

5. From Java to Python

First Python Program, Transfer Java → Python, Dynamic Data Structures in Python

Learning Objectives

- see a new programming language (Python) and learn how to transfer from one programming language to another
- learn the most important differences between Java and Python, both from a syntactical and semantical point of view
- learn about the basic data types of Python (list, set, dict, tuple) and operations leveraging the use of such data types
- get used to the new programming language and environment (Python) by re-implementing known algorithms

94

95

First Java Program

```
public class Hello {  
    public static void main (String[] args) {  
        System.out.print("Hello World!");  
    }  
}
```

96

First Python Program

```
print("Hello World!")
```

97

Comments

Comments are preceded by a #

```
# prints 'Hello World!' to the console
print("Hello World!")
```

Formatting Matters: Statements

- Whitespace is relevant
- Each line represents a statement
- So, *exactly one* Statement per line
- Comments start with #

Example program with two statements:

```
# two print-statements
print("Hurray, finally ...")
print("... no Semicolons!")
```

98

99

Formatting Matters: Blocks

- Blocks must be indented.
- All indented statements are part of a block. The block ends as soon as the indentation ends.
- Start of a Block is marked by a colon ":"

```
# in Python
while i > 0:
    x = x + 1 / i
    i = i - 1

print(x)
```

```
// in Java
while (i > 0) {
    x = x + 1.0 / i;
    i = i - 1;
}
System.out.print(x)
```

100

Literals: Numbers

- **integer**: 42, -5, 0x1b, 0o33, 7729684762313578932578932
Arbitrary precise integer numbers
- **float**: -0.1, 34.567e-4
Like **double** in Java, but precision depends on platform (CPU/
operating system)
- **complex**: 2 + 3j, (0.21 - 1.2j)
Complex numbers in the form $a+bj$. Optional round parentheses.

101

Literals: Booleans

- True
- False

Literals: Strings

- 'a single quoted string\nand a second line'
- "a double quoted string\nand a second line"
- Multi-line strings (triple double quotes):

```
"""a multiline string  
and a second line"""
```

102

103

Literals: Sequences

- *arrays*: There are no primitive arrays in Python
- **lists**: [17, True, "abc"], []
Mutable ordered sequence of 0 or more Values of arbitrary types.
- **tuples**: (17, True, "abc"), (42,)
Immutable ordered sequence of 1 or more Values of arbitrary types.

Literals: Collections

- **dicts**: { "a": 42, "b": 27, False: 0 }, {}
Mutable Key-Value store. Keys and values may have arbitrary types.
- **sets**: {17, True, "abc"}, {42}
Mutable unordered sequence of 0 or more Values of arbitrary types. No duplicates.

104

105

Variables

- Variables are automatically created upon the first assignment
- The type of a variable is not checked upon assignment. That is, values of different types can be assigned to a variable over time.
- Assignment of values with the assignment operator: =
- Assignment to multiple variables with tuples

```
a = "Ein Text"
print(a) # prints: Ein Text
a = 42
print(a) # prints: 42
```

```
x, y = 4, 5
print(x) # prints: 4
print(y) # prints: 5
```

106

Variables

- Variables must always be assigned first before it's possible to read their value

Assume b never got a value assigned:

```
a = b
```

Results in the following error

NameError: name 'b' is not defined

107

Numeric and Boolean Operators

- Numeric operators as in Java: +, -, *, /, %, **, //
- Caution: “ / ” always results in a floating-point number
- **: Power function, $a**b = a^b$.
- //: Integer division, $5//2$ results in 2.5.
- Comparison operators as in Java: ==, >=, <=, >, <, !=
- Logical Operators: `and`, `or`, `not`
- Membership Operator: “ `in` ” Determines if a value is in a list, set or string.
- Identity Operator: “ `is` ” Checks if two variables point to the same object.

108

Input/Output

- Reading of inputs using `input()`
- A prompt can be provided.
- Output using `print(...)`
- `print` accepts one or more arguments and prints them separated with a space

```
name = input("What is your name: ")
print("Hello", name)
```

109

Input/Output

- Input is always read as string
- To read a number, the input must be converted to a number first
- No implicit conversion happens
- Explicit conversion using:
`int()`, `float()`, `complex()`, `list()`, ...

```
i = int(input("Enter a number: "))
print("The", i, "th power of two is", 2**i)
```

110

Conditions

- No parentheses required around the test
- `elif` to test another case
- Mind the indentation!

```
a = int(input("Enter a number: "))
if a == 42:
    print("Naturally, the answer")
elif a == 28:
    print("A perfect number, good choice")
else:
    print(a, "is just some boring number")
```

