How beginner-friendly is a programming language?

A short analysis based on Java and Python examples

Jean-Philippe Pellet[1,2], Amaury Dame[2], and Gabriel Parriaux[1]

[1] University of Teacher Education, Lausanne, Switzerland
[2] École polytechnique fédérale de Lausanne, Switzerland

ISSEP 2019, November 19th, 2019

Outline

1. Introduction & context
2. Comparison of excerpts & desiderata
3. Conclusion

Outline

hep/ EPFL

1. **Introduction & context**
2. Comparison of excerpts & desiderata
3. Conclusion

Our context

hep/ EPFL

- Freshman course given to future engineers at EPFL
  - — Material scientists, civil engineers
- For many, first programming course… and last
- What/how to teach? Not future computer scientists
  - — Focus on programming as a tool or on concepts of programming?
    - ➡ "*As a tool*" would mean more libraries, recipes, examples
    - ➡ "*Concepts*" has as goal a deeper understanding of the code and execution model

Java → Python

hep/ EPFL

- Course had been in Java for many years
- Faculty demanded to switch to Python

Java → Python

- Course had been in Java for many years
- Faculty demanded to switch to Python
- *Oh no…*

Java → Python

- Course had been in Java for many years
- Faculty demanded to switch to Python
- *Oh no…*

Java → Python

- Course had been in Java for many years
- Faculty demanded to switch to Python
- *Oh no…*
- Whined about it…
  Did it…
  Rather liked it!

What's relevant for the language choice?

hep/ EPFL

- Pears et al. (2007): intrinsic vs extrinsic criteria
  - *Intrinsic*: syntax, compiled vs. interpreted, paradigm, typed…
  - *Extrinsic*: industry demands, trends, availability of teaching material…
- Focusing here on *intrinsic*
- Goal is not to fuel an ongoing war
  - See how easily some cross-cutting concepts can be illustrated
  - "Easily": free of syntactic or conceptual noise
  - *Examples with excerpts from Java and Python*

Common discussions (1/2)

Common discussions (1/2)

hep/ EPFL

- Can't compare Java and Python!

  — *At some point, we have to, so that we can make a choice*

Common discussions (1/2)

hep/ EPFL

- Can't compare Java and Python!
  - *At some point, we have to, so that we can make a choice*
- Don't want to use a single language!
  - *Transference comes with mastery, important to stick to one language in the beginning*
  - *Plus: choosing two is not automatically easier than choosing one*

Common discussions (1/2)

hep/ EPFL

- Can't compare Java and Python!
  - *At some point, we have to, so that we can make a choice*
- Don't want to use a single language!
  - *Transference comes with mastery, important to stick to one language in the beginning*
  - *Plus: choosing two is not automatically easier than choosing one*
- I don't agree that having to teach difficult feature X or language Y is a problem. You have to teach formal details as well
  - *Time is limited. Time not spent on low-level points can be spent on higher-level points – just different, according to course objectives*

Common discussions (2/2)

Common discussions (2/2)

hep/ EPFL

- One should start with a visual programming language anyway
  - *Yes, we'd recommend it! Not part of the options we had*
  - *Execution model not always easier with VPLs*

Common discussions (2/2)

hep/ EPFL

- One should start with a visual programming language anyway
  - *Yes, we'd recommend it! Not part of the options we had*
  - *Execution model not always easier with VPLs*
- I want static typing!
  - *You can still get some of it in most dynamically typed languages*

Outline

1. Introduction & context
2. **Comparison of excerpts & desiderata**
3. Conclusion

1. A slightly more complicated “hello world”

hep/ EPFL

Want to exemplify:

- *Variable assignment*
- *Arithmetic op*
- *Print to terminal*

Java

```
public class Demo {
  public static void main(String[] args) {
    int side = 4;
    int area = side * side;
    System.out.println(area);
  }
}
```

1. A slightly more complicated “hello world”

hep/ EPFL

Want to exemplify:

- *Variable assignment*
- *Arithmetic op*
- *Print to terminal*

Java

```
public class Demo {
  public static void main(String[] args) {
    int side = 4;
    int area = side * side;
    System.out.println(area);
  }
}
```

1. A slightly more complicated “hello world”

hep/ EPFL

Want to exemplify:

- *Variable assignment*
- *Arithmetic op*
- *Print to terminal*

Java

```
public class Demo {
  public static void main(String[] args) {
    int side = 4;
    int area = side * side;
    System.out.println(area);
  }
}
```

1. A slightly more complicated “hello world”

hep/ EPFL

Want to exemplify:

- *Variable assignment*
- *Arithmetic op*
- *Print to terminal*

Java

```
public class Demo {
  public static void main(String[] args) {
    int side = 4;
    int area = side * side;
    System.out.println(area);
  }
}
```

1. A slightly more complicated “hello world”

hep/ EPFL

Want to exemplify:

- *Variable assignment*
- *Arithmetic op*
- *Print to terminal*

Java

```
public class Demo {
  public static void main(String[] args) {
    int side = 4;
    int area = side * side;
    System.out.println(area);
  }
}
```

1. A slightly more complicated “hello world”

hep/ EPFL

Want to exemplify:

- *Variable assignment*
- *Arithmetic op*
- *Print to terminal*

Java

```
public class Demo {
  public static void main(String[] args) {
    int side = 4;
    int area = side * side;
    System.out.println(area);
  }
}
```

