Introduction to Programming

Christoph Knorr

Aims of this course

- Give you an overview of the world of programming
- To explain basic principles like variables, branching, loops and functions
- Show you some examples how to illustrate your source code

Contents

What is programming?

- “Programming is the formulation of a problem in the form of order instructions with the help of a language, which is translated into commands, that can be executed by a computer.“

Why no intuitive language?

- A computer “thinks” different than a human
- Computers are dumb – they can only do exactly as they’re said
- Languages always try to get even more intuitive ➔ languages are more difficult to read

Example of a program


```
def Dihedral(xyz1,xyz2,xyz3,xyz4):
    return (360/(2*math.pi))*BP.calc_dihedral(BP.Vector(xyz1.iloc[0,:]),BP.Vector(xyz2.ilo

Models = ['ModBase','SwissModel','T0863','T0872','T0886','T0892','T0917','T0944']
Strucs = ['Single_Test','Single_Training','XRay5000']
Dirs = ['ModBase','SwissModel','XRay5000']
AA = ['ALA','ARG','ASN','ASP','CYS','GLN','GLU','HIS','ILE','LEU','LYS','MET','PHE','PRO',
Output = pd.DataFrame(data=0,index=AA+['All','Strucs'],columns=Dirs)

for Dir in Dirs:
    os.chdir(Dir)
    if Dir in Models: Files = glob.glob('*.pdb')
    else: Files = glob.glob('*_clean.pdb')
    for F in Files:
        Count = 0
        ppdb = PandasPDB()
        Struc = ppdb.read_pdb(F)
        Chains = list(set(Struc.df['ATOM']['chain_id']))
        Chains.sort()
        Numbers = list(set(Struc.df['ATOM'][Struc.df['ATOM']['chain_id'] == Chains[0]]['re
        Numbers.sort()
        for Nr in Numbers:
            Amino = list(set(Struc.df['ATOM'][(Struc.df['ATOM']['residue_number'] == Nr) &
            if Amino in AA:
                C = Struc.df['ATOM'][(Struc.df['ATOM']['atom_name'] == 'C') & (Struc.df['A
                CA = Struc.df['ATOM'][(Struc.df['ATOM']['atom_name'] == 'CA') & (Struc.df[
                N = Struc.df['ATOM'][(Struc.df['ATOM']['atom_name'] == 'N') & (Struc.df['A
                CB = Struc.df['ATOM'][(Struc.df['ATOM']['atom_name'] == 'CB') & (Struc.df[
                if C.size > 1 and CA.size > 1 and N.size > 1 and CB.size > 1:
                    Zeta = Dihedral(CA,N,C,CB)
                    if pd.notnull(Zeta):
                        if Zeta < 0:
                            Output.loc[Amino,Dir] += 1
                            Count += 1
                            Output.loc['All',Dir] += 1
        if Count > 0: Output.loc['Strucs',Dir] += 1
```

Desktop Applications

Desktop Applications

- Everything that is an executable programm on a computer
- E.g. Spreadsheets, eMail-programs, Webbrowser, Text editiors etc.
- Can either be executed by themselves (binary) or with the help of a compiler (Scripting languages)

Desktop Applications - Pro

- Biggest pool of resources in all field
- Make it easy to reach a good “user performance“
- Nearly gapless support through the internet
- The most flexible for the developer

Desktop Applications - Contra

- Need for installation
- Maintenance has to be done regularly (bugfixes, updates, etc.)
- Difficult to transfer between different systems

Programs with GUIs

- GUI = graphical user interface:
  - Inputs, outputs, setting parameters, data manipulations, handling of the program ins done via a graphical interface

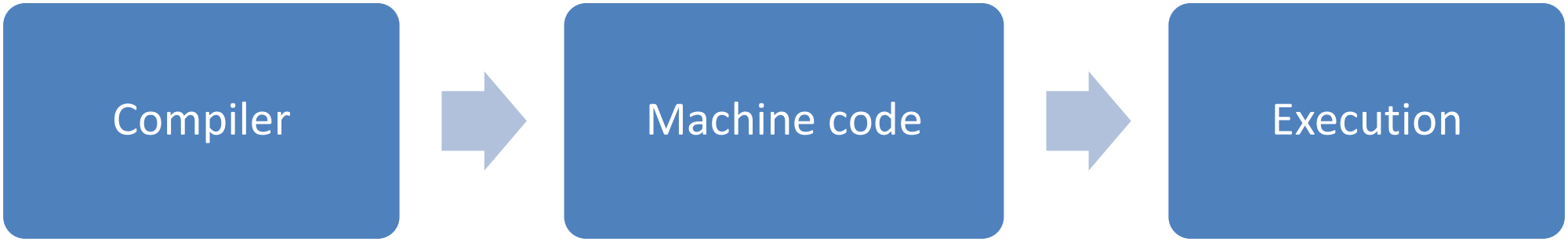
Example: Powerpoint

Console programs

- Programs are run exclusively from the console
- Parameters have to be set in text form at the start of the program
- Most small tools, and even some bigger programs, in bioinformatics are console programs

From Source code to program

From Source code to program



Language

- Essential form of communication
- Consists of specific letters (alphabet)
- Grammar defines membership of a language
- Sentence structure is determined via syntax
- Semantics is language independent – describes the contents of the spoken

Programing language

- Hand over commands to the compiler
- Programing languages differ in their syntax and their powerfulness
- Are explicit
- Aren‘t spoken
- Pseudocode is used as communication between people

Compiler and Interpreter language

- Interpreter **don‘t** generate executable programs (binarys)
- Source code is operating system dependent, compiler aren‘t
- Languages, that generate executable programs themself, can‘t be used for reverse engineering

Syntax

- Dictates if a command is recognized as a part of the language
- Syntax errors prevent the translation of the source code into machine code
- Are detected by the compiler
- Use gedit or atom to prevent easy syntax errors

Semantics

- Refers to the contents of the commands
- Semantics errors can only be found by the programer and not by the compiler
- Often hard to spot
  - Plan everything as good as possible **prior to** starting with the programing

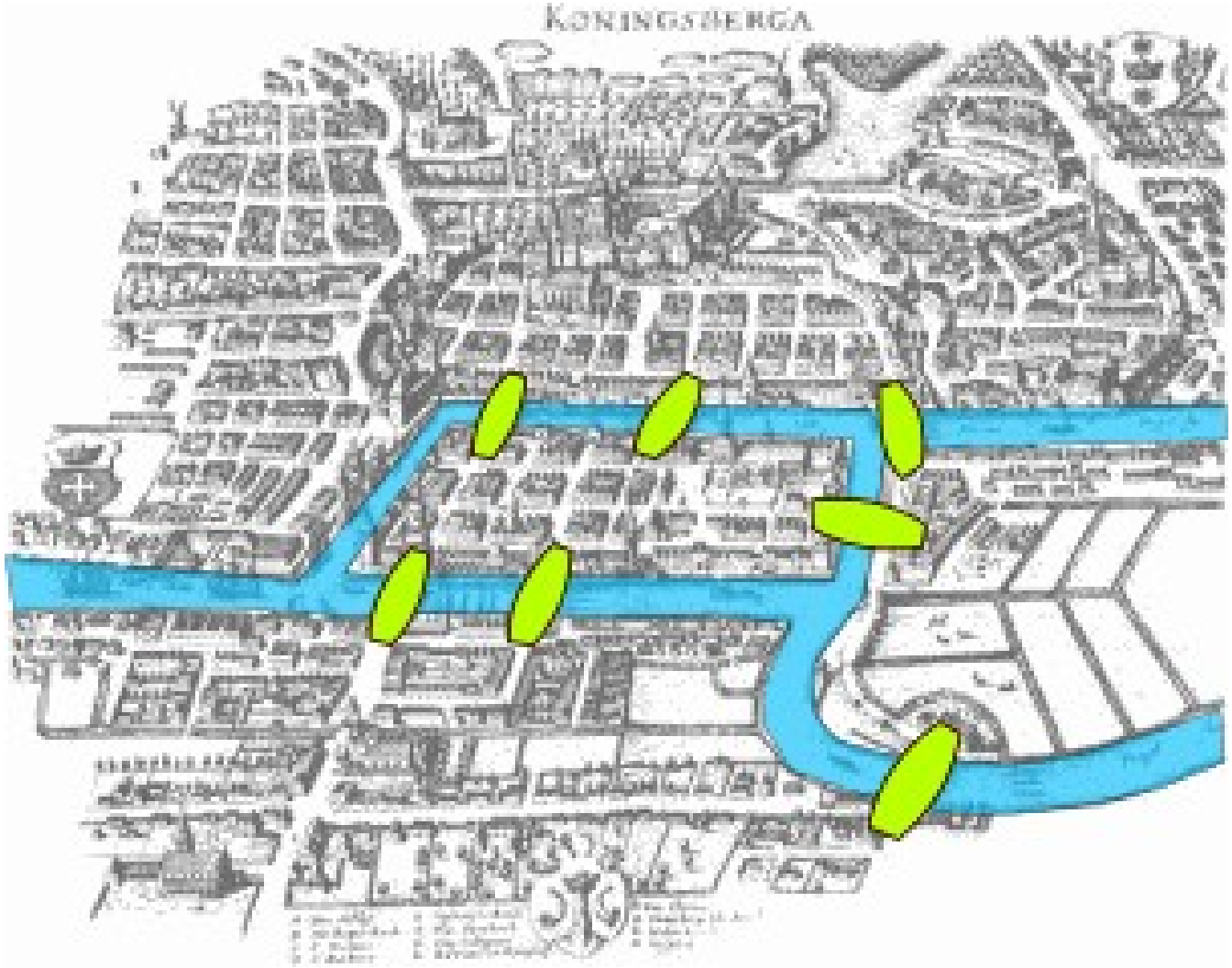
Algorithmic

- An algorithm is a complex of related commands, that serve a specific purpose
- Algorithms are instructions for the systematic solution of a problem
- Tries to guarantee good or best possible results