111

While-Loops

The well-known Collatz-Folge

```
a = int(input("Enter a number: "))
while a != 1:
    if a % 2 == 0:
        a = a // 2
    else:
        a = a * 3 + 1
    print(a, end=' ')
```

112

For-Loops

- For-Loops work differently than in Java
- Iterates over the elements of the given set

```
some_list = [14, 'lala', 22, True, 6]
total = 0;
for item in some_list:
    if type(item) == int:
        total += item
print("Total of the numbers is", total)
```

113

For-Loops over a value range

- The function `range(start, end, step)` creates a list of values, starting with `start` until `end` - exclusive. Stepsize is `step`.
- Step size is 1 if the third argument is omitted.

```
# the following loop prints "1 2 3 4"
for i in range(1,5):
    print(i, end=' ')

```

```
# the following loop prints "10 8 6 4 2"
for i in range(10, 0, -2):
    print(i, end=' ')

```

Methods

- The Cookie Calculator revisited

```
def readInt(prompt, atleast = 1):
    """Prompt for a number greater 0 (or min, if specified)"""
    number = 0;
    while number < atleast:
        number = int(input(prompt))
        if (number < atleast):
            print("Too small, pick a number larger than", atleast)
    return number

```

```
kids = readInt("Kids: ")
cookies = readInt("Cookies: ", atleast=kids)
print("Each Kid gets", cookies // kids, "cookies.")
print("Papa gets", cookies % kids, "cookies.")

```

114

115

Lists: Basic Operations

- Element-Access (0-based): `a[2]` points to the third element.
- Negative indices count from the last element!

```
a = [ 3, 7, 4]
print(a[-1]) # prints '4'

```

- Add value to the tail: `a.append(12)`
- Test if an element is in a collection:

```
if 12 in a:
    print('12 is in the list, we just added it before')

```

- Anzahl Elemente in einer Collection: `len(a)`

116

Lists: Slicing

- Slicing: address partition: `a[start:end]`
- `a` and/or `b` are positive or negative indices.
- `end` is not inclusive

```
a = [ 1, 2, 3, 4, 5, 6, 7, 8, 9]
print(a[2:4])    # [3, 4]
print(a[3:-3])  # [4, 5, 6]
print(a[-3:-1]) # [7, 8]
print(a[5:])    # [6, 7, 8, 9]
print(a[:3])    # [1, 2, 3]

```

117

Dictionaries

Dictionaries are very important primitive data structures in Python

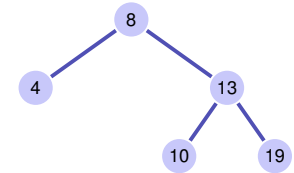
- Easy and efficient possibility to name and group several fields of data
- Build hierarchical data structures by nesting
- Accessing elements using [] Operator

```
record = { 'Firstname': 'Hermann', 'Lastname': 'Lehner',
          'Salary': 420000, 'Mac User': True }
record['Salary'] = 450000
if record['Mac User']:
    print('... one more thing!')
```

118

Dynamic Data Structures with Dicts

```
tree = {
    'key': 8,
    'left' : {
        'key': 4, 'left' : None, 'right': None
    },
    'right': {
        'key': 13,
        'left' : {
            'key': 10, 'left' : None, 'right': None
        },
        'right': {
            'key': 19, 'left' : None, 'right': None
        }
    }
}
```



119

Dynamic Data Structures with Dicts

- Working with Dicts (Examples)

```
l = tree['left'] # assign left subtree to variable l
l['key'] = 6     # changes key from 4 to 6

if l['left'] is None: # proper way to test against None
    print("There is no left child here...")
else:
    print("Value of left subtree is", l['left']['key'])
```

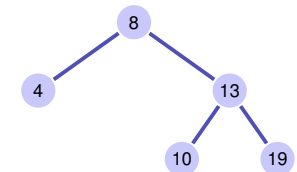
120

Dynamic Data Structures with Classes

```
class Node:
    def __init__(self, k, l=None, r=None):
        self.key, self.left, self.right = k, l, r

# create the tree depicted on the right
rightSubtree = Node(13, l=Node(10), r=Node(19))
tree = Node(8, l=Node(4), r=rightSubtree)

# an example query
print(tree.right.right.key) # prints: 19
```



121

Modules

Python has a vast amount of libraries in form of modules that can be imported.

- Importing a whole module:

```
import math  
x = math.sqrt(4)
```

```
from math import *  
x = sqrt(4)
```

- Importing parts of a module:

```
from datetime import date  
t = date.today()
```