

# F02A02: 有 Polylingual Pingu

Diese Aufgabe ist Teil der freiwilligen inoffiziellen Zusatzaufgaben von Eric Jacob und Jonas Wende, erstellt im WS 23/24 für *IN0002: Grundlagenpraktikum Programmierung*.

Weder sind sie durch die ÜL überprüft, noch unbedingt vollständig richtig.

Fehler gerne melden: [eric.jacob.2003@gmail.com](mailto:eric.jacob.2003@gmail.com)

🎄 *Advent, Advent, ein Server brennt...* oder so. Vorweihnachtlich gibt es jeden Adventssonntag eine freiwillige inoffizielle Zusatzaufgabe, die weit über den Inhalt von PGdP hinausgehen und euch einige Programmierkonzepte zeigen sollen, die ihr so in PGdP nicht lernt.

## F02A02: 有 Polylingual Pingu

🎯 Lernziele

📖 Backstory

📄 Aufgabenbeschreibung

🐍 Abteilung für Würgeschlangen - Python

📋 Aufgaben

🔍 Lösungsvorschlag

😋 Abteilung für Unsinn und österreichisches Essen - JavaScript-Schnitzeljagd

📋 Aufgabe

Anmerkung

📁 Anhang

Das Speklatius-Lachs-Rezept von Pingu-Opa Max

Das Lachskekse-Rezept

Eine Liste weiterer *interessanter* Eigenheiten von JavaScript

## 🎯 Lernziele

In dieser Aufgabe lernt ihr zwei weitere Sprachen neben Java kennen - und, dass ihr die meisten Programmiersprachen mit eurem Java-Wissen verstehen könnt, auch wenn ihr noch nie zuvor in diesen programmiert habt.

## 📖 Backstory

Zur vorweihnachtlichen Tradition der Pinguine gehört seit eh und je der alljährliche Besuch im "Museum für Alles" in Cod-City. Heute, am 02. Adventssonntag, ist es wieder soweit - nach dem gemeinsamen Mittagessen, für welches Pingu-Opa Max seinen berühmten [Lachs mit Speklatiuskruste](#) gekocht hat, ging es los ins Museum. In jeder Abteilung fallen den Babypinguinen neben all den antiken Statuen und Zeichnungen besonders die überlieferten Schriftstücke auf, welche in Vitrinen an der Wand hängen. Trotz deren Alter von teils mehreren Jahrhunderten ist die Schrift darauf noch gut erhalten - nur schade, dass die Pinguine die Schriftzeichen nicht entziffern können... Kannst du ihnen helfen?

## 📄 Aufgabenbeschreibung

Ziel dieser Aufgabe ist es, die Ähnlichkeit und Unterschiede zwischen verschiedenen Programmiersprachen zu sehen und Code in verschiedenen Sprachen umzusetzen.

## Abteilung für Würgeschlangen - Python

Neben den ganzen imposanten Terrarien hängt an der Wand hinter einer Glasscheibe folgendes Papyrus der alten Pharaouinen:

```
1 #####
2 # Imports #
3 #####
4
5 # ignore the imports, they are not that relevant for understanding the Python
  language right now
6 from __future__ import annotations
7 from typing import *
8
9 import random as rnd
10
11
12 #####
13 # Classes declaration #
14 #####
15
16 class Snake:
17     """
18     This class represents a Snake with:
19     - name
20     - genus
21     - mother
22     - father
23     - date born
24     """
25
26     snake_genera = ["Zwergpython", "Baumpython", "Schwarzkopfpython",
27 "wasserppython", "Raupenpython", "Netzpython"]
28
29     # a Snake constructor which requires a name, genus, parents tuple and
  birthday
30     def __init__(self, name: str, genus: str, parents: Tuple[Union[Snake,
  None], Union[Snake, None]], birthday: int) -> None:
31         self.name = name
32         if genus not in self.snake_genera:
33             print("This looks like a weird mutation...")
34             self.genus = self.snake_genera[0]
35         else:
36             self.genus = genus
37             self.mother, self.father = parents
38             self.birthday = birthday
```

```

39     # lets the snake make a "hiss" sound on the console
40     def hiss(self) -> None:
41         print("🐍 " + self.name + " hisses!")
42
43     # lets the snake slither around
44     def slither(self) -> None:
45         print("🐍", self.name, "slithers!")
46
47     # breeds this egg with another Snake and returns a new Egg
48     def breed(self, other_snake: Snake) -> Egg:
49         return Egg(rnd.choice(self.snake_genera), (self, other_snake))
50
51 class Egg:
52     """
53     This class represents a (Snake) Egg with:
54     - information about the days until it hatches
55     - genus
56     - mother
57     - father
58     """
59
60     next_name_index = 0
61     next_snake_names = ["Sssusan", "Zzzoe", "Sssteven", "Franc-hiss"]
62
63     # an Egg constructor requiring a genus and parents tuple
64     def __init__(self, genus: str, parents: Tuple[Snake, Snake]) -> None:
65         self.days_until_hatch = 5
66         self.genus = genus
67         self.mother, self.father = parents
68
69     # a method that slowly hatches the egg when incubated and always returns
the "current entity",
70     # i.e. either the yet-to-hatch Egg or a Snake once hatched
71     def incubate(self, current_day: int) -> Union[Egg, Snake]:
72         if rnd.randrange(2) == 0:
73             self.days_until_hatch -= 1
74             print("🥚 The egg cracked a little. It will hatch soon!")
75         if self.days_until_hatch <= 0:
76             snake_name = Egg.next_snake_names[Egg.next_name_index]
77             Egg.next_name_index += 1
78             return Snake(
79                 snake_name,
80                 self.genus,
81                 (self.mother, self.father),
82                 current_day
83             )
84         else:
85             return self
86
87
88 #####
89 # Code procedure #