Known problems

- 7 Bridges of Königsberg
- Travelingsalesman(/person) problem
- Binary search
- 8 Queens problem

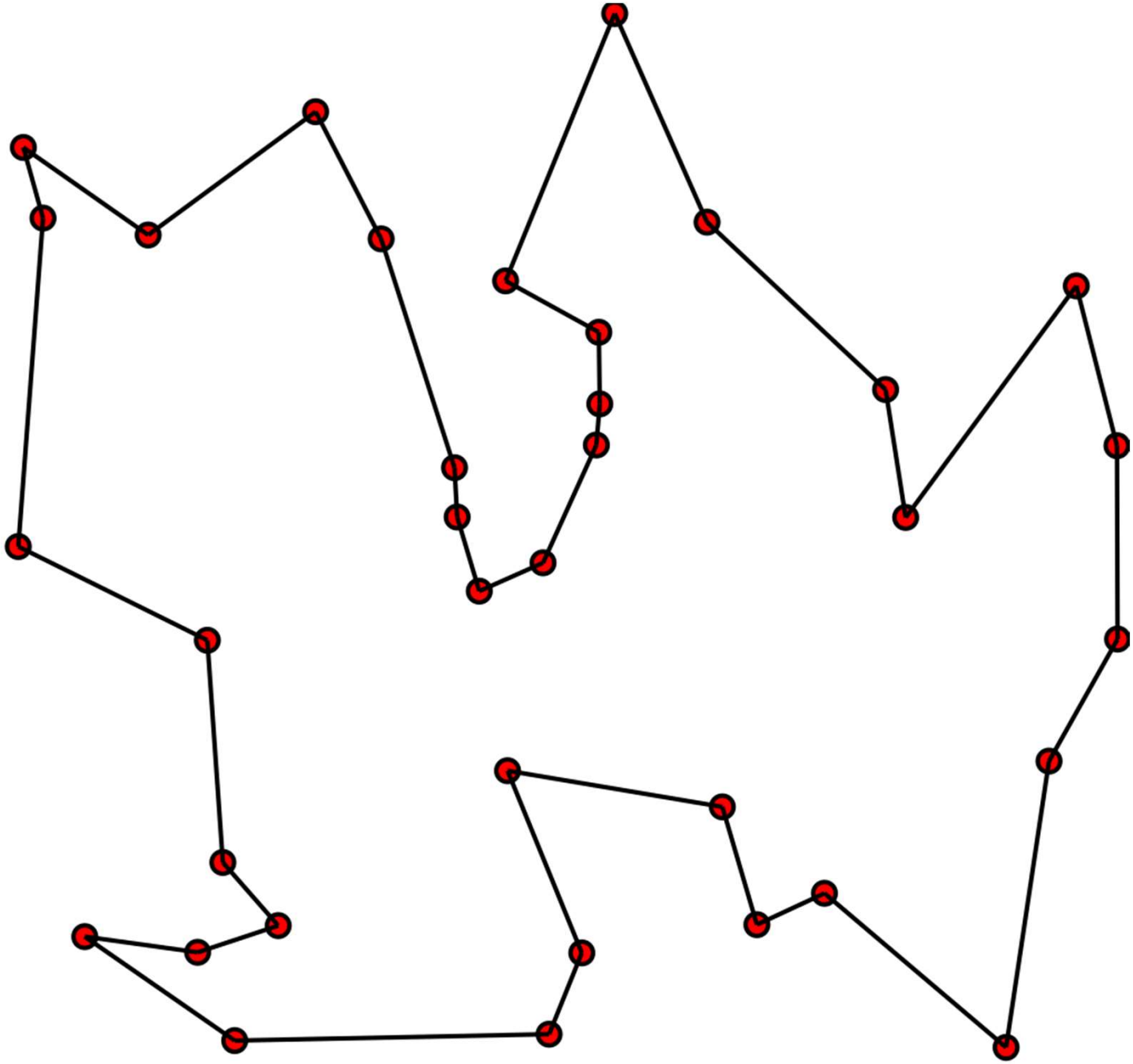
7 Bridges of Königsberg [1]



7 Bridges of Königsberg [1]

- Historical mathematical problem
- Negative solution was found by Leonhard Euler in 1736
- Can you travel through the city of Königsberg by traversing each of the 7 bridges exactly one time
- Euler‘s solution laid the basis of the graph theory

Travelingsalesman(/person) problem [2]



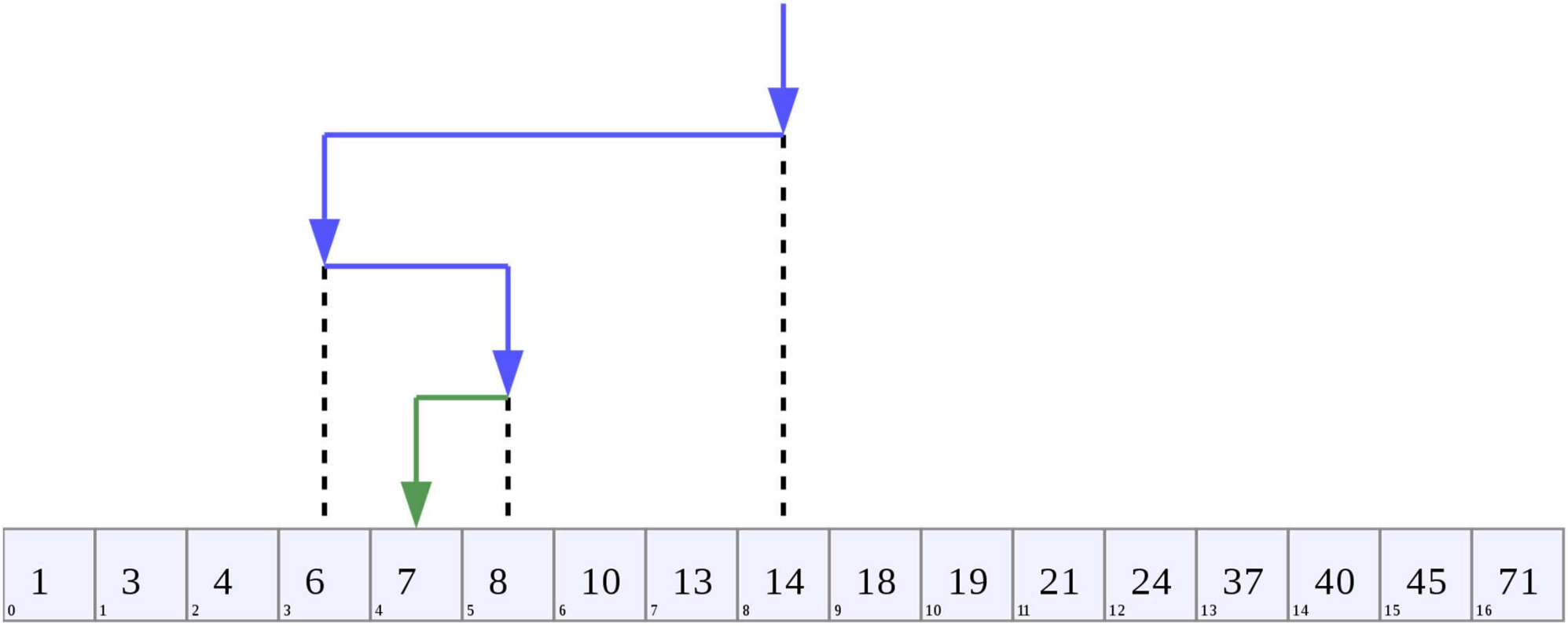
Travelingsalesman(/person) problem [2]

- You have a list of cities and the given distances between them
- What is the shortest route do visit each city exactly once and come back to the starting city
- This problem was formulated in 1930 and is a benchmark for different optimization processes

Binary search [3]

- Also called half-interval or logarithmic search
- Searching an array for a specific entry
- The target is compared to the middle value of the array
  - If these values differ, the part of the array in which the target can‘t be found is discarded and the process is started over with the remaining array
  - Remains an empty array at the end of the algorithm, the target wasn‘t in the array

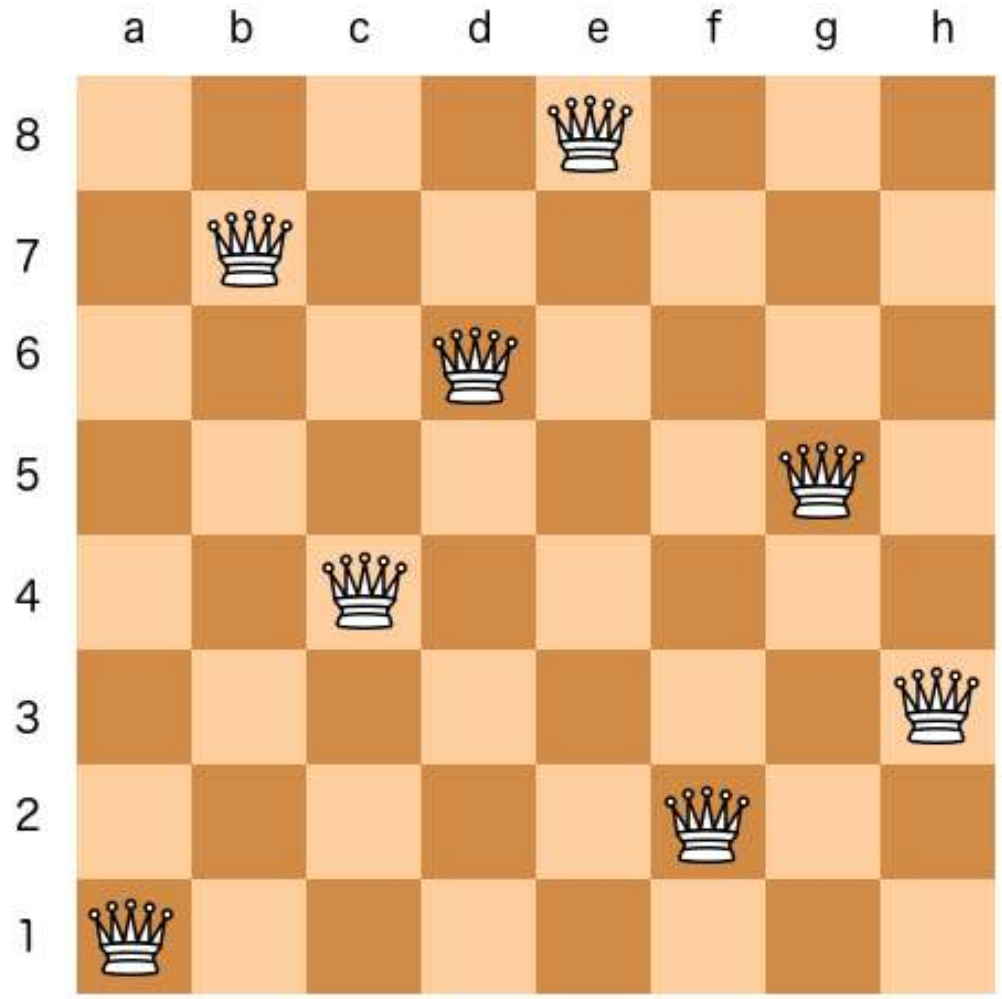
Binary search [3]



8 Queens problem [4]

- You have to place 8 queens on a chess board with the dimensions 8x8
  - No queen is allowed ot be in danger of another queen
  - No two quuens are allowed in the same row, column or diagonal
  - Was published by Max Bezzel in 1848 and the first solution was published by Franz Nauck in 1850, who applied the problem even further for n queens in nxn dimensions
- 4,426,165,368 possible arrangements of eight queens on an 8×8 board, but only 92 solutions

8 Queens problem [4]



Punctuation in programming

- Normal sentences end with a period
- For computers instructions end with special characters
- Sentences are ordered with the help of paragraphs in normal language
- Instructions on the computer are structured by blocks

Work of a compiler

- The translation of the source code into machine code is called compiling
- The compiler walks through the code line by line
- Jumps are possible – we will learn more about this than we speak about branchings

A cooking recipe

- Bake the cake in the oven at 180°C for 20 minutes

Computers are dumb

- For a human these instructions are clear
- … but not for a PC
- Computers need much more precise instructions
- If not:


90

Baking with a PC

- Bake the cake in the oven at 180°C for 20 minutes
  - There is no clarification that the door of the oven has to be opened beforehand
  - In the worst case scenario the door of the oven is damaged or the cake (“corrupted”)
  - For simple programs this is rather rare

Baking with a PC

- Open the door of the oven
- Put the cake into the oven
- Close the door of the oven
- Bake the cake in the oven at 180°C for 20 minutes
  - The computer could stop the program if the door is already open

Baking with a PC

- If the oven door is closed, open the oven door and put the cake into the oven
- When not put the cake directly into the oven
- Close the door of the oven
- Bake the cake in the oven at 180°C for 20 minutes
  - This runs even if something is still in the oven. If there is an old cake in the oven it could catch fire

The complete recipe

- If the oven door is closed, open the oven door and when the oven is empty put the cake into the oven. If not empty the oven and put the cake into the oven
- When not put the cake directly into the oven if the oven is empty. If not empty the oven and put the cake into the oven afterwards
- Close the door of the oven
- Bake the cake in the oven at 180°C for 20 minutes

Logical sequence

Door open?
- If YES:
  - Oven emtpy?
    - if YES:
      - put the cake into the oven
    - if NO:
      - empty the oven
      - put the cake into the oven
- if NO:
  - open the oven
  - Oven emtpy?
    - if YES:
      - put the cake into the oven
    - if NO:
      - empty the oven
      - put the cake into the oven

Bake the cake in the oven at 180°C for 20 minutes.

First Pseudocode


```
IF (isOvenOpen())
        IF (isOvenEmpty())
                putCakeInOven();
        ELSE
                emptyOven();
                putCakeInOven();
ELSE
        openOven();
        IF (isOvenEmpty())
                putCakeInOven();
        ELSE
                emptyOven();
                putCakeInOven();
closeOven()
bakeCake(180,20)
```

Think beforehand

- Prior to saving a file – is there enough memory?
- Prior to opening a file – does this file exist?
- Prior to working with a variable –
  - is the variable declared?
  - has the variable the right type?
- Etc…

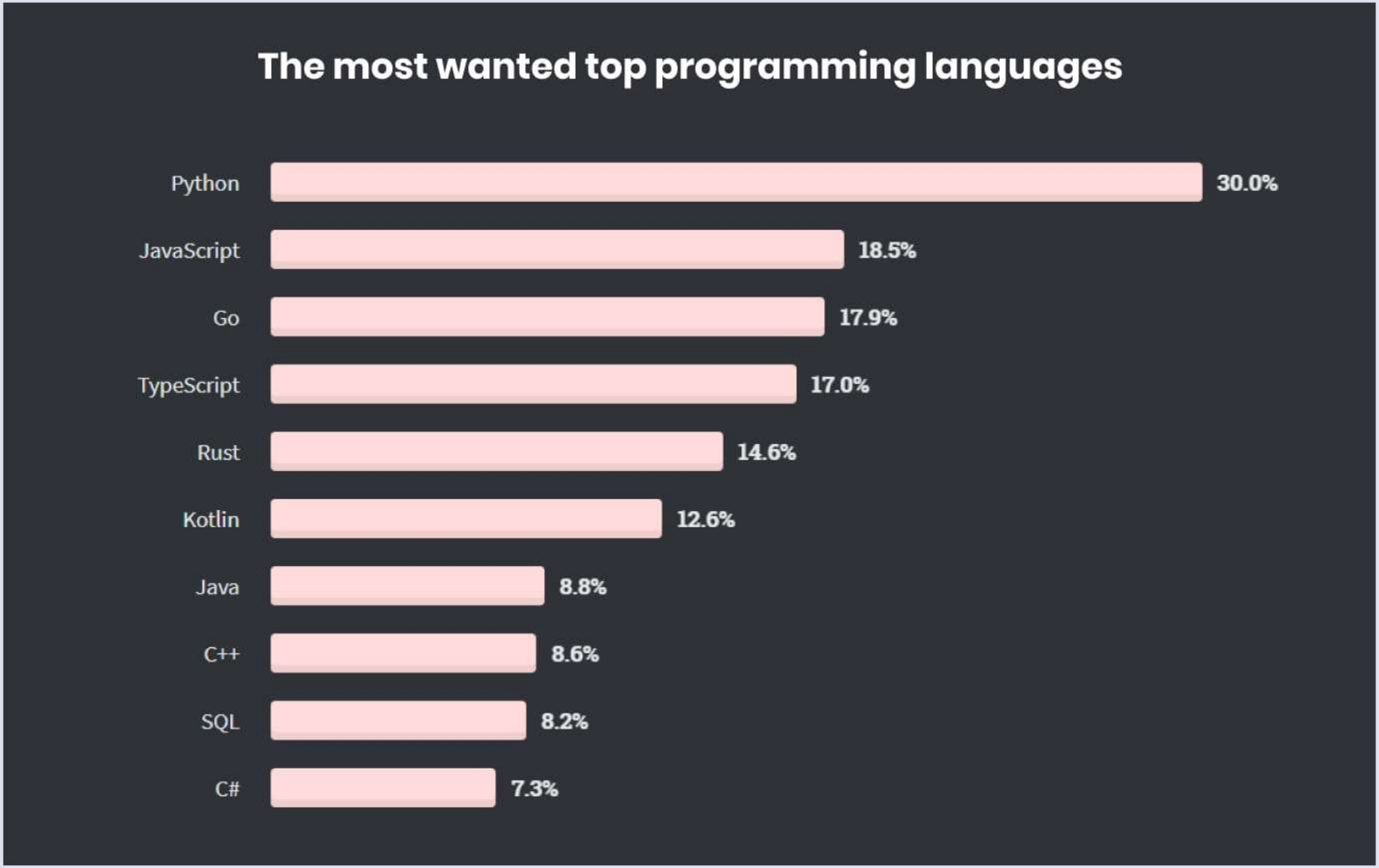
History [5]

- Developed by Guido van Rossum in the early 90's
- As successor of the language ABC
- Name inspired by Monty Python
- First full version in 1994
- Since 2008 two different versions 2.7.18 and 3.10.0
- Python 2 is no longer supported

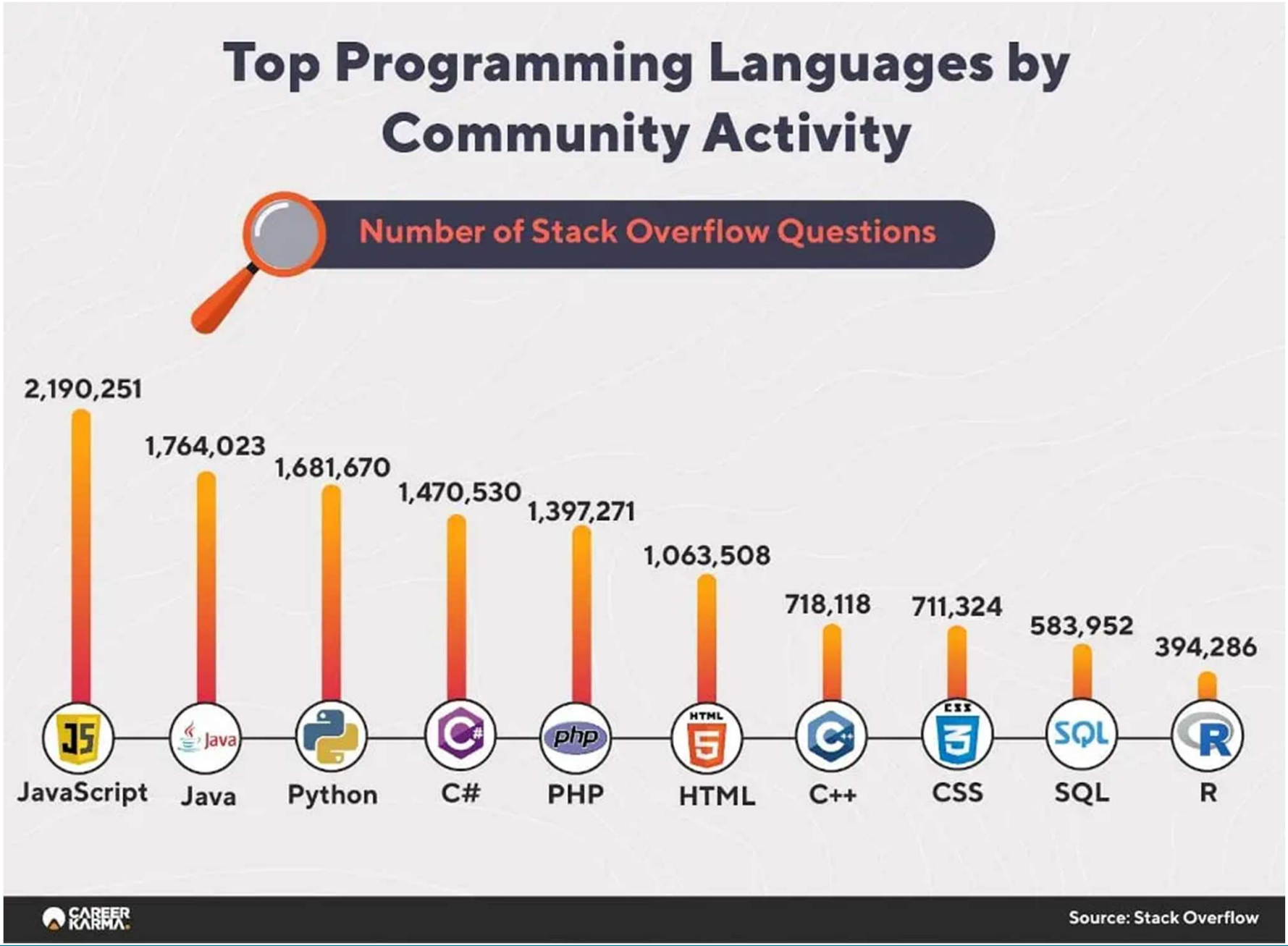
Why Python [5]

- Minimalistic language
- Full range of object orientation
- Interactive mode (Python shell)
- Many (scientific) modules
- Non-commercial
- Big, active community (Stack Overflow)

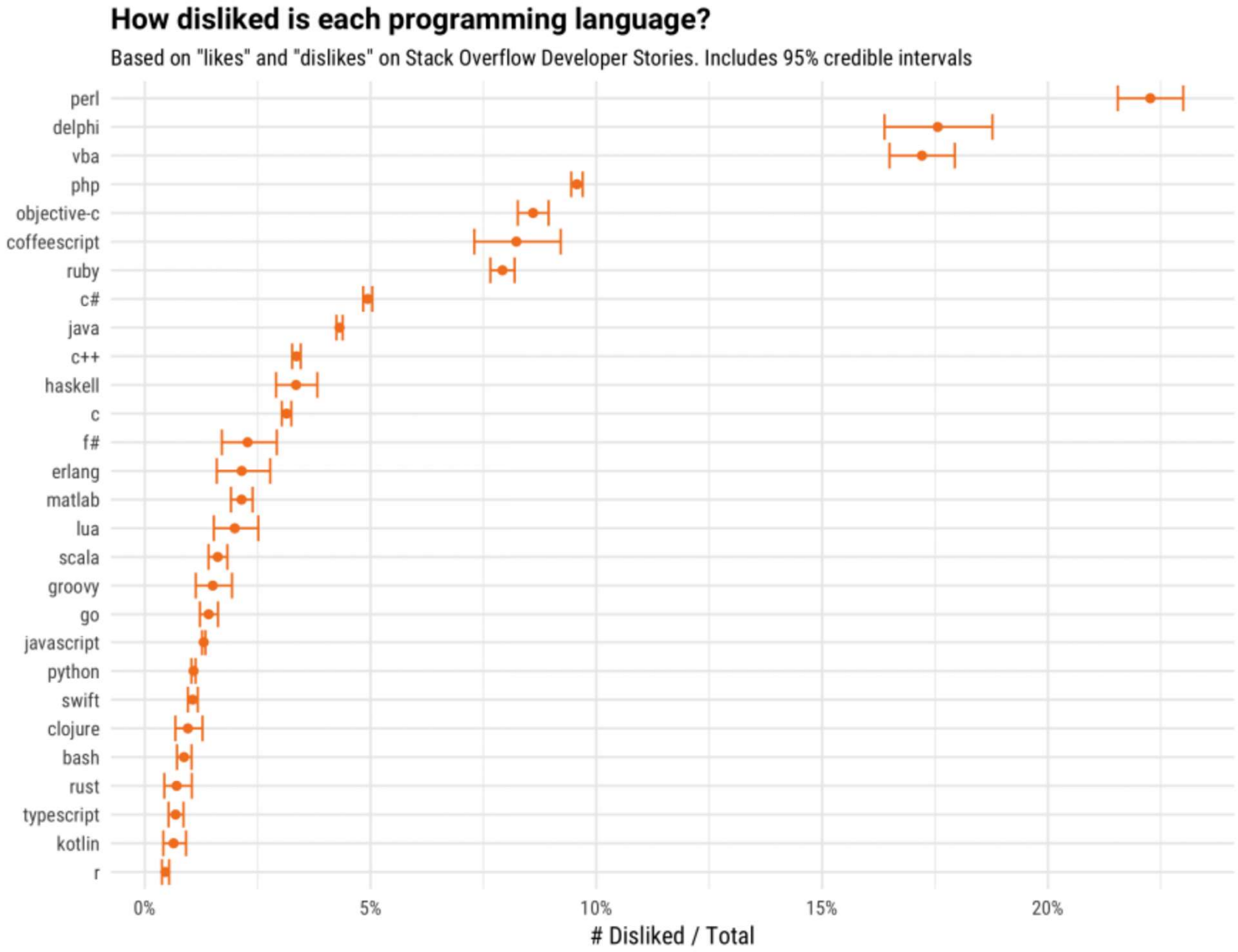
Why Python [6]


The most wanted top programming languages
Python
30.0%
JavaScript
18.5%
Go
17.9%
TypeScript
17.0%
Rust
14.6%
Kotlin
12.6%
Java
8.8%
C++
8.6%
SQL
8.2%
C#
7.3%

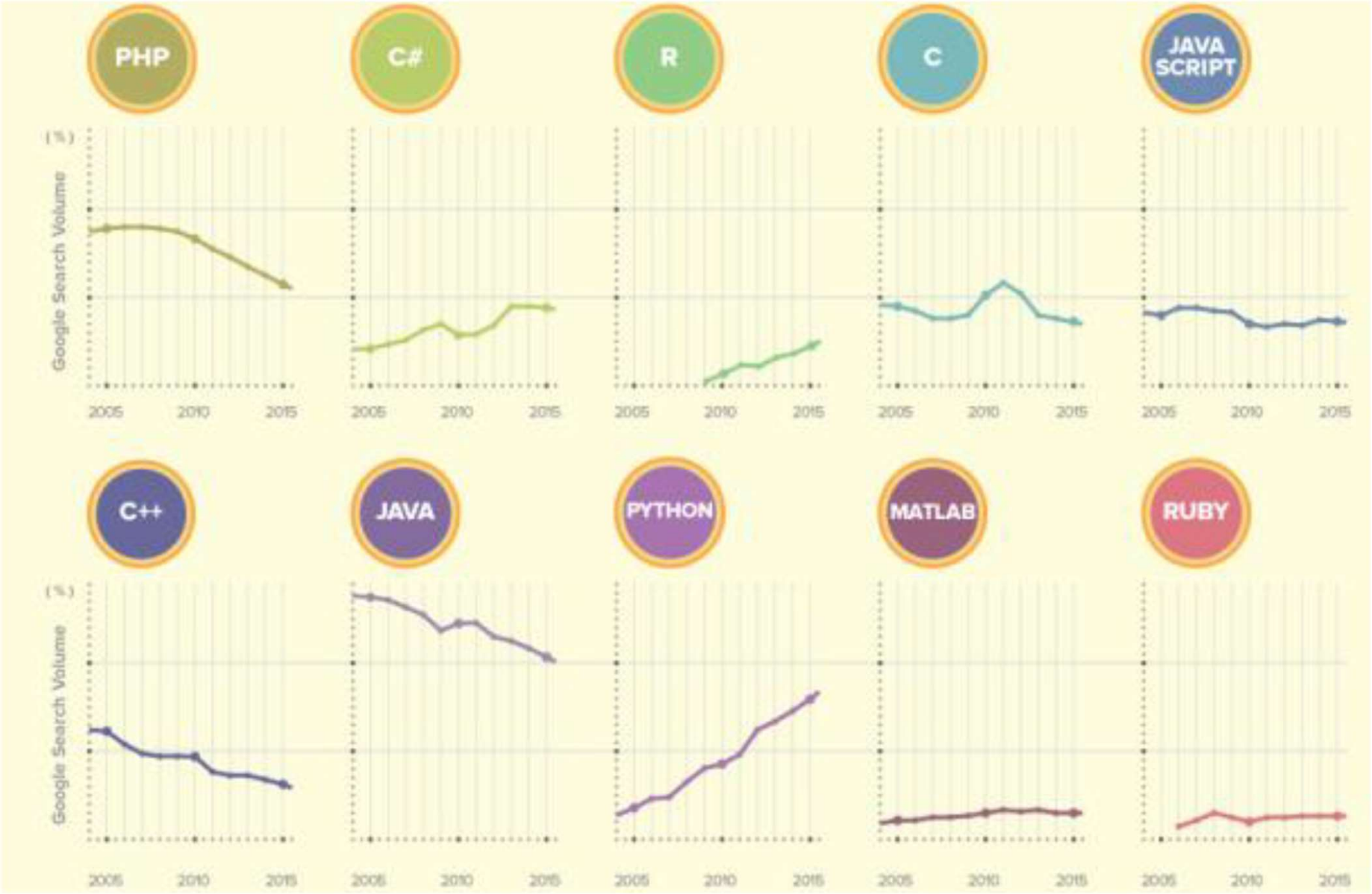
Why Python [7]


Top Programming Languages by Community Activity
Number of Stack Overflow Questions
2,190,251
1,764,023
1,681,670
1,470,530
1,397,271
1,063,508
718,118
711,324
583,952
394,286
JavaScript
Java
Python
C#
PHP
HTML
C++
CSS
SQL
R
CAREER KARMA
Source: Stack Overflow

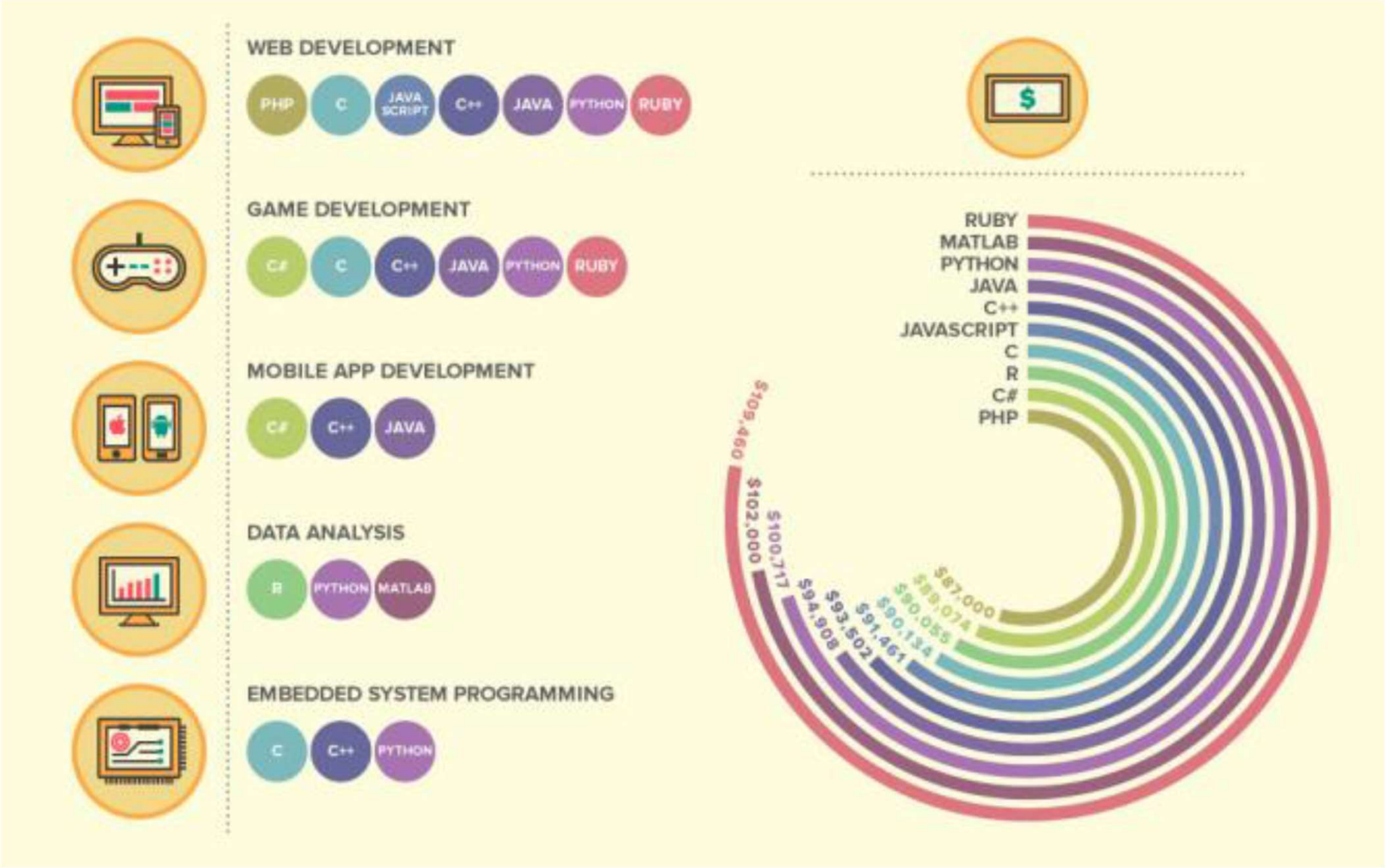
Why Python [5]


How disliked is each programming language?
Based on "likes" and "dislikes" on Stack Overflow Developer Stories. Includes 95% credible intervals
perl
delphi
vba
php
objective-c
coffeescript
ruby
c#
java
c++
haskell
c
f#
erlang
matlab
lua
scala
groovy
go
javascript
python
swift
clojure
bash
rust
typescript
kotlin
r
0%
5%
10%
15%
20%
Disliked / Total

Why Python [5]


PHP
C#
R
C
JAVA SCRIPT
C++
JAVA
PYTHON
MATLAB
RUBY
Google Search Volume
(%)
2005
2010
2015

Why Python [5]


WEB DEVELOPMENT
PHP
C
JAVA SCRIPT
C++
JAVA
PYTHON
RUBY
GAME DEVELOPMENT
C#
C
C++
JAVA
PYTHON
RUBY
MOBILE APP DEVELOPMENT
C#
C++
JAVA
DATA ANALYSIS
R
PYTHON
MATLAB
EMBEDDED SYSTEM PROGRAMMING
C
C++
PYTHON
$
RUBY
MATLAB
PYTHON
JAVA
C++
JAVASCRIPT
C
R
C#
PHP
$105,490
$102,000
$100,717
$94,908
$93,502
$91,461
$90,134
$89,074
$87,000

Python characteristics [5]

- Whitespace sensitive
- Object orientated (everything is an object)
- Typed and interpreted dynamically
- Functional
- Aspect orientated
- Interactive (ad-hoc scripts)
- Automatic memory management
- Easy readable
- “There should be one, and preferably only one, obvious way to do it.”

Zen of Python [5]

- Beautiful is better than ugly.
- Explicit is better than implicit.
- Simple is better than complex.
- Complex is better than complicated.
- Flat is better than nested.
- Sparse is better than dense.
- Readability counts.
- Special cases aren't special enough to break the rules.
- Although practicality beats purity.
- Errors should never pass silently.

Zen of Python [5]

- Unless explicitly silenced.
- In the face of ambiguity, refuse the temptation to guess.
- There should be one and preferably only one obvious way to do it.
- Although that way may not be obvious at first unless you're Dutch.
- Now is better than never.
- Although never is often better than right now.
- If the implementation is hard to explain, it's a bad idea.
- If the implementation is easy to explain, it may be a good idea.
- Namespaces are one honking great idea let's do more of those!

Python versions

- We will use Python3 (to be more precise Python 3.9) in this course
- As already said there is also Python 2.7, which is no longer supported and thus at the end of it's life
- One difference that you should keep in mind concerns division rules:
  - Python3 ➔ Division ➔ Float
  - Python2 ➔ Division ➔ Int and Int ➔ Int

    ➔ Float and Float ➔ Float

Getting started [8]

- Programming tutorials always start with the program called “Hello World”
- We will do the same.
- You have to create a file hello.py with atom
- Open a new file and save it as hello.py
- Than add the following line to the sript
  - `print(“Hello World”)`
- Save it again and start it in your Terminal

Getting started [8]

- Add the following commands to the file
  - `print("Jack and Jill went up a hill")`
  - `print("to fetch a pail of water;")`
  - `print("Jack fell down, and broke his crown,")`
  - `print("and Jill came tumbling after.")`
- Save the file and than run it again from the terminal
- Now we look again at our file *

Comments

- You can add lines starting with a “#” to add comments that help you to understand what the program wants to do
- Always comment your code!!!
- Not only does it help you if you come back to work on a program later but it’s also important for sharing programs in a work group if you try to tackle a problem with more than just you.
- To generate comments that span multiple lines use:
  - “””
  - Comment1
  - Comment2
  - “””

Exercise 1 – Simple printing

- Write a program that prints out at least 5 of the following:
  - Your full name
  - Your birthday
  - Your study
  - A simple sentence
  - The shout STOP!!!
  - The name of this course

Introduction to language elements

- An algorithm contains four different language elements:
  - Variables
  - Declarations
  - Branches
  - Loops

Variables

- No algorithm without variables
- Variables are wildcards for actual values
- Consist of three elements
  - Variable type
  - Concrete value
  - Name of the variable

Variables

Variable type

Variable with a value

Declaration

- Prior to using variables they have to be made accessible (“declared“)
- Name and type have to be declared
- So the compiler knows how to deal with the memory content
- “Assignment” can be done later

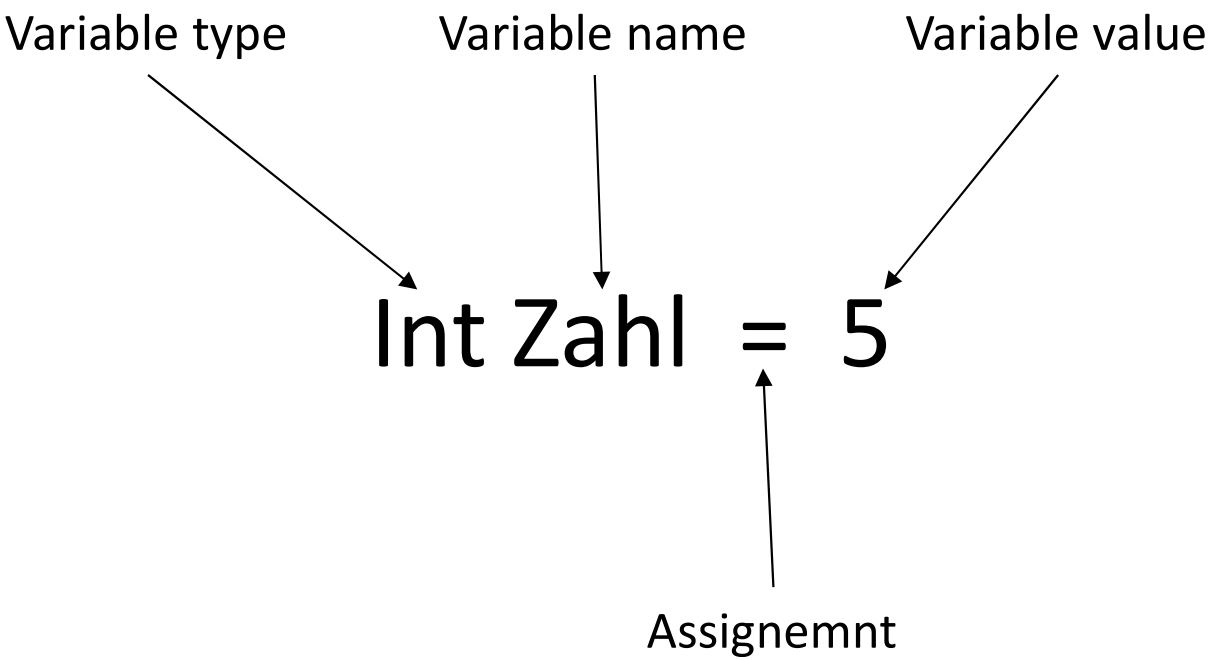
Assignments

- Assigns a specific value to a variable
- Normally done with the “=“ operator
- Not to mixed up with the mathematical equalness
- Variable is always left and the assignment is on the right side
- Assigning a value of a wrong type leads to compiler errors

Executing the program

- If the program is executed a simple text replacement is taking place – every instance of the variable name is replaced with the specified value and the source code is executed with these values

Declaration and Assignment



Variable types (primitives)

Data type	Description	Memory	Values
byte	Whole positive numbers	8bit	0 to 255
sbyte	Whole numbers with signs	8bit	-128 to 127
short	Whole numbers with signs	16bit	-32,768 to 32,767
int	Whole numbers with signs	32bit	-2,147,483,648 to 2,147,483,647
long	Whole numbers with signs	64bit	-92,233,727,203,685,477,508 to 92,233,727,203,685,477,507
float	Floating-point numbers with simple accurace	32bit	-3.402823e38 to 3.402823e38
double	Floating-point numbers with double accurace	64bit	-1.7976931348623e308 to 1.7976931348623e308
Bool	Truth value (Yes, No)	8bit	true / false
Char	One symbol	16bit	One unicode symbol
string	Text		Random text

Assign variables in Python [8]

- Variables are assigned in Python with the help of the equal sign (=)
- Per definition the variable always stands on the left side of the = and the value to be stored on the right side
- The line `first_name = "Bill"` can be interpreted as put the value "Bill" in the Box first_name
  - That's how you store strings
- The Line `b = 432` means put the value 432 into the box b
  - That's how you store numbers

Input

- Now that you know how to print out text it's time for a more complicated program
- This is an opportunity to hand over inputs to the program
- The values handed over to the program have to be stored somehow

  ➔ This is were variables enter the fray

Exercise 2 – Inputting

- Try to write a new Program that takes at least four different inputs
- Two of these inputs should be numbers
- Print out the hand over string variables
- Let the program do at least two different mathematical operations (+ or – or * or /) with the numbers and print out the results of these operations

Mathematical operations [8]

- There are seven different operations that can be used for numbers:
  - Addition: + ➔ 1+2 == 3
  - Subtraction: - ➔ 4-2 == 2
  - Multiplication: * ➔ 3*5 == 15
  - Division: / ➔ 14/3 == 4.6666
  - Integer Division: // ➔ 14//3 == 4
  - Power: ** ➔ 5**2 == 25
  - Remainder: % ➔ 14%3 == 2

Exercise 3 – Math operations

- Write a program that shows the use of all 7 math functions.

Short cuts

Instruction	Variant 1	Variant 3
Increase X by 1	X = X + 1	X+=1
Decrease X by 1	X = X – 1	X-=1
Three times X	X = X * 3	X*=3
X divided by 3	X = X/3	X/=3 (*)
Increase X by 3	X = X + 3	X+=3
Decrease X by 3	X = X – 3	X-=3

(*) /= represents in some progaming languages “not equal“

Branchings

Branchings

Opportunity to execute only specific blocks of code, while excluding other blocks of code

Mostly done by using IF clauses, which can be supplemented with ELSE but don‘t have to

Default Order


```
IF (Predicate)
        THEN
                instructions
                instructions
        ELSE
                instructions
FI
```

Predicate

- A statement that can be either true or false
- The answer is always yes or no
- In all cases of the type bool
- You can link predicates with logical operators like OR and AND

Exercise 4 – Names

- Write a program that asks the user their name, if they enter your name say "That is a nice name", if they enter "John Cleese" or "Michael Palin", tell them how you feel about them ;), otherwise tell them "You have a nice name."

Exercise 5 – Big numbers

- Write a program that asks for two numbers
- If the sum of the numbers is greater than 100, print "That is a big number.“
- If the sum is between ten and 100 print “That is a mediocre number.”
- If the sum is between 5 and ten print “That is a small number”
- In all other cases print “This is a very small number”

Logical Operators

- bool A and bool B – is only true if A as well as B are true
- bool A or bool B – is true if either A or B is true
- Var A == Var B, is only true if A=B

Exercise 6 – Comparing numbers

- Write a program that gets four numbers as input from the user
- The program should print out a positive encouragement if one of these conditions are fullfilled and a negative encouragement if none of them are true
  - Input 1 = Input 2 and Input 3 = Input 4
  - Input 1 = Input 3 and Input 2 = Input 4
  - Input 1 = Input 4 or Input 2 = Input 3

Comparisons [8]

- There are 6 possible mathematical expressions that can be used:
  - < ➔ less than
  - <= ➔ less than or equal
  - > ➔ greater than
  - >= ➔greater than or equal
  - == ➔ equal
  - != ➔ not equal

Exercise 7 – Other comparisons

- Take the program from Exercise 6 and add some more statements for printing out a positive encouragement
  - Input 1 > Input 2 and Input 3 > Input 4
  - Input 1 < Input 3 and Input 2 < Input 4
  - Input 1 = Input 4 or Input 2 >= Input 3
  - Input 1 <= Input 4 and Input2 = Input 3

Summary Branchings

- You learned how to use Branchings in Python with the help of Bool Statements called Predicates
- As of now you should now the following structure:

```
if Predicate:
    Instructions
elif Predicate:
    Instructions
else:
    Instructions
```

Collections of primitives: Arrays

- Numbered collection of variables of one type
- In Python we talk about lists instead
- Only the list has a name
- Elements are addressed with [] and a number (the index)
- Index normaly starts at 0

Creating a list in Python


```
weekdays = [“Monday“,“Tuesday“, … ,“Sunday“]
```

… represents the remaining weekdays

The List weekdays

Index	0	1	2	3	4	5	6
Value	“Monday”	“Tuesday”	“Wednesday”	“Thursday”	“Friday”	“Saturday”	“Sunday”

Length of the list: 7

Exercise 8 – Creating lists

- Write a Program that creates at least four different lists
- Two of them should contain only strings and two of them should contain only numbers
- Try to generate a list that mixes strings and numbers
  - Does this work?

Range function and len()

- With the help of the function range you can create lists containing a range of numbers
- ATTENTION: range is a generator and has to be casted to list with the operator list.
  - `Numbers = list(range(16))`
  - `Numbers2 = list(range(5,26))`
  - `Numbers3 = list(range(0,21,2))`
- You can check for the lengths of lists with the help of the len() function
  - `C = len(Numbers)`

Exercise 9 – Working with range and len

- Write a program that generates four lists with different lengths by using the range() funtion
- The user should be able to input which lengths he needs
- Let the program print out the generated lists as well as the corresponding lengths

Exercise 10 – Skipping parts with range

- Write a Program that generates different lists with the help of the range function
- The program should generate a list that doesn‘t start at 0
- The program should generate two other lists that skip some numbers (three arguments for range)
- Don‘t forget to print out all three lists at the end

Elongating lists

- In Python lists can be elongated with the help of two different commands
  - `A = [1,2,3]`
  - `B = [4,5,6]`
  - `A.append(B)` ➔ `[1,2,3,[4,5,6]]`
  - `A.extend(B)` ➔ `[1,2,3,4,5,6]`

Exercise 11 – Longer lists

- Write a Program that generates six different lists
- Append the third list to first one and the fourth list to the second one
- Extend the first list with the help of the fifth list and the second one with the help of the sixth list

More on lists [8]

- As you know you can access a single element of the list numbers=[0,1,2,3,4,5] as follows numbers[0] = 0 or numbers[3] = 3
- If you want to access the last element you could use the function len by writing numbers[len(numbers)-1] = 5
- But in Python there is an easier way since the index -1 always stands for the last entry (numbers[-1] = 5) the index -2 for the second to last (numbers[-2] = 4) and so on.

Slicing of lists [8]

- Another useful way to get parts of a list is called slicing
- Take the following list:
  - Things = [0, 'Fred', 2, 'S.P.A.M.', 'Stocking', 42, "Jack", "Jill"]
  - Things[0:8] ➔ [0, 'Fred', 2, 'S.P.A.M.', 'Stocking', 42, 'Jack', 'Jill']
  - Things[2:4] ➔ [2, 'S.P.A.M.']
  - Things[4:7] ➔ ['Stocking', 42, 'Jack']
- Slicing is used to return parts of a list
  - The operator is in the form Things[first_index:last_index]
  - It cuts before the first index and before the last index (not included in new list)

Exercise 12 – Slicing of lists

- Write a program that generates two different lists with contents of your choice
- Let the user input four numbers
- If the numbers entered are bigger than the length of the lists ask a second time for smaller numbers
- Slice your lists with the numbers from the user
- If the new numbers are still too big generate slices until the last entry of the lists

Special slices

- Python allows you to use some kind of wildcards for slicing lists:
  - `Numbers[:3]` ➔ Generates a sliced list that contains the first three entries of the original list
  - `Numbers[3:]` ➔ Generates a sliced list that starts at the fourth entry of the list numbers and includes all entries afterwards
  - `Numbers[::3]` ➔ Generates a sliced list that only takes every third entry of the original list
  - `Numbers[::-1]` ➔ Generates a reversed list of the original list

Exercise 13 – Slicing with a :

- Write a program that generates a list with the help of the range function
- The user should input the length of the wanted list and than input four other different numbers that can be used for slicing the list
- The first number should be an endpoint of a new list, the second number a starting point and the other two numbers should represent skips you can do with slicing
- Print out the new lists

Shortening lists

- You can remove entries from list with the remove command
  - Mylist = [„apple“,“banana“,“orange“]
  - Mylist.remove(„banana“)
- Remove only takes one argument so you can remove only one entry at a time

Exercise 14 – Removing entries

- Write a program that generates at least three different lists
- Print these lists out
- Ask the user for one entry from each list he wants to have removed
- Let the program remove the entry
- Print out the new lists

Checking for values in lists

- The next step is to look whether something is in a list or something is not in a list
- This can be done with the „in“ command

```
Mylist = [0,1,2,3,4,5,6]

Check = 4

Check2 = 9

if Check in Mylist:

    print(„Hello“)

If Check2 not in Mylist:

    print(„Hello“)
```

Exercise 61 – Checking for values

- Write a program that generates a list with contents of your choice
- Now let the user input at least three different things
- Check whether these inputs are part of your list or not
- Print out an appropriate answer

Inserting values

- You can insert values at a specific point in lists with the help of the insert command
  - `Mylist = [0,1,2,3,4,5,6]`
  - `Mylist.insert(3,20)`
  - ➔ `[0,1,2,20,3,4,5,6]`

Exercise 62 – Inserting values

- Write a program that generates three lists of different sizes and print them out
- Let the user choose where on these lists he wants to insert which number
- Don’t forget to check whether the chosen index is part of the list or not

Other list functions

- sort(): sort the list in ascending order
- index(): returns the first appearance of a particular value
- max(list): returns the maximum value
- min(list): returns the minimum value
- clear(): removes all entries from the list
- count(): returns the number of elements with the required value
- reverse(): reverses the order of the list
- copy(): creates a duplicate of the list

Multidimensional lists

- Contain two or more running indices
- Are the least standardized

- `[[],[]]` – is a possible listing for two-dimensional lists (depends on the language)
- Especially important for matrices

What is the matrix?

Matrix

- A two-dimensional vector with two indices
- M X N matrix is a matrix with the dimensions N and M
- Operations on the matrix are part of the linear algebra
  - `Matrix1 = [list(range(5))]*5`
  - `Matrix2 = [[0,1,2],[0,1,2],[0,1,2],[0,1,2]]`

Exercise 15 – New matrices

- Write a program that creates at least three different matrices with different sizes
- For this exercise rows stands for the numbers of sublists and columns for the number of entries in the sublists
- One of them should be quadratic
- One should have more columns than rows
- One should have more rows than columns
- Let the user input the dimensions of your matrices

Removing entries

- If you want to remove entries from a multi-dimensional list you have to use remove in a slightly different way
  - `List = [[1,2,3],[1,2,3],[1,2,3]]`
  - `List[1].remove(List[1][1])`
  - ➔ `[[1,2,3],[1,3],[1,2,3]]`

Exercise 60

- Take your lists from exercise 15 and remove at least two different entries from each of these lists with the help of indices
- Let the user choose which entries he wants to have removed after he declared the dimensions of the lists
- Run at least on security check (if statement) to see if the given indices are part of the lists (len operator)

Dictionaries

- Dictionaries have keys and values
- The keys are used to find the corresponding values
- The line `numbers = {}` tells Python that numbers is a dictionary
- The command `numbers.keys()` returns an unsorted list containing all keys that can be used by a for loop
- Similar to lists with the help of `numbers[x]` you can access a specific member of the dictionary, but in this case x is a string

Dictionaries

- The line `numbers[name] = phone` adds a name and the number to the dictionary
  - If the name had already been in the dictionary phone would replace the old entry
- The command `if name in numbers` checks if the name was already in numbers and `del numbers[name]` removes the entry if it is

Exercise 16 – Simple Dictionary

- Write a program that generates a simple dictionary consisting of at least three pre defined entries
- Ask the user for a new entry and add this new one to the dictionary afterwards
- Then ask the user for an entry that should be removed and remove the entry from the dictionary
- Print out the remaining dictionary

Exercise 17 – Searching in dictionaries

- Create a dictionary with at least ten entries
- Let the user search the dictionary for a specific entry
- If the entry is already present ask the user for an update of the contents for this entry, if the entry doesn‘t exist add it to the dictionary
- Run this option at least three times during one call of the program
- Print out the original dictionary and the modified dictionary at the end

Complex dictionaries

- Recap:
  - Dictionaries have keys and values
  - Keys can be strings or numbers
  - Keys point to values
  - Values can be any type of variable, including lists or dictionaries, that can also contain lists or dictionaries and so on
  - Sounds Scary?

Exercise 18 – Address Book

- Write a program that creates a complex dictionary in form of an address book
- Each entry of the dictionary „address_book“ should contain another dictionary in which the address, phone number, birth day and e-mail address are collected
- The final dictionary should contain information of at least four persons
- Print out the final dictionary at the end

Exercise 19 – Adjusting the address book

- Take your dictionary from exercise 27 and write a program that allows the user to make some changes
  - Ask the user if he wants to add an entry, delete an entry or change an entry
  - If the user wants to delete an entry ask for the entry which should be removed and remove it
  - If the user wants to change an entry ask which part should be changed and what the new content should be
  - If the user wants to add an entry ask for the information necessary to create a new entry
- Print the dictionary at the start and end of the program

Tuples

- Tuples are a third form of datastructures that can be generated in Python
- They are ordered, unchangeable and indexed
- They are generated with the help of round brackets
  - `Mytuple = („apple“,“banana“,“orange“)`
  - `Mytuple = („apple“,)`
- Ordered means that you can‘t change the order of the values
- Unchangable means that you can‘t add or remove values from the tuple

Exercise 20 – Working with tuples

- Write a program that creates four tuple of different lengths
- The lengths should be dictated by the user
- Print out the tuples
- Afterwards let the user chose one entry from each tuple to be displayed
- Try to generate a tuple with duplicate values.

Sets

- Sets are the last type of data structure that is included in the standard Python package
- Sets are unordered, unchangeable and can‘t be indexed
- They are also created with curly brackets but you can‘t create an empty set
  - `Myset = {„apple“,“banana“,“orange“}`
  - `Myset = {„apple“,}`
- Unordered means that the order of the values can change on different calls
- Unchangeable means that you can‘t change the items in the set but you can add or remove items

Exercise 21 – Working with Sets

- Write a program that creates four sets of different lengths
- The lengths should be dictated by the user
- Print out the sets

Elements of sets

- You can remove elements of sets to be able to work with them with the help of the pop command
- You can only remove one element at a time
  - `Elem = myset.pop()`
- You can add elements to sets by using the add command
  - `Myset.add(elem)`

Exercise 22 – Adding and removing

- Take your sets from exercise 21
- Remove some elements from them
- Try to remove two elements directly one after the other
- Check which elements were removed with the print command
- Add the elements again to the set
- Print out the new sets
- What changes can you see?

Summary Datastructures

- You should now know the basics of the four basic datastructures included in the standard Python Package
  - List
  - Dictionary
  - Tuple
  - Set
- The characteristics of each datastructure should be known to you by now

Loops

Loops

- Repeating source code
- … in most cases with different assignments of the control variable
- Mostly two types:
  - For – Loop
  - While – Loop
- Like branches are operated via predicates

For-loop

- Default parts:
  - Running index (mostly int i)
  - Starting assignment for i, update for i in each loop

```
FOR (int i=0; i < iterations; i++)
BEGIN
END
```

Uses of the for loop

- If you would normally say “from/to” or “for all” and you need a control variable
- If you know the number of runs through the loop
- E.g. loop over an array

Iteration over all elements of a list

- for i in range(16):

  instructions

  instructions

- i runs from 0 up to 15
- i increases with every iteration by 1

Exercise 23 – Your first loops

- All following loops should run at least over a range from 0 to 21
- Write one loop that sums up all numbers in the setted range
- Write another loop that sums up only the even numbers in the setted range
- Write a third loop that generates a list that contains the values of the range in reverse order
- Print out all three results

„for each“ loop

- We saw how to use a for loop with the help of a range
- Python also allows you to loop through each entry of a list:

```
for elem in list:
    Instructions
    Instructions
```

Exercise 63 – More for loops

- Create a list that contains at least 20 different numbers that aren't sorted
  - Don't use range for the generation of the list
- Write a loop that runs over all elements of the list and adds them all up
- Write a second loop that tells the user for each element whether it is bigger or lower than a number specified by him
  - This number should be generated by using input before the start of the loop

Summary for loops

- Each of the two versions of for loops have their own uses
- If you want to do something for a specific number of times, without working with a list, you would use the normal for loop with the range() operator
- If you want to work directly with the contents of a list (or other data structure) you would use the „for each“ loop by iterating over the elements of the corresponding list

While loop

- If you don‘t know the number of loops
- Operated by a predicate
- Is repeated until predicate is no longer true
- You have to put in the control variable manually
- `rand = random.randint(0,100)`
- `list = list(range(rand))`
- `random.shuffle(list)`

While loop construction


```
while (predicate):
        instructions
        add up run_index
```

Exercise 24 – 3 while loops

- Write a program that uses while loops to finish the tasks below:
  - Searching for a specific number (e.g. 5) in an integer list of unknown length
  - Multiplying all elements with each other of an integer list of unknown length
  - Printing out the contents of a string list of unknown length elementwise
  - `rand = random.randint(0,100)`
  - `list1 = list(range(rand))`
  - `random.shuffle(list1)`

Nested loops


```
FOR (int i=0; i<3; i++)
BEGIN
        FOR (int j=0; j<3; j++)
        BEGIN
                print i j // prints the values of i and j on the
                          // console
        END
END
```

Nested loops


```
FOR (int i=0; i<3; i++)
BEGIN
        FOR (int j=0; j<3; j++)
        BEGIN
                print matrix[i][j]
        END
END
```

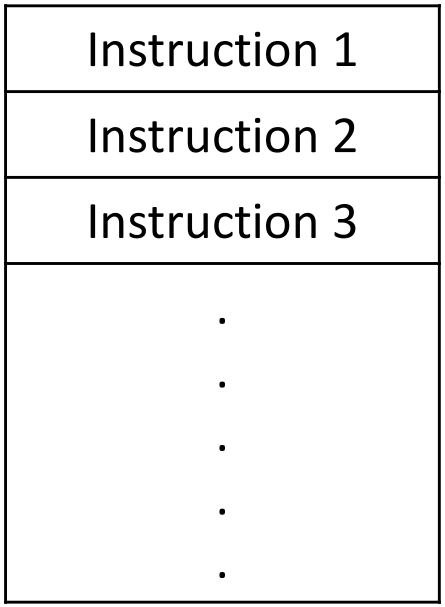
Exercise 25 – Loop through a matrix

- Write a Program that uses nested loops to tackle the tasks below:
  - Search for the coordinates of maximal value of the matrix
  - Sum up all entries of the matrix
  - Describe the multiplication of the components of two matrices in a third resulting matrix

Structograms

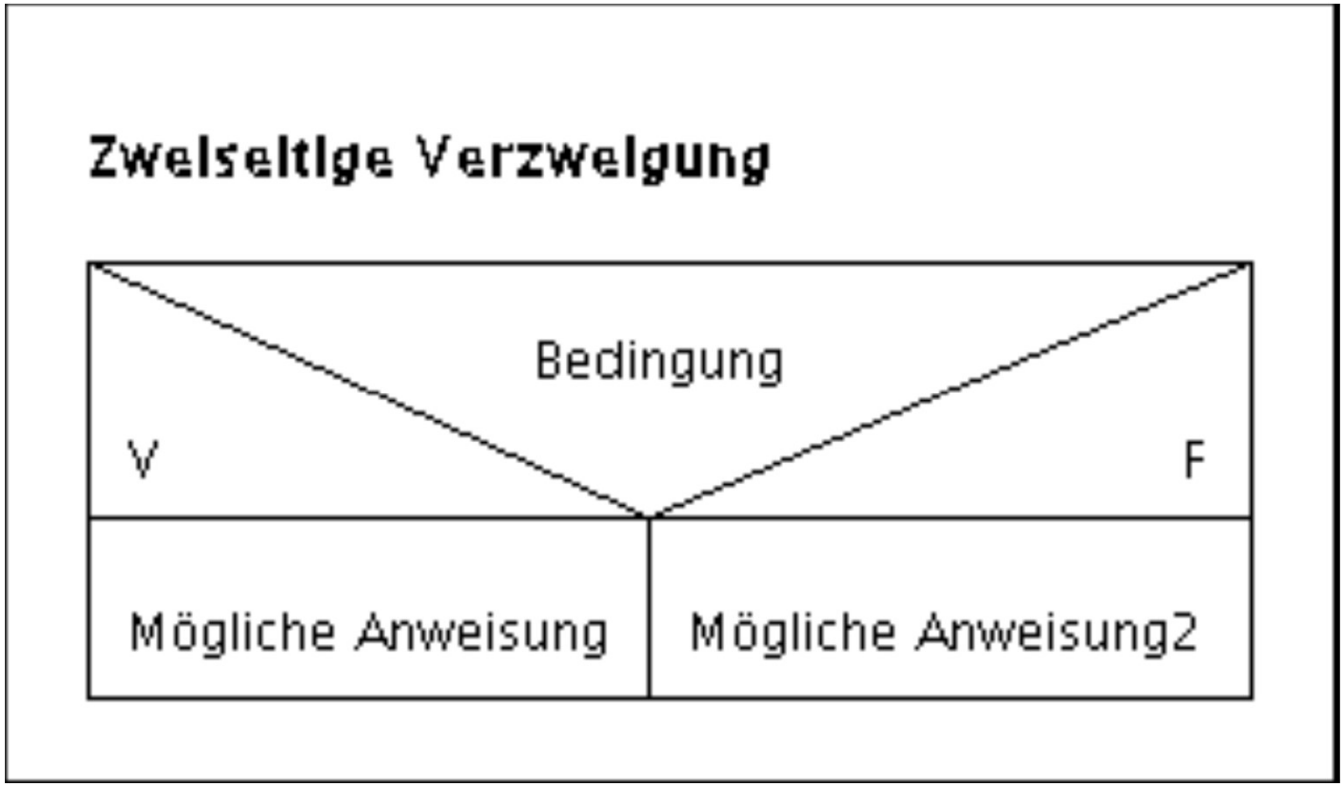
- Are another option für visualizing Pseudocode
- Elements of an algorithm are represented by specific graphics
- Can contain instructions, but is no necessity
- Are normally done by hand on paper

Instructions