Need to explain/ignore:

- *Braces*
- *Brackets*
- *Dot*
- *Class*
- *Method*
- *Static*
- *Argument*
- *Semicolon*
- *Types*

1. A slightly more complicated “hello world”

hep/ EPFL

Want to exemplify:

- *Variable assignment*
- *Arithmetic op*
- *Print to terminal*

Java

```
public class Demo {
  public static void main(String[] args) {
    int side = 4;
    int area = side * side;
    System.out.println(area);
  }
}
```

Python

```
side = 4
area = side * side
print(area)
```

Need to explain/ignore:

- *Braces*
- *Brackets*
- *Dot*
- *Class*
- *Method*
- *Static*
- *Argument*
- *Semicolon*
- *Types*

Noise

Noise

- **Noise** in a code excerpt refers to extraneous elements that have to be present (and, potentially, understood), while being unrelated, or very indirectly related, to the concept we wish to illustrate with that except

Noise

hep/ EPFL

- **Noise** in a code excerpt refers to extraneous elements that have to be present (and, potentially, understood), while being unrelated, or very indirectly related, to the concept we wish to illustrate with that except

  — **Syntactical noise** → *semicolons*

  — **Conceptual noise** → *need for* `class` *in Java*

Desideratum 1

hep/ EPFL

The first steps in a language (e.g., console print, variable assignment) should be painless and minimize the noise (syntactic and conceptual).

Desideratum 1

hep/ EPFL

CONCEPTUAL MODEL

The first steps in a language (e.g., console print, variable assignment) should be painless and minimize the noise (syntactic and conceptual).

2. Creating an associative data structure

hep/ EPFL

Want to exemplify:

- *Associative data structure definition*

Java

```
Map<Integer, String> numbers = new HashMap<>();
numbers.put(1, "one");
numbers.put(2, "two");
numbers.put(10, "ten");
```

Need to explain/ignore:

2. Creating an associative data structure

hep/ EPFL

Want to exemplify:

- *Associative data structure definition*

Java

```
Map<Integer, String> numbers = new HashMap<>();
numbers.put(1, "one");
numbers.put(2, "two");
numbers.put(10, "ten");
```

Need to explain/ignore:

- *Integer ≠ int*

2. Creating an associative data structure

hep/ EPFL

Want to exemplify:

- *Associative data structure definition*

Java

```
Map<Integer, String> numbers = new HashMap<>();
numbers.put(1, "one");
numbers.put(2, "two");
numbers.put(10, "ten");
```

Need to explain/ignore:

- *Integer ≠ int*
- *new*

2. Creating an associative data structure

hep/ EPFL

Want to exemplify:

- *Associative data structure definition*

Java

```
Map<Integer, String> numbers = new HashMap<>();
numbers.put(1, "one");
numbers.put(2, "two");
numbers.put(10, "ten");
```

Need to explain/ignore:

- *Integer ≠ int*
- *new*
- *Diamond*

2. Creating an associative data structure

hep/ EPFL

Want to exemplify:

- *Associative data structure definition*

Java

```
Map<Integer, String> numbers = new HashMap<>();
numbers.put(1, "one");
numbers.put(2, "two");
numbers.put(10, "ten");
```

Need to explain/ignore:

- *Integer ≠ int*
- *new*
- *Diamond*
- *Methods to define data*

2. Creating an associative data structure

hep/ EPFL

Want to exemplify:

- *Associative data structure definition*

Java

```
Map<Integer, String> numbers = new HashMap<>();
numbers.put(1, "one");
numbers.put(2, "two");
numbers.put(10, "ten");
```

Need to explain/ignore:

- *Integer ≠ int*
- *new*
- *Diamond*
- *Methods to define data*

Python

```
numbers = {
  1: "one",
  2: "two",
  10: "ten"
}
```

2. Creating an associative data structure

hep/ EPFL

Want to exemplify:

- *Associative data structure definition*

Java

```
Map<Integer, String> numbers = new HashMap<>();
numbers.put(1, "one");
numbers.put(2, "two");
numbers.put(10, "ten");
```

Need to explain/ignore:

- *Integer ≠ int*
- *new*
- *Diamond*
- *Methods to define data*

Python

```
numbers = {
  1: "one",
  2: "two",
  10: "ten"
}
```

- *More declarative approach*
- *Mental model easier*

Desideratum 2

Broadly used concepts sufficiently different from other concepts should have dedicated syntax (and not rely on existing syntactic constructs).

Desideratum 2

hep/ EPFL

CONCEPTUAL SEPARATION

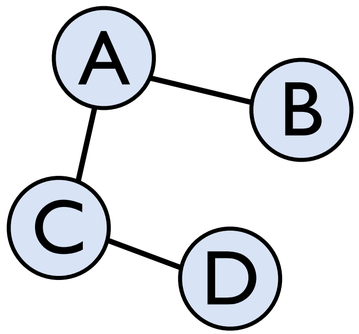
Broadly used concepts sufficiently different from other concepts should have dedicated syntax (and not rely on existing syntactic constructs).

3. Extracting code in functions/methods

hep/ EPFL

Want to exemplify:

- *1:n relation*
- *Extraction of repeated code in methods/functions*



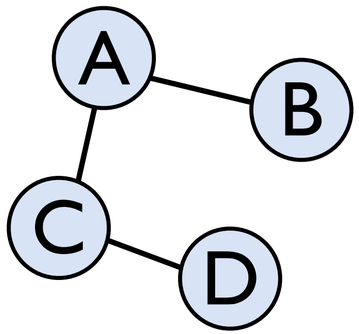
A → {B, C}
B → {A}
C → {A, D}
D → {C}

3. Extracting code in functions/methods

hep/ EPFL

Want to exemplify:

- *1:n relation*
- *Extraction of repeated code in methods/functions*



A → {B, C}
B → {A}
C → {A, D}
D → {C}

Python

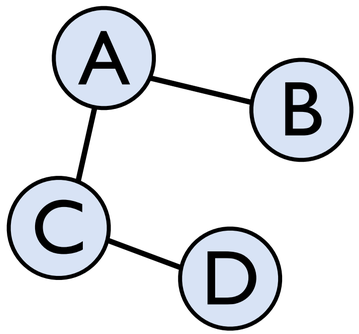
```
friendships = {}

add_friends("A", "B")
add_friends("A", "C")
add_friends("D", "C")

print(friendships)
```

3. Extracting code in functions/methods

hep/ EPFL

Want to exemplify:

- *1:n relation*
- *Extraction of repeated code in methods/functions*



A → {B, C}
B → {A}
C → {A, D}
D → {C}

Python

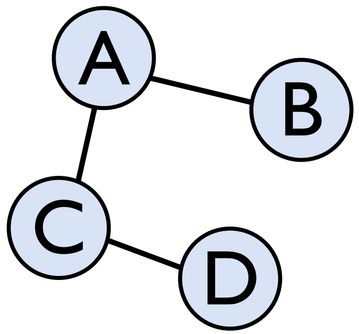
```
friendships = {}

def add_friends(name1, name2):

  def add_one_way(name1, name2):
    if name1 in friendships:
      friendships[name1].add(name2)
    else:
      friendships[name1] = {name2}

  add_one_way(name1, name2)
  add_one_way(name2, name1)

add_friends("A", "B")
add_friends("A", "C")
add_friends("D", "C")

print(friendships)
```

3. Extracting code in functions/methods

hep/ EPFL

Want to exemplify:

- *1:n relation*
- *Extraction of repeated code in methods/functions*



A → {B, C}
B → {A}
C → {A, D}
D → {C}

Python

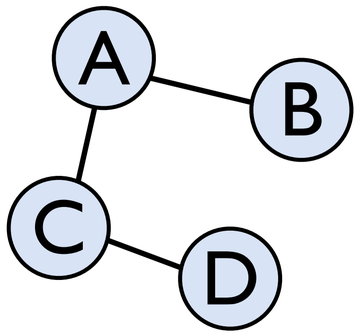
```python
friendships = {}

def add_friends(name1, name2):

    def add_one_way(name1, name2):
        if name1 in friendships:
            friendships[name1].add(name2)
        else:
            friendships[name1] = {name2}

    add_one_way(name1, name2)
    add_one_way(name2, name1)

add_friends("A", "B")
add_friends("A", "C")
add_friends("D", "C")

print(friendships)
```

3. Extracting code in functions/methods

hep/ EPFL

Want to exemplify:

- *1:n relation*
- *Extraction of repeated code in methods/functions*



A → {B, C}
B → {A}
C → {A, D}
D → {C}

Python

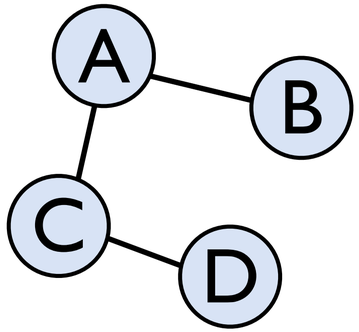
```
friendships = {}

def add_friends(name1, name2):
    def add_one_way(name1, name2):
        if name1 in friendships:
            friendships[name1].add(name2)
        else:
            friendships[name1] = {name2}

    add_one_way(name1, name2)
    add_one_way(name2, name1)

add_friends("A", "B")
add_friends("A", "C")
add_friends("D", "C")

print(friendships)
```

3. Extracting code in functions/methods

hep/ EPFL

Want to exemplify:

- *1:n relation*
- *Extraction of repeated code in methods/functions*



A → {B, C}
B → {A}
C → {A, D}
D → {C}

Python

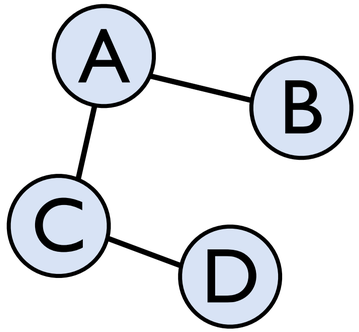
```
friendships = {}

def add_friends(name1, name2):

    def add_one_way(name1, name2):
        if name1 in friendships:
            friendships[name1].add(name2)
        else:
            friendships[name1] = {name2}

    add_one_way(name1, name2)
    add_one_way(name2, name1)

add_friends("A", "B")
add_friends("A", "C")
add_friends("D", "C")

print(friendships)
```

3. Extracting code in functions/methods

hep/ EPFL

Want to exemplify:

- *1:n relation*
- *Extraction of repeated code in methods/functions*



A → {B, C}
B → {A}
C → {A, D}
D → {C}

Java

```java
public class Friends {
  public static void main(String[] args) {
    Map<String, Set<String>> friendships =
        new HashMap<>();
    addFriends(friendships, "A", "B");
    addFriends(friendships, "A", "C");
    addFriends(friendships, "D", "C");
    System.out.println(friendships);
  }
  public static void addFriends(
        Map<String, Set<String>> friendships,
        String name1, String name2) {
    addOneWay(friendships, name1, name2);
    addOneWay(friendships, name2, name1);
  }
  public static void addOneWay(
        Map<String, Set<String>> friendships,
        String name1, String name2) {
    if (friendships.containsKey(name1)) {
      friendships.get(name1).add(name2);
    } else {
      Set<String> newSet = new HashSet<>();
      newSet.add(name2);
      friendships.put(name1, newSet);
    }
  }
}
```

Python

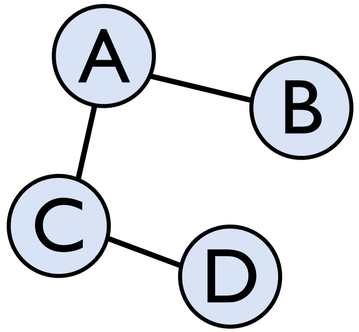
```python
friendships = {}

def add_friends(name1, name2):

    def add_one_way(name1, name2):
        if name1 in friendships:
            friendships[name1].add(name2)
        else:
            friendships[name1] = {name2}

    add_one_way(name1, name2)
    add_one_way(name2, name1)

add_friends("A", "B")
add_friends("A", "C")
add_friends("D", "C")

print(friendships)
```

3. Extracting code in functions/methods

hep/ EPFL

Want to exemplify:

- *1:n relation*
- *Extraction of repeated code in methods/functions*



A → {B, C}
B → {A}
C → {A, D}
D → {C}

Java

```java
public class Friends {
  public static void main(String[] args) {
    Map<String, Set<String>> friendships =
        new HashMap<>();
    addFriends(friendships, "A", "B");
    addFriends(friendships, "A", "C");
    addFriends(friendships, "D", "C");
    System.out.println(friendships);
  }
  public static void addFriends(
        Map<String, Set<String>> friendships,
        String name1, String name2) {
    addOneWay(friendships, name1, name2);
    addOneWay(friendships, name2, name1);
  }
  public static void addOneWay(
        Map<String, Set<String>> friendships,
        String name1, String name2) {
    if (friendships.containsKey(name1)) {
      friendships.get(name1).add(name2);
    } else {
      Set<String> newSet = new HashSet<>();
      newSet.add(name2);
      friendships.put(name1, newSet);
    }
  }
}
```

Python

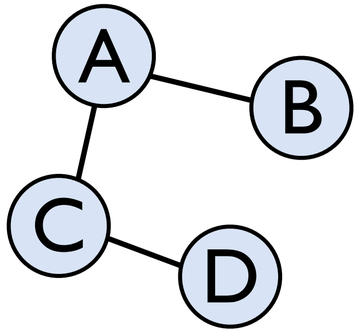
```python
friendships = {}

def add_friends(name1, name2):

    def add_one_way(name1, name2):
        if name1 in friendships:
            friendships[name1].add(name2)
        else:
            friendships[name1] = {name2}

    add_one_way(name1, name2)
    add_one_way(name2, name1)

add_friends("A", "B")
add_friends("A", "C")
add_friends("D", "C")

print(friendships)
```

3. Extracting code in functions/methods

hep/ EPFL

Want to exemplify:

- *1:n relation*
- *Extraction of repeated code in methods/functions*



A → {B, C}
B → {A}
C → {A, D}
D → {C}

Java

```java
public class Friends {
  public static void main(String[] args) {
    Map<String, Set<String>> friendships =
        new HashMap<>();
    addFriends(friendships, "A", "B");
    addFriends(friendships, "A", "C");
    addFriends(friendships, "D", "C");
    System.out.println(friendships);
  }
  public static void addFriends(
        Map<String, Set<String>> friendships,
        String name1, String name2) {
    addOneWay(friendships, name1, name2);
    addOneWay(friendships, name2, name1);
  }
  public static void addOneWay(
        Map<String, Set<String>> friendships,
        String name1, String name2) {
    if (friendships.containsKey(name1)) {
      friendships.get(name1).add(name2);
    } else {
      Set<String> newSet = new HashSet<>();
      newSet.add(name2);
      friendships.put(name1, newSet);
    }
  }
}
```

Python

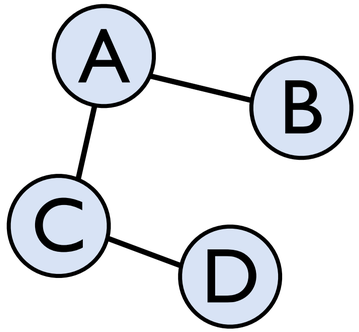
```python
friendships = {}

def add_friends(name1, name2):

    def add_one_way(name1, name2):
        if name1 in friendships:
            friendships[name1].add(name2)
        else:
            friendships[name1] = {name2}

    add_one_way(name1, name2)
    add_one_way(name2, name1)

add_friends("A", "B")
add_friends("A", "C")
add_friends("D", "C")

print(friendships)
```

3. Extracting code in functions/methods

hep/ EPFL

Want to exemplify:

- *1:n relation*
- *Extraction of repeated code in methods/functions*



A → {B, C}
B → {A}
C → {A, D}
D → {C}

Java

```java
public class Friends {
  public static void main(String[] args) {
    Map<String, Set<String>> friendships =
        new HashMap<>();
    addFriends(friendships, "A", "B");
    addFriends(friendships, "A", "C");
    addFriends(friendships, "D", "C");
    System.out.println(friendships);
  }
  public static void addFriends(
        Map<String, Set<String>> friendships,
        String name1, String name2) {
    addOneWay(friendships, name1, name2);
    addOneWay(friendships, name2, name1);
  }
  public static void addOneWay(
        Map<String, Set<String>> friendships,
        String name1, String name2) {
    if (friendships.containsKey(name1)) {
      friendships.get(name1).add(name2);
    } else {
      Set<String> newSet = new HashSet<>();
      newSet.add(name2);
      friendships.put(name1, newSet);
    }
  }
}
```

Python

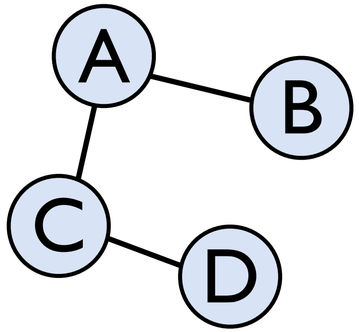
```python
friendships = {}

def add_friends(name1, name2):

    def add_one_way(name1, name2):
        if name1 in friendships:
            friendships[name1].add(name2)
        else:
            friendships[name1] = {name2}

    add_one_way(name1, name2)
    add_one_way(name2, name1)

add_friends("A", "B")
add_friends("A", "C")
add_friends("D", "C")

print(friendships)
```

3. Extracting code in functions/methods

hep/EPFL

Want to exemplify:

- *1:n relation*
- *Extraction of repeated code in methods/functions*



A → {B, C}
B → {A}
C → {A, D}
D → {C}

Java

```java
public class Friends {
  public static void main(String[] args) {
    Map<String, Set<String>> friendships =
        new HashMap<>();
    addFriends(friendships, "A", "B");
    addFriends(friendships, "A", "C");
    addFriends(friendships, "D", "C");
    System.out.println(friendships);
  }
  public static void addFriends(
        Map<String, Set<String>> friendships,
        String name1, String name2) {
    addOneWay(friendships, name1, name2);
    addOneWay(friendships, name2, name1);
  }
  public static void addOneWay(
        Map<String, Set<String>> friendships,
        String name1, String name2) {
    if (friendships.containsKey(name1)) {
      friendships.get(name1).add(name2);
    } else {
      Set<String> newSet = new HashSet<>();
      newSet.add(name2);
      friendships.put(name1, newSet);
    }
  }
}
```

Python

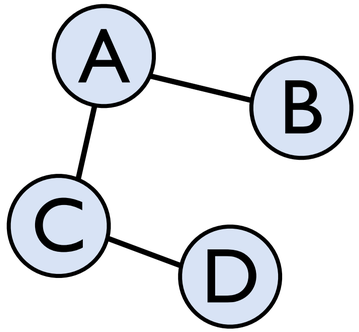
```python
friendships = {}

def add_friends(name1, name2):

    def add_one_way(name1, name2):
        if name1 in friendships:
            friendships[name1].add(name2)
        else:
            friendships[name1] = {name2}

    add_one_way(name1, name2)
    add_one_way(name2, name1)

add_friends("A", "B")
add_friends("A", "C")
add_friends("D", "C")

print(friendships)
```

3. Extracting code in functions/methods

hep/ EPFL

Want to exemplify:

- *1:n relation*
- *Extraction of repeated code in methods/functions*



A → {B, C}
B → {A}
C → {A, D}
D → {C}

Java

```java
public class Friends {
  public static void main(String[] args) {
    Map<String, Set<String>> friendships =
        new HashMap<>();
    addFriends(friendships, "A", "B");
    addFriends(friendships, "A", "C");
    addFriends(friendships, "D", "C");
    System.out.println(friendships);
  }
  public static void addFriends(
      Map<String, Set<String>> friendships,
      String name1, String name2) {
    addOneWay(friendships, name1, name2);
    addOneWay(friendships, name2, name1);
  }
  public static void addOneWay(
      Map<String, Set<String>> friendships,
      String name1, String name2) {
    if (friendships.containsKey(name1)) {
      friendships.get(name1).add(name2);
    } else {
      Set<String> newSet = new HashSet<>();
      newSet.add(name2);
      friendships.put(name1, newSet);
    }
  }
}
```

Python

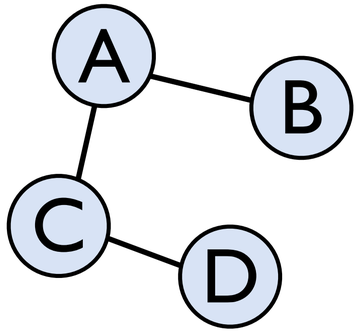
```python
friendships = {}

def add_friends(name1, name2):

    def add_one_way(name1, name2):
        if name1 in friendships:
            friendships[name1].add(name2)
        else:
            friendships[name1] = {name2}

    add_one_way(name1, name2)
    add_one_way(name2, name1)

add_friends("A", "B")
add_friends("A", "C")
add_friends("D", "C")

print(friendships)
```

3. Extracting code in functions/methods

hep/EPFL

Want to exemplify:

- *1:n relation*
- *Extraction of repeated code in methods/functions*



A → {B, C}
B → {A}
C → {A, D}
D → {C}

Java

```java
public class Friends {
  public static void main(String[] args) {
    Map<String, Set<String>> friendships =
        new HashMap<>();
    addFriends(friendships, "A", "B");
    addFriends(friendships, "A", "C");
    addFriends(friendships, "D", "C");
    System.out.println(friendships);
  }
  public static void addFriends(
      Map<String, Set<String>> friendships,
      String name1, String name2) {
    addOneWay(friendships, name1, name2);
    addOneWay(friendships, name2, name1);
  }
  public static void addOneWay(
      Map<String, Set<String>> friendships,
      String name1, String name2) {
    if (friendships.containsKey(name1)) {
      friendships.get(name1).add(name2);
    } else {
      Set<String> newSet = new HashSet<>();
      newSet.add(name2);
      friendships.put(name1, newSet);
    }
  }
}
```

Python

```python
friendships: Dict[str, Set[Str]] = {}

def add_friends(name1, name2):

    def add_one_way(name1, name2):
        if name1 in friendships:
            friendships[name1].add(name2)
        else:
            friendships[name1] = {name2}

    add_one_way(name1, name2)
    add_one_way(name2, name1)

add_friends("A", "B")
add_friends("A", "C")
add_friends("D", "C")

print(friendships)
```

Desideratum 3

Functions should be syntactically easily definable and quickly recognizable so as to allow convenient code factorization.

Desideratum 3

hep/ EPFL

MODULARIZATION
ABSTRACTION

Functions should be syntactically easily definable and quickly recognizable so as to allow convenient code factorization.

4. Simple compound data type

hep/ EPFL

Want to exemplify: *struct-like class*

Java

```
public class Rectangle extends BoardElement {
  public double width;
  public double height;
  public Rectangle(Point center,
         double width, double height) {
    super(center);
    this.width = width;
    this.height = height;
  }
  @Override public String toString() {
    return "Rectangle(c=" + center + ", w=" +
        width + ", h=" + height + ")";
  }
}

Rectangle r = new Rectangle(
    new Point(10.0, 10.0), 5.0, 2.0);
```

4. Simple compound data type

hep/ EPFL

Want to exemplify: *struct-like class*

Java

```
public class Rectangle extends BoardElement {
  public double width;
  public double height;
  public Rectangle(Point center,
         double width, double height) {
    super(center);
    this.width = width;
    this.height = height;
  }
  @Override public String toString() {
    return "Rectangle(c=" + center + ", w=" +
        width + ", h=" + height + ")";
  }
}

Rectangle r = new Rectangle(
    new Point(10.0, 10.0), 5.0, 2.0);
```

4. Simple compound data type

hep/ EPFL

Want to exemplify: *struct-like class*

Java

```java
public class Rectangle extends BoardElement {
  public double width;
  public double height;
  public Rectangle(Point center,
         double width, double height) {
    super(center);
    this.width = width;
    this.height = height;
  }
  @Override public String toString() {
    return "Rectangle(c=" + center + ", w=" +
        width + ", h=" + height + ")";
  }
}

Rectangle r = new Rectangle(
    new Point(10.0, 10.0), 5.0, 2.0);
```

Python

```python
class Rectangle(BoardElement):
  def __init__(self, center,
               width, height):
    BoardElement.__init__(self, center)
    self.width = width
    self.height = height

  def __repr__(self):
    return (
      f"Rectangle(c={self.center}, "
      f"w={self.width}, h={self.height})"
    )

r = Rectangle(
        Point(10.0, 10.0), 5.0, 2.0)
```

4. Simple compound data type

hep/ EPFL

Want to exemplify: *struct-like class*

Java

```java
public class Rectangle extends BoardElement {
  public double width;
  public double height;
  public Rectangle(Point center,
         double width, double height) {
    super(center);
    this.width = width;
    this.height = height;
  }
  @Override public String toString() {
    return "Rectangle(c=" + center + ", w=" +
        width + ", h=" + height + ")";
  }
}

Rectangle r = new Rectangle(
    new Point(10.0, 10.0), 5.0, 2.0);
```

Python

```python
class Rectangle(BoardElement):
  def __init__(self, center,
               width, height):
    BoardElement.__init__(self, center)
    self.width = width
    self.height = height

  def __repr__(self):
    return (
      f"Rectangle(c={self.center}, "
      f"w={self.width}, h={self.height})"
    )

r = Rectangle(
        Point(10.0, 10.0), 5.0, 2.0)
```

4. Simple compound data type

hep/ EPFL

Want to exemplify: *struct-like class*

Java

```java
public class Rectangle extends BoardElement {
  public double width;
  public double height;
  public Rectangle(Point center,
         double width, double height) {
    super(center);
    this.width = width;
    this.height = height;
  }
  @Override public String toString() {
    return "Rectangle(c=" + center + ", w=" +
        width + ", h=" + height + ")";
  }
}

Rectangle r = new Rectangle(
    new Point(10.0, 10.0), 5.0, 2.0);
```

Python

```python
class Rectangle(BoardElement):
  def __init__(self, center,
               width, height):
    BoardElement.__init__(self, center)
    self.width = width
    self.height = height

  def __repr__(self):
    return (
      f"Rectangle(c={self.center}, "
      f"w={self.width}, h={self.height})"
    )

r = Rectangle(
        Point(10.0, 10.0), 5.0, 2.0)
```

4. Simple compound data type

hep/ EPFL

Want to exemplify: *struct-like class*

Java

```
public class Rectangle extends BoardElement {
  public double width;
  public double height;
  public Rectangle(Point center,
         double width, double height) {
    super(center);
    this.width = width;
    this.height = height;
  }
  @Override public String toString() {
    return "Rectangle(c=" + center + ", w=" +
        width + ", h=" + height + ")";
  }
}

Rectangle r = new Rectangle(
    new Point(10.0, 10.0), 5.0, 2.0);
```

Python

```
class Rectangle(BoardElement):
  def __init__(self, center,
               width, height):
    BoardElement.__init__(self, center)
    self.width = width
    self.height = height

  def __repr__(self):
    return (
      f"Rectangle(c={self.center}, "
      f"w={self.width}, h={self.height})"
    )

r = Rectangle(
        Point(10.0, 10.0), 5.0, 2.0)
```

4. Simple compound data type

hep/ EPFL

Want to exemplify: *struct-like class*

Java

```java
public class Rectangle extends BoardElement {
  public double width;
  public double height;
  public Rectangle(Point center,
         double width, double height) {
    super(center);
    this.width = width;
    this.height = height;
  }
  @Override public String toString() {
    return "Rectangle(c=" + center + ", w=" +
        width + ", h=" + height + ")";
  }
}

Rectangle r = new Rectangle(
    new Point(10.0, 10.0), 5.0, 2.0);
```

Python

```python
class Rectangle(BoardElement):
  def __init__(self, center,
               width, height):
    BoardElement.__init__(self, center)
    self.width = width
    self.height = height

  def __repr__(self):
    return (
      f"Rectangle(c={self.center}, "
      f"w={self.width}, h={self.height})"
    )

r = Rectangle(
        Point(10.0, 10.0), 5.0, 2.0)
```

4. Simple compound data type

hep/ EPFL

Want to exemplify: *struct-like class*

Java

```java
public class Rectangle extends BoardElement {
  public double width;
  public double height;
  public Rectangle(Point center,
         double width, double height) {
    super(center);
    this.width = width;
    this.height = height;
  }
  @Override public String toString() {
    return "Rectangle(c=" + center + ", w=" +
        width + ", h=" + height + ")";
  }
}

Rectangle r = new Rectangle(
    new Point(10.0, 10.0), 5.0, 2.0);
```

Python

```python
class Rectangle(BoardElement):
  def __init__(self, center,
               width, height):
    BoardElement.__init__(self, center)
    self.width = width
    self.height = height

  def __repr__(self):
    return (
      f"Rectangle(c={self.center}, "
      f"w={self.width}, h={self.height})"
    )

r = Rectangle(
        Point(10.0, 10.0), 5.0, 2.0)
```

4. Simple compound data type

hep/ EPFL

Want to exemplify: *struct-like class*

Java

```java
public class Rectangle extends BoardElement {
  public double width;
  public double height;
  public Rectangle(Point center,
         double width, double height) {
    super(center);
    this.width = width;
    this.height = height;
  }
  @Override public String toString() {
    return "Rectangle(c=" + center + ", w=" +
        width + ", h=" + height + ")";
  }
}

Rectangle r = new Rectangle(
    new Point(10.0, 10.0), 5.0, 2.0);
```

Python

```python
class Rectangle(BoardElement):
  def __init__(self, center,
               width, height):
    BoardElement.__init__(self, center)
    self.width = width
    self.height = height

  def __repr__(self):
    return (
      f"Rectangle(c={self.center}, "
      f"w={self.width}, h={self.height})"
    )

r = Rectangle(
        Point(10.0, 10.0), 5.0, 2.0)
```

Haskell: `data Rectangle = Rectangle Point Float Float`

4. Simple compound data type

hep/ EPFL

Want to exemplify: *struct-like class*

Java

```java
public class Rectangle extends BoardElement {
  public double width;
  public double height;
  public Rectangle(Point center,
         double width, double height) {
    super(center);
    this.width = width;
    this.height = height;
  }
  @Override public String toString() {
    return "Rectangle(c=" + center + ", w=" +
        width + ", h=" + height + ")";
  }
}

Rectangle r = new Rectangle(
    new Point(10.0, 10.0), 5.0, 2.0);
```

Python

```python
class Rectangle(BoardElement):
  def __init__(self, center,
               width, height):
    BoardElement.__init__(self, center)
    self.width = width
    self.height = height

  def __repr__(self):
    return (
      f"Rectangle(c={self.center}, "
      f"w={self.width}, h={self.height})"
    )

r = Rectangle(
        Point(10.0, 10.0), 5.0, 2.0)
```

Haskell:
```haskell
data Rectangle = Rectangle Point Float Float
```

Scala:
```scala
case class Rectangle(
    center: Point, width: Double, height: Double)
```

4. Simple compound data type

hep/ EPFL

Want to exemplify: *struct-like class*

Java

```java
public class Rectangle extends BoardElement {
  public double width;
  public double height;
  public Rectangle(Point center,
         double width, double height) {
    super(center);
    this.width = width;
    this.height = height;
  }
  @Override public String toString() {
    return "Rectangle(c=" + center + ", w=" +
        width + ", h=" + height + ")";
  }
}

Rectangle r = new Rectangle(
    new Point(10.0, 10.0), 5.0, 2.0);
```

Python

```python
class Rectangle(BoardElement):
  def __init__(self, center,
               width, height):
    BoardElement.__init__(self, center)
    self.width = width
    self.height = height

  def __repr__(self):
    return (
      f"Rectangle(c={self.center}, "
      f"w={self.width}, h={self.height})"
    )

r = Rectangle(
        Point(10.0, 10.0), 5.0, 2.0)
```

Haskell:
```haskell
data Rectangle = Rectangle Point Float Float
```

Scala:
```scala
case class Rectangle(
    center: Point, width: Double, height: Double)
```

Python: `@dataclass`, `namedtuple`, `TypedDict`…

Desideratum 4

hep/ EPFL

Compound data types should be definable in a syntactically light way and should clearly allow the identification of their fields.

Desideratum 4

hep/ EPFL

MODELIZATION
ABSTRACTION

Compound data types should be definable in a syntactically light way and should clearly allow the identification of their fields.

5. Manipulating numerical data

hep/ EPFL

Want to exemplify: *operations on non-basic numeric types*

Java

```java
Vector3 v1 = new Vector3(1, 2, 3);
Vector3 v2 = new Vector3(4, 5, 6);
Vector3 v3 = v1.plus(v2).div(2);

// Given a class similar to this (with a proper
// toString() and equals() omitted here for brevity):
public class Vector3 {
  public final double x, y, z;
  public Vector3(double x, double y, double z) {
    this.x = x; this.y = y; this.z = z;
  }
  public Vector3 plus(Vector3 v) {
    return new Vector3(x + v.x, y + v.y, z + v.z);
  }
  public Vector3 div(double d) {
    return new Vector3(x / d, y / d, z / d);
  }
}
```

5. Manipulating numerical data

hep/ EPFL

Want to exemplify: *operations on non-basic numeric types*

Java

```
Vector3 v1 = new Vector3(1, 2, 3);
Vector3 v2 = new Vector3(4, 5, 6);
Vector3 v3 = v1.plus(v2).div(2);

// Given a class similar to this (with a proper
// toString() and equals() omitted here for brevity):
public class Vector3 {
  public final double x, y, z;
  public Vector3(double x, double y, double z) {
    this.x = x; this.y = y; this.z = z;
  }
  public Vector3 plus(Vector3 v) {
    return new Vector3(x + v.x, y + v.y, z + v.z);
  }
  public Vector3 div(double d) {
    return new Vector3(x / d, y / d, z / d);
  }
}
```

5. Manipulating numerical data

hep/ EPFL

Want to exemplify: *operations on non-basic numeric types*

Java

```java
Vector3 v1 = new Vector3(1, 2, 3);
Vector3 v2 = new Vector3(4, 5, 6);
Vector3 v3 = v1.plus(v2).div(2);

// Given a class similar to this (with a proper
// toString() and equals() omitted here for brevity):
public class Vector3 {
  public final double x, y, z;
  public Vector3(double x, double y, double z) {
    this.x = x; this.y = y; this.z = z;
  }
  public Vector3 plus(Vector3 v) {
    return new Vector3(x + v.x, y + v.y, z + v.z);
  }
  public Vector3 div(double d) {
    return new Vector3(x / d, y / d, z / d);
  }
}
```

Python

```python
from numpy import array

v1 = array([1, 2, 3])
v2 = array([3, 2, 1])
v3 = (v1 + v2) / 2
```

5. Manipulating numerical data

hep/ EPFL

Want to exemplify: *operations on non-basic numeric types*

Java

```java
Vector3 v1 = new Vector3(1, 2, 3);
Vector3 v2 = new Vector3(4, 5, 6);
Vector3 v3 = v1.plus(v2).div(2);

// Given a class similar to this (with a proper
// toString() and equals() omitted here for brevity):
public class Vector3 {
  public final double x, y, z;
  public Vector3(double x, double y, double z) {
    this.x = x; this.y = y; this.z = z;
  }
  public Vector3 plus(Vector3 v) {
    return new Vector3(x + v.x, y + v.y, z + v.z);
  }
  public Vector3 div(double d) {
    return new Vector3(x / d, y / d, z / d);
  }
}
```

hashCode()!

Python

```python
from numpy import array

v1 = array([1, 2, 3])
v2 = array([3, 2, 1])
v3 = (v1 + v2) / 2
```

Desideratum 5

Common arithmetic, equality, and comparison operators should work on compound data when relevant to ensure syntactic consistency with basic types.

Desideratum 5

hep/ EPFL

NUMERIC DATA MANIPULATION

Common arithmetic, equality, and comparison operators should work on compound data when relevant to ensure syntactic consistency with basic types.

Outline

1. Introduction & context
2. Comparison of excerpts & desiderata
3. **Conclusion**

Conclusion

- A pleasure not to have to defer explanations
- Code almost always shorter
- Data manipulation easier

  — *Couldn't help but change the Python course*

- Less time spent on lower-level details, more time for higher-level abstraction
- Same exam for all, no statistically significant difference

  — *But: the Python course taught more!*

Thank you!

hep/
EPFL

Q&A, or…

… guided tour & dinner!