```

```

90 #####
91
92 # set up the terrarium
93 terrarium = {
94     "snakes" : [],
95     "eggs" : []
96 }
97
98 # two initial snakes - Adam and Eve
99 adam = Snake("Adam", "Zwergpython", (None, None), 0)
100 eve = Snake("Eve", "Wasserpython", (None, None), 0)
101 terrarium["snakes"].append(adam)
102 terrarium["snakes"].append(eve)
103 print(adam.name + " and " + eve.name + " moved into the terrarium.")
104
105
106 # let Adam and Eve lay 3 eggs
107 for i in range(3):
108     terrarium["eggs"].append(adam.breed(eve))
109     print("🥚 An egg was laid!")
110
111
112 # now incubate the eggs until all the baby snakes hatched
113 print("The eggs will now be incubated")
114 day = 0
115 while not len(terrarium["eggs"]) == 0:
116     print("🌞 A new day " + str(day) + " begins")
117     for i in range(len(terrarium["eggs"])):
118         print("Incubating egg in the hatchery station at position", str(i),
119             " - needs", str(terrarium["eggs"][i].days_until_hatch), "more
120             days to hatch"
121             )
122         egg_or_snake = terrarium["eggs"][i].incubate(day)
123         if type(egg_or_snake) == Snake:
124             print("Placing the new snake into the terrarium")
125             terrarium["snakes"].append(egg_or_snake)
126             terrarium["snakes"][-1].slither()
127         print("🧹 Cleaning up the eggshells of hatched snakes")
128         terrarium["eggs"] = [x for x in terrarium["eggs"] if x.days_until_hatch >
129         0]
130     day += 1
131
132 # let's simulate the snakes living in the terrarium until the visitor leaves
133 print("The terrarium is now opened to visitors")
134 visitor_still_watching = True
135 while visitor_still_watching:
136     for _ in range(rnd.randrange(1, 4)):
137         snake_taking_action = rnd.choice(terrarium["snakes"])
138         snake_taking_action.hiss() if rnd.randrange(2) == 0 else
139         snake_taking_action.slither()

```

```

139     user_input = ""
140     while True:
141         user_input = input("👤 The museum curator asks: Do you want to keep
watching the [s]nakes or [l]eave? ")
142         if user_input == "l":
143             print("The museum is now closed for the day.")
144             exit()
145         elif user_input == "s":
146             break
147         else:
148             print("The museum curator did not understand what you said.")

```

## Aufgaben

**Wichtig:** Da viele der folgenden Aufgaben rein konzeptuell und zum Nachdenken sind, gibt es dafür keine Tests, lediglich einige Notizen zur Lösung auf der nächsten Seite.

1. Führe das Skript ein paar Mal aus.  
Damit ihr euch nicht mit nervigem Setup rumschlagen müsst, gibt es den Code auch [hier](#) als Repl, das im Browser läuft.
2. Schau dir den Python-Code an und versuche, ihn zu verstehen. Was fällt dir auf? Was ist gleich, was ist anders als in Java?
3. Für Rückgaben und Parameter bei Funktionen und Methoden muss man in Python eigentlich keine types angeben – warum kann das trotzdem sinnvoll sein?
4. Versuche, den Python-Code in Java umzusetzen. Was geht dabei in Java einfacher, was in Python?

## Lösungsvorschlag

2. Ein paar Dinge, die auffallen könnten:

- Anderer Syntax. Unter anderem:
  - Keine Semikolons
  - Kommentare beginnen mit einem `#`, Kommentarblöcke stehen zwischen `"""`
  - `boolean` heißt in Python `bool`, `String` heißt `str`
  - Es gibt Wörter für logische Operationen: `&&` = `and`, `!` = `not`, `||` = `or`
  - Funktionen, Klassen, `if`s und Schleifen werden nicht mit `{}` umschlossen, sondern unter `:` eingerückt
  - Keine runden Klammern um Statements bei `if`, `for`, `while`
  - Die `for`-Schleife nimmt eine `range` entgegen - aus `for (int i = 0; i < ziel; i++)` wird `for i in range(ziel)`.
  - Die `enhanced for`-Schleife kann mit `in` direkt alle Werte aus Listen nehmen: `for (element : liste)` wird zu `for element in liste`.
  - Python hat kein Prä-/Postinkrement/-dekrement – das kürzeste Mögliche ist `+=` bzw `--`.
  - User input (und auch Dateien lesen/schreiben, hier nicht gezeigt) ist deutlich einfacher. Statt `Scanner`-Objekte erstellen zu müssen und Fehler abzufangen schreibt man einfach `variable = input()` und bekommt einen String zurück.
  - Datentypen-Umwandlung ist einfacher; statt `Integer.parseInt("23")` schreibt man `int("23")`. Umgekehrt kann z.B. die `print`-Funktion nur mit `str`s umgehen:

```
1 print("Der wert ist " + 23) # Fehler, da man auf strs nicht
  addieren kann
2 print("Der wert ist " + str(23)) # korrekt
```

- In Klassen müssen Attribute nicht zu Beginn der Klasse deklariert werden. Üblicherweise werden diese im Konstruktor (`__init__()`) erstellt. Auch muss jeder Methode einer Klasse `self`, also die Referenz auf das zu bearbeitende Objekt (ähnlich zu `this` in Java) mitgegeben werden.
- Es gibt keine alles umschließende Klasse und keine `main`-Methode, die bei Programmstart aufgerufen wird. Python ist eine sogenannte Skriptsprache, die zwar Objektorientierung bietet, aber nicht vorschreibt. Beim Ausführen wird einfach Zeile für Zeile des Programms ausgeführt. In Java dagegen muss alles in einer Klasse passieren. Weniger Objektorientierung hat auch zur Folge, dass man nicht wie in Java erst aus der Klasse `System` das Attribut `out` wählen muss, um darauf `println()` auszuführen - man kann einfach `print()` schreiben.
- Allgemein werden keine Datentypen wie `int`, `String`, etc beim Erstellen einer Variable angegeben. Python wählt automatisch einen passenden Typ und passt intern auch die Größe beliebig an (es gibt also keine Probleme wie in Java, dass eine Zahl nicht in einen `short` passt und abgeschnitten wird). Man kann mit `variable: datatype` zwar explizite *type annotation* vorgeben (statt `variable = "Hi"` also `variable: str = "Hi"`), muss dies aber nicht. Das hat auch zur Folge, dass z.B. folgender Code keinen Fehler wirft:

```

1 variable = "Hi"
2 print(type(variable)) # gibt `str` aus
3
4 variable = 23 # Python ändert den Typ von `variable` hier selbst um
5 print(type(variable)) # gibt `int` aus
6
7 variable = []
8 print(type(variable)) # gibt `list` aus

```

oder auch innerhalb einer Liste (das Python-Äquivalent zu Arrays, allerdings ohne fixe Länge und ohne fixen Datentyp):

```

1 liste = ["Hallo", 6.5, [9]]

```

- Python wird im Gegensatz zu Java beim Ausführen nicht *kompiliert*, sondern *interpretiert*. Darum muss – im Gegensatz zu Java – alles im Code definiert werden, *bevor* es verwendet werden kann. Beispiel:

```

1 print(fkt(2)) # Fehler, da fkt noch nicht definiert wurde
2
3 def fkt(zahl):
4     return zahl + 2
5
6 print(fkt(2)) # funktioniert

```

3. Das kann z.B. von Vorteil sein, um dem Nutzer einen Hinweis zu geben, was eine Funktion erwartet. Aus

```

1 def do_something(value):
2     # Code

```

kann der Nutzer beim Aufruf von `do_something()` nicht schließen, welchen Typ er hier übergeben soll oder was er zurückbekommt, ohne sich den Code anzuschauen. So ist es deutlich klarer (auch, weil eine IDE beim Hovern dann anzeigt, was erwartet wird):

```

1 def do_something(value: int) -> bool:
2     # Code

```

4. Siehe `würgeschlangen/wuergeschlangenJava`

Nach dieser kniffligen Aufgabe gibt's für die Jungpinguine erstmal ein paar [Lachsplätzchen](#) zur Belohnung. Eine kurze Pause später begeben ihr euch zur nächsten Abteilung im Museum:



# Abteilung für Unsinn und österreichisches Essen - JavaScript-Schnitzeljagd

Einmal die Treppe hoch, schon steht ihr in der *Abteilung für Unsinn und österreichisches Essen*. Wie das zum interaktiven Mitmachspiel an der Wand, einer JavaScript-Schnitzeljagd, passt? "Ganz einfach", sagt Museumsdirektorin Jeanine-Sahra: "In JavaScript, there is a beautiful, elegant, highly expressive language that is buried under a steaming pile of good intentions and blunders".

Etwas verunsichert betretet ihr den Raum. Links und rechts wird der Raum von beleuchteten Vitrinen gesäumt, welche verschiedenste abstruse Konstruktionen und Kunstinstallationen beherbergen; besonders viele bilden Schnitzel, Klöße, Tafelspitz und Kaiserschmarrn ab.

Ein vergilbtes Blatt Papier aus dem Jahre 1995 gibt euch die erste von zehn kleinen Aufgaben, deren Lösungen ihr nacheinander [bei der Museumsdirektorin abgeben](#) müsst, um die nächste Aufgabe zu erhalten. "Wer alle zehn Rätsel löst", sagt sie, "erhält einen kleinen Pinguinaufkleber".

## Aufgabe

Diese Aufgabe soll euch als Kontrast zum mehr oder weniger logischen Python-Syntax davor einige weniger logische ✨ *Besonderheiten* ✨ der Sprache JavaScript aufzeigen, welche besonders auf Webseiten verwendet wird, um Animationen und interaktive Elemente zu erstellen.

Ihr erhaltet am Ende dieses Texts die erste Teilaufgabe. Sobald ihr die Lösung (d.h. den Konsolen-Output der `console.log()`-Zeilen) ermittelt habt, könnt ihr diese in ihrer Reinform (d.h. ohne weitere Anführungszeichen davor oder danach, kein "Lösung: " davor schreiben, etc.) bei der Museumsdirektorin in [diesem Formular](#) abgeben. Sofern eure Antwort richtig ist, erhaltet ihr einen Hinweis, wo auf dem Garchinger Unicampus ihr die nächste Aufgabe findet (das sind physische kleine Zettel, die z.B. hinter einer Heizung oder an anderen unauffälligen Orten hängen). Insgesamt gibt es 10 Teilaufgaben (plus eine Aufgabe 0), d.h. 10 Code-Snippets, die ihr suchen und lösen müsst, um einen Sticker zu erhalten.

**Wichtig:** Aus technischen Gründen müsst ihr beim erneuten Öffnen des Formulars leider auch alle vorigen Antworten erneut eingeben; merkt euch diese also! Alternativ loggt ihr euch mit eurem Google-Account ein (dieser wird nicht erfasst, speichert aber euren bisherigen Bearbeitungsstatus des Formulars).

```
1 //Aufgabe 0 (warm-up)
2 console.log("hello");
```

Hinweis: Ihr solltet euch über die Aufgaben zunächst selbst Gedanken machen. Falls ihr jedoch an einer Stelle nicht weiterkommt, könnt ihr den JavaScript-Code in eurem Browser testen. Drückt dafür `Strg+Shift+J` bzw. `Cmd+Option+J` oder öffnet den Inspector mit `F12` und geht danach zum Tab `Console` (bzw. `Konsole`). Dort könnt ihr den Code einfach eingeben und erhaltet den Output der `console.log()`-Zeilen.

---



## Anmerkung

*Anmerkung:* Eigentlich hatten wir hier noch eine dritte Aufgabe geplant - leider eignen sich nicht alle Programmiersprachen gleich gut für diese Aufgabe und Tutor:innen kennen auch nur begrenzt viele Sprachen. Solltest du allerdings noch Ideen haben, welches Beispiel wir hier für die kommenden Jahre ergänzen könnten, schreib uns eine Mail oder Zulip-Nachricht: [eric.jacob.2003@gmail.com](mailto:eric.jacob.2003@gmail.com).



## Anhang

---

### Das Speklatius-Lachs-Rezept von Pingu-Opa Max

<https://www.essen-und-trinken.de/rezepte/59961-rzpt-lachsfilet-mit-spekulatiuskruste>

### Das Lachskekse-Rezept

<https://www.falstaff.com/de/rezepte/kochen/lachskekse>

### Eine Liste weiterer *interessanter* Eigenheiten von JavaScript

<https://github.com/denysdovhan/wtfjs>