable)`	Typumwandlung zu `Float`
Mathematische Operatoren	
`x ** y`	Exponent, x^y
`x % y`	Modulus; berechnet den Rest der Division `x` geteilt durch `y`
`x / y`	Division
`x * y`	Multiplikation
`x - y`	Subtraktion
`x + y`	Addition
`x op y`, und `x`,`y` sind beide Ints	Resultat der Operation ist ein Int (`op` kann `+, -, *, /` sein)
`x op y`, und `x` oder `y` ist Float	Resultat der Operation ist ein Float (`op` kann `+, -, *, /` sein)
Funktionen	

`def hallo():`
`  print('Hallo!')`
`hallo() # Aufruf`
	Eine Funktion ohne Rückgabewert

`def hallo(name):`
`  print('Hallo, ', name)`
`hallo('Klasse')  # Aufruf`
	Eine Funktion mit einem Argument

`def summe(x, y, z):`
`  return x + y + z`
`print(summe(1,2,3)) # Aufruf`
	Eine Funktion mit einem Rückgabewert

`import math`
`math.sqrt(zahl) # Wurzel`
`math.log(zahl) # Logaithmus`
	Mathematische Funktionen verwenden

"Was war das denn? Ich versteh' gar nichts mehr"

– Ihr werdet während diesem Kurs evtl. in gewissen Situationen überfordert sein, gewisse Konzepte nicht gleich auf Anhieb verstehen, oder allgemein das Gefühl haben, dass ihr nicht für's Programmieren gemacht seid…

  …und das sind alles Gefühle, die in unserer Situation völlig normal sind, denn wir lernen in diesen zwei Kurstagen einige Themen kennen, die zu einer für euch komplett neuen Disziplin gehören.

  Auch mir ging es so, als ich mit dem Programmieren begonnen habe.

  Dies ist alles Teil des Lernprozesses. Es ist völlig okay, sich so zu fühlen.

Fragen während und nach dem Kurs sind zu jeder Zeit erlaubt und erwünscht

- **Deshalb wichtig**: Auch wenn ihr das Gefühl habt, dass eine Frage evtl. "nicht gut genug" sein könnte oder ein Konzept einfach zu verstehen sein sollte, und ihr deshalb nicht fragen möchtet, stellt die Frage bitte trotzdem ☺
  - Dies gibt mir auch die Möglichkeit, nach besseren Erklärungen für gewisse Konzepte zu suchen
- Fragen gehören zum Lernprozess und sollen zu jeder Zeit gestellt werden dürfen
- Ihr dürft mich auch sehr gerne während oder nach dem Kurs kontaktieren, falls ihr zu irgendwelchen Themen fragen habt: g@accaputo.ch

Feedback

– Konstruktives Feedback ist in meinen Kursen immer sehr willkommen, einerseits weil ich gerne meinen Kurs verbessere, und andererseits weil es mir wichtig ist, dass ihr euch während dem Kurs wohl fühlt

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

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

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

Ein Beispielprogramm

1. Zahl 1 hat Wert 10
2. Zahl 2 hat Wert 30
3. Gib Zahl 1 + Zahl 2 auf dem Bildschirm 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

Warum Python?

– Idee für ein Programm:

IDEE

1. Zahl 1 hat Wert 10
2. Zahl 2 hat Wert 30
3. Gib Zahl 1 + Zahl 2 auf dem Bildschirm aus

– Programm in Python umgesetzt:

CODE

```
zahl_1 = 10
zahl_2 = 30
print(zahl_1 + zahl_2)
```

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.

Learning by Doing (and by Making Errors)

- Programmieren ist eine *hands-on experience*
- Habt keine Angst Sachen in Python auszuprobieren
- Fehler machen ist eine wesentliche Komponente des Lernprozesses und hilft euch auch gewisse Situationen besser zu verstehen
  - Falls ihr irgendwo stecken bleibt während einer Aufgabe oder inmitten eines Slides, meldet euch bitte gleich ungeniert bei mir oder bei euren Mitstudenten
- Versucht auch Aufgaben mit Blatt und Stift zu lösen, vorallem wenn ihr gerade versucht, ein neues Konzept besser zu verstehen
  - Es kann manchmal sehr helfen eine Computer-Pause einzulegen

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

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 Datentypen

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

- Datentypen charakterisieren / beschreiben:
  - eine spezifische Menge von zusammengehörenden Werten (z.B. ganze Zahlen)
  - welche Operationen darauf ausgeführt werden können (z.B. Addition zweier Zahlen)
  - wie die Werte abgespeichert werden

- *Beispiel*: Wir haben einen Katze Namens Petra.
  - *Petra* ist eine konkrete Instanz / Realisierung einer Katze und *Katze* ist der Typ
  - Der Typ *Katze* fasst einige Eigenschaften zusammen, welche alle Katzen gemeinsam haben
    - Sehr vereinfacht kann man sagen:
      - Alle Katzen haben 4 Beine → Petra ist eine Katze → Petra hat 4 Beine
      - Alle Katzen miauen → Petra ist eine Katze → Petra kann miauen

Datentypen

- Ein Wert gehört immer zu einem bestimmten Datentyp
  - Der Wert `2` ist eine *ganze Zahl* (ein *Int*, Abkürzung für *Integer*)
  - Der Wert `'Python'` ist eine *Zeichenkette* (ein *String*)
  - Der Wert `3.14159` ist eine *Fliesskommazahl* (ein *Float*, Abkürzung für *Floating Point Number*)

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'`

Aufgabe • Datentypen

[Aufgabe]

Wert	Datentyp
`'Hello, there!'`	?
`3.14`	?
`9000`	?
`'4123.314239'`	?

Datentyp ermitteln

– 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))
```

INTERPRETER

PYCHARM

```
<class 'str'>
<class 'float'>
<class 'int'>
```

Variablen

Variablen

- Eine *Variable* ist wie eine beschriftete Box, in welcher wir einen Wert verstauen (*speichern*) können
  - Eine Variable besteht aus einem *Namen* und einem zugewiesenen *Wert (*inklusive *Datentypen)*
- 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
name = 'Giuseppe'

print(answer)
print(name)
print(type(name))
```

INTERPRETER

PYCHARM

```
42
Giuseppe
<class 'str'>
```

Variablen

CODE

```
answer = 42
name = 'Giuseppe'

print(answer)
print(name)
print(type(name))
```

INTERPRETER

PYCHARM

```
42
Giuseppe
<class 'str'>
```

- Versuchen wir den obigen Code als eine Liste von Befehlen anzuschauen:
  - Der Befehl `answer = 42` sagt «weise der Variable `answer` die Ganzzahl (*Int*) `42` zu»
  - Der Befehl `name = 'Giuseppe'` sagt «weise der Variable `name` die Zeichenkette (*String*) `'Giuseppe'` zu»
  - Der Befehl `print(answer)` sagt «gib bitte den Wert der Variable `answer` aus»

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$`

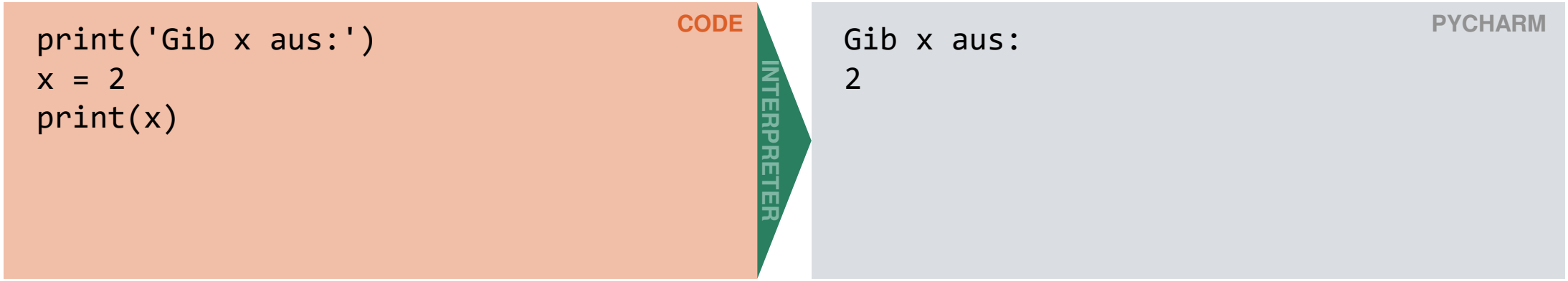
Wählt aussagekräftige Variablennamen

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

Anweisungen und Ausdrücke

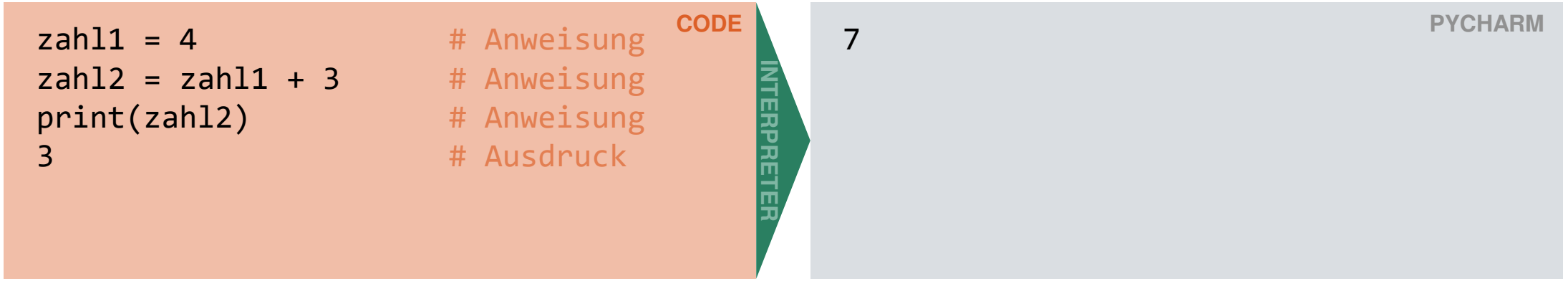
Anweisungen («Statements»)

- 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») [Wichtiges Konzept]

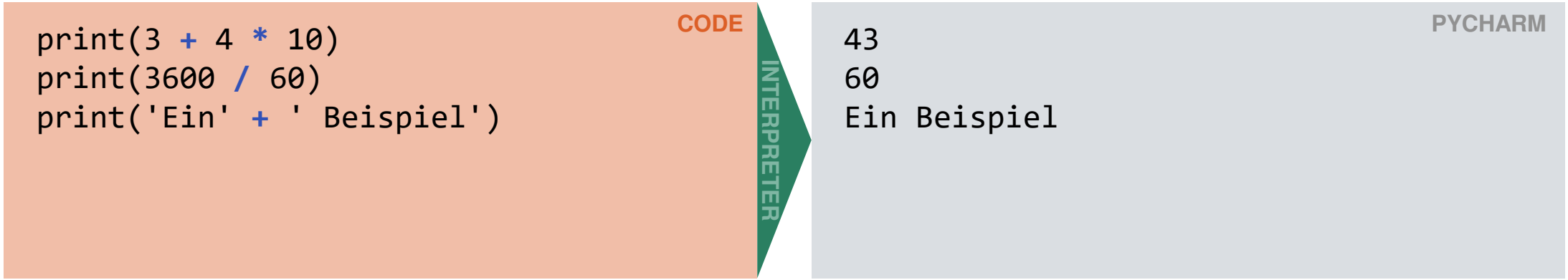
- Ein *Ausdruck* ist eine Kombination von Werten, Variablen, und Operatoren
- Ein Ausdruck kann immer ausgewertet werden, d.h. der Ausdruck wird zu einem Wert evaluiert
- In einem Code ist ein alleinstehender Ausdruck eine legale Anweisung



- **`zahl2 = zahl1 + 3`** ist eine Anweisung, wobei sich nach dem Gleichzeichen (**`=`**) ein Ausdruck befindet

Operatoren

- *Operatoren* sind spezielle Symbole, die z.B. Berechnungen wie die Addition oder Multiplikation darstellen
- Werte, die durch Operatoren verknüpft werden, heissen *Operanden*
- *Wichtig*: 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



Eingaben einlesen

- Wir können auch Eingaben einlesen, d.h. wir können eine Zeichenfolge eingeben und z.B. in eine Variable speichern
- Mittels `input()` können wir verlangen, dass eine Zeichenfolge eingelesen wird

CODE

```
print("Bitte etwas eingeben: ")
meine_eingabe = input()
print(meine_eingabe)
```

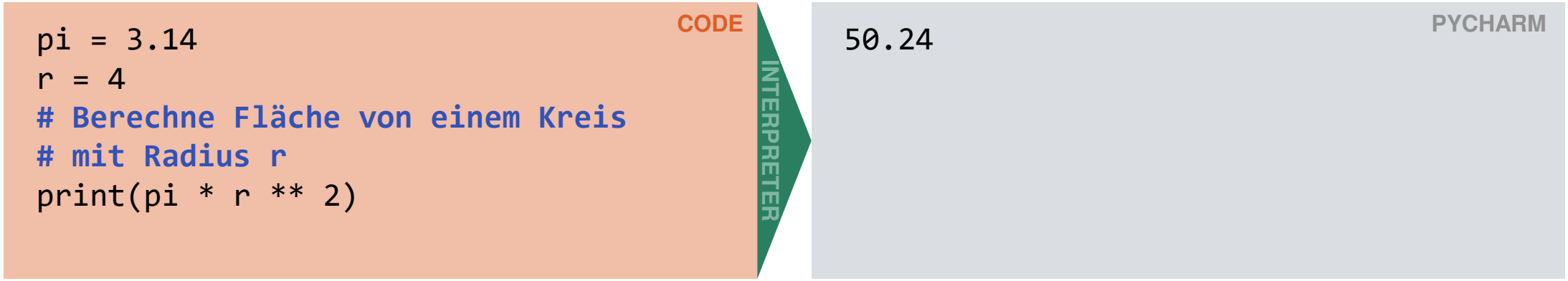
INTERPRETER

PYCHARM

```
Bitte etwas eingeben: Hallo!
Hallo!
```

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



Zeichenketten / Strings

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?

CODE

```
print('Area' + 51)
```

INTERPRETER

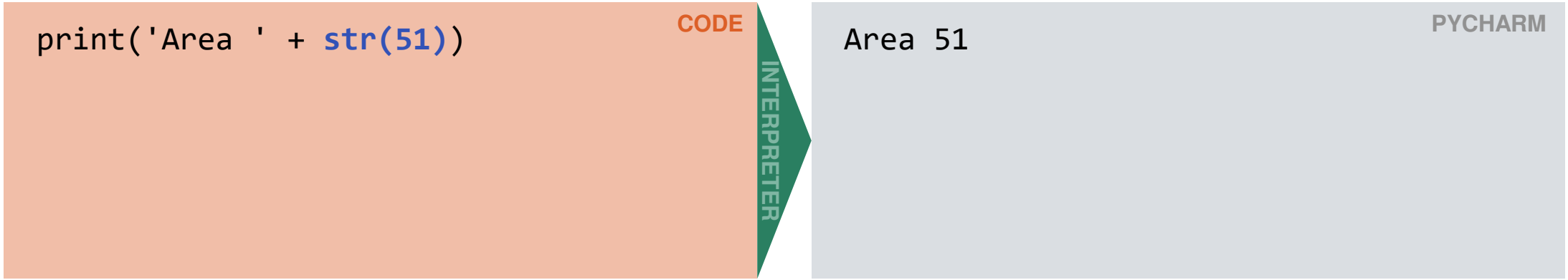
PYCHARM

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: must be str, not int
```

– *Tipp*: Operatoren nur auf Operanden anwenden, die von ähnlichen Typen (z.B. Zahlen) stammen
  – 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



Typumwandlungen

– Wir können auch einen String in eine Zahl umwandeln, angenommen der String beinhaltet nur eine Zahl:

CODE

```
print(int('123'))
print(float('3.14'))

# Funktioniert nicht!
print(int('3.14'))
```

INTERPRETER

PYCHARM

```
123
3.14
ValueError: invalid literal for
            int() with base 10: '3.14'
```

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`

Aufgabe • Vorrangregeln

[Aufgabe]

Ausdruck	Ergebnis
`4 * 5 / 2`	?
`(6 / 2) * 4 + 3`	?
`2 ** (1 + 2) + 3`	?
`2 ** 3 + 2 * 3`	?

Addition einer Fliesskommazahl mit einer ganzen Zahl

- Fliesskommazahlen (`float`) enthalten in der Regel viel mehr Informationen (Nachkommastellen) als Integer (keine Nachkommastellen)
- Implizite Typumwandlung zum Informationsreicheren Datentyp, nämlich `float`

CODE

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

print(type(summe))
print(summe)
```

INTERPRETER

PYCHARM

```
<class '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

```
<class 'int'>
4000
```

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

```
<class '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

```
<class 'int'>
20
```

LC 1.1 • Durchschnittslohn berechnen {Live Coding}

Lasst uns gemeinsam ein Programm schreiben, dass folgende Anweisungen ausführt:

1. Es sind 50 Franken verfügbar. Speichert diesen Wert in die Variable `anz_franken` ab
2. Des Weiteren arbeiten gerade 4 Helfer. Speichert diesen Wert in die Variable `anz_helfer` ab
3. Nun möchten wir herausfinden, wie viele Franken jeder Helfer erhält.
   Schreibt eine Anweisung (auf einer Zeile), die diesen Wert berechnet und ihn in die Variable `anz_franken_pro_helfer` speichert
4. Gebt die Variable `anz_franken_pro_helfer` auf dem Bildschirm aus

Division zweier Zahlen – Regeln

- Division zweier Zahlen mittels `/` Operator wird in Python 3.* automatisch zu einem Float umegewandelt
- Der Operator `//` kann hingegen verwendet werden, um ein Ganzzahl Ergebnis zu generieren

CODE

```
div1 = 1 / 2
div2 = 1 // 2

print(type(div1))
print(div1)

print(type(div2))
print(div2)
```

INTERPRETER

PYCHARM

```
<class 'float'>
0.5
<class 'int'>
0
```

Typumwandlung

- Python bietet die Möglichkeit an, den Typ von Variablen zu ändern
- `int(zahl)` → `zahl` wird in ganze Zahl umgewandelt (kann zu Informationsverlust führen)
- `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

```
<class '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

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. … wie wir Funktionen um Parameter erweitern können

Funktionen

- Thema ist erfahrungsgemäss anfänglich kompliziert / evtl. schwer verständlich (vorallem auch, weil wir schon seit einigen Stunden hier sind)
- Fragen sind zu jedem Zeitpunkt erlaubt und wirklich erwünscht

Vorschau

CODE

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

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

PYCHARM

```
Hello, Giuseppe Accaputo!
```

Funktionen

- Beispiele von Funktionen (Thema für nächste Woche):
  - Funktionen, die eine Ausgabe auf dem Bildschirm generieren, z.B. `print()`
  - Funktionen, die zu einem Wert evaluieren, z.B. mathematische Funktion oder auch `input()`

Die `print()` Funktion

- Eingabe: Eine Zeichenfolge bestehend aus Zeichen, Zahlen, etc.
- Ausgabe: Die Zeichenfolge wird auf dem Bildschirm ausgegeben

Die `print()` Funktion

CODE

```
print('Eins')
print('Zwei')
print('Drei')
```

INTERPRETER

PYCHARM

```
Eins
Zwei
Drei
```

Eine Funktion definieren

– Eine Funktion ist wie ein Miniprogramm im Programm selbst

CODE

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

hello()
hello()
```

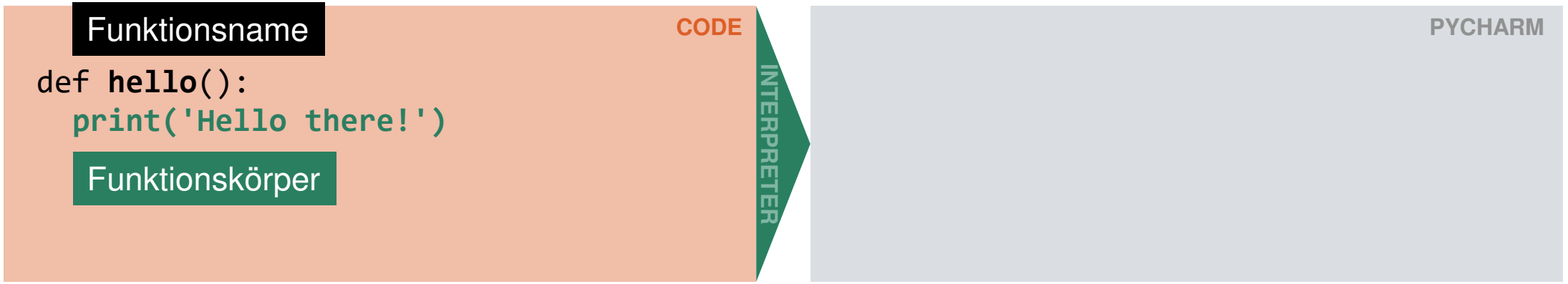
INTERPRETER

PYCHARM

```
Hello!
Hello!
```

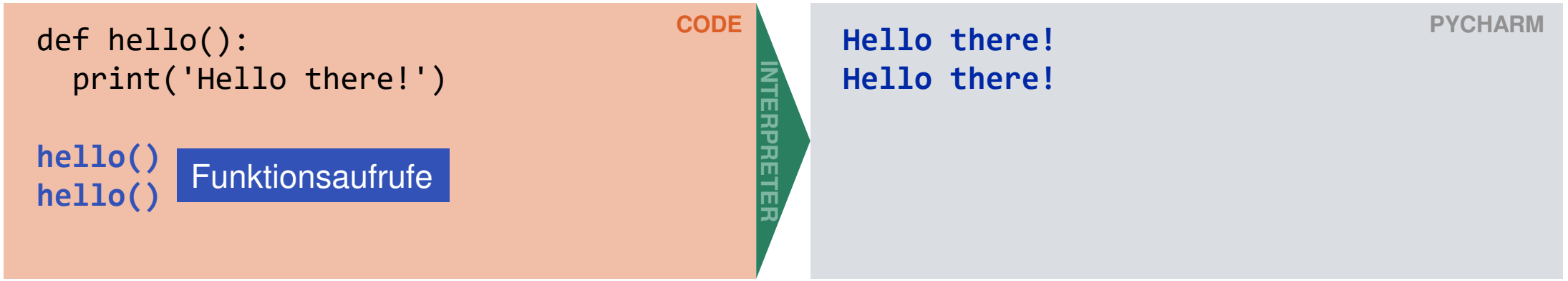
Eine Funktion definieren

- 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
    - *Wichtig*: Die Definition einer Funktion wird lediglich *registriert*



Eine Funktion aufrufen

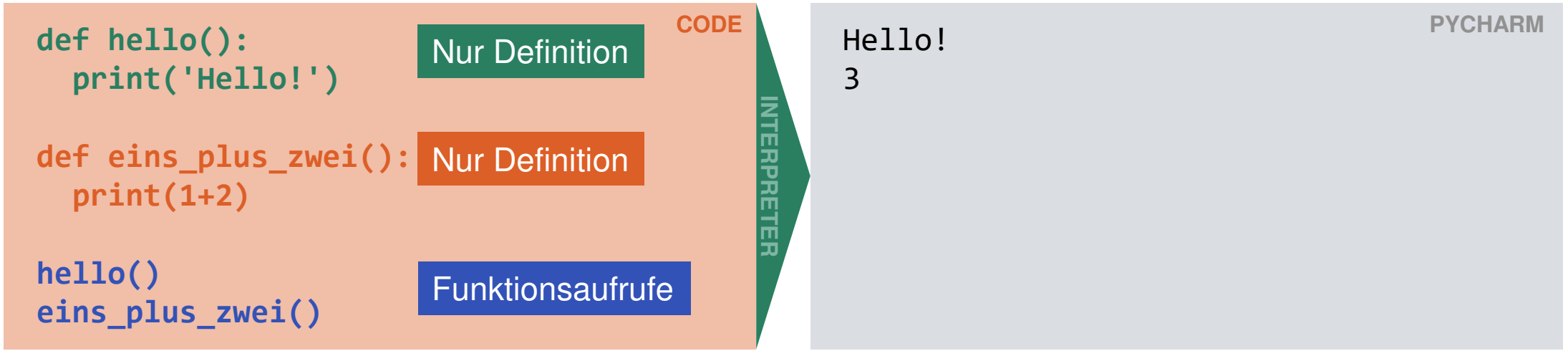
– Eine selbst-definierte Funktion kann man aufrufen, indem man den Funktionsnamen gefolgt von einem Klammerpaar im Code tippt:



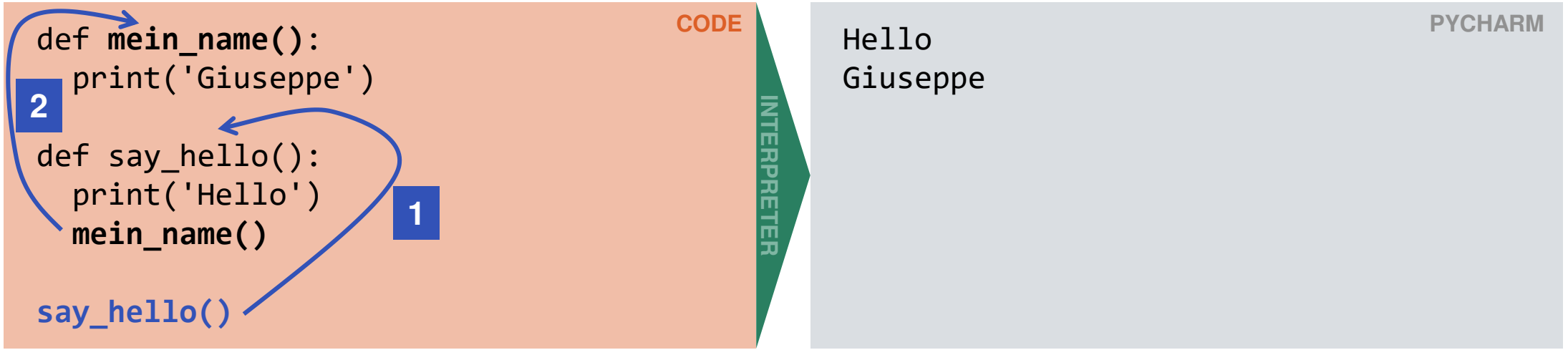
– Erst bei einem Funktionsaufruf wird die Funktion (bzw. der dazugehörige Funktionskörper) ausgeführt

Mehrere Funktionen

- Nachdem die Funktion ausgeführt / evaluiert wurde, geht der Code an der Stelle weiter, an welcher der Funktionsaufruf getätigt wurde
- Im folgenden Code wird zuerst `hello()` ausgeführt, und danach `eins_plus_zwei()`:



Funktionen in Funktionen aufrufen



Strukturierung durch Einrückung

– Anweisungen, die zusammen gehören, müssen die gleiche Einrückungstiefe haben. Dabei kann man z.B. zwei Leerzeichen für solch eine Einrückungstiefe verwenden:

CODE

```
def grosse_funktion():
  print('Eine')
  print('grosse')
  print('Funktion)

def kleine_funktion():
  print('Ganz klein')

print('Gehört zu keiner Funktion!')

def weitere_funktion():
  print('Noch eine Funktion!')
```

Strukturierung durch Einrückung

– Anweisungen, die zusammen gehören, müssen die gleiche Einrückungstiefe (Gelb) haben. Dabei kann man z.B. zwei Leerzeichen für solch eine Einrückungstiefe verwenden:

CODE

```
def grosse_funktion():
  print('Eine')
  print('grosse')
  print('Funktion)

def kleine_funktion():
  print('Ganz klein')

print('Gehört zu keiner Funktion!')

def weitere_funktion():
  print('Noch eine Funktion!')
```

Aufgabe • Ausgaben verstehen [Aufgabe]

– Was wird bei Ausführung des folgenden Codes ausgegeben?

CODE

```
def funktion_a():
  print("A")

def funktion_b():
  print("B")
  funktion_a()

def funktion_c():
  funktion_b()
  print("C")
  funktion_a()

funktion_c()
```

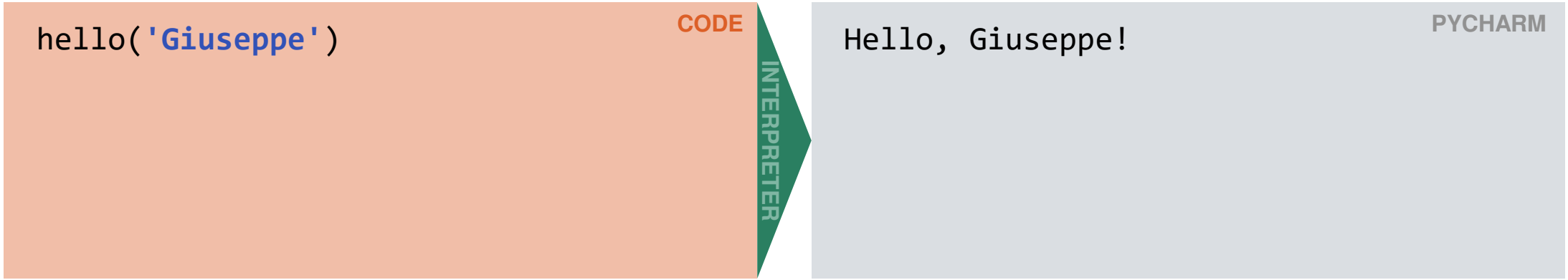
Einer Funktion ein Parameter übergeben

- Einer Funktion kann man *Werte* mitgeben. Diese nennt man *Parameter* oder *Argument* 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 String sie ausgeben soll

- Bezogen auf die Funktion `summe(a,b,c)` sind die folgenden Aussagen korrekt:
  1. Die Funktion `summe` nimmt 3 Parameter entgegen
  2. Wir können die Funktion `summe` mit 3 Parameter aufrufen (oder: Wir können der Funktion `summe` 3 Parameter übergeben)

Einer Funktion ein Parameter 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 ein 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 der Wert des Parameters (z.B. `'Giuseppe'`) gespeichert wird
- Wenn wir also `hello('Giuseppe')` aufrufen, so wird beim Eintritt in die Funktion `hello` der Variable `name` der Wert `'Giuseppe'` zugewiesen («`name = 'Giuseppe'`»)

Gültigkeitsbereich («Scope») von Parametern

CODE

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

hello('Giuseppe')
print(name)
```

- Die Variable `name` ist nur in der Funktion `hello` gültig und kann von ausserhalb nicht verwendet werden
  - *Wichtig*: Unbedingt auf Einrückungstiefe (Gelb) achten

Gültigkeitsbereich («Scope») von Parametern

CODE

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

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

INTERPRETER

PYCHARM

```
Hello, Giuseppe!
NameError: name 'name' is not defined
```

- Die Variable `name` ist nur in der Funktion `hello` gültig und kann von ausserhalb nicht verwendet werden
  - *Wichtig*: Unbedingt auf Einrückungstiefe achten: `name` ist nur im markierten Bereich verwendbar

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 (getrennt mittels Komma)

Lernziele – Check

– Nach dieser Einheit wissen wir:
  1. … was eine Funktion ist
  2. … wie wir eine Funktion definieren können
  3. … wie wir Funktionen um Parameter erweitern können

Bitte sichert eure Dateien

Python for Beginners Videos by Microsoft

- Playlist mit allen Python Videos: https://www.youtube.com/playlist?list=PLlrxD0HtieHhS8VzuMCfQD4uJ9yne1mE6
  - Code: https://github.com/microsoft/c9-python-getting-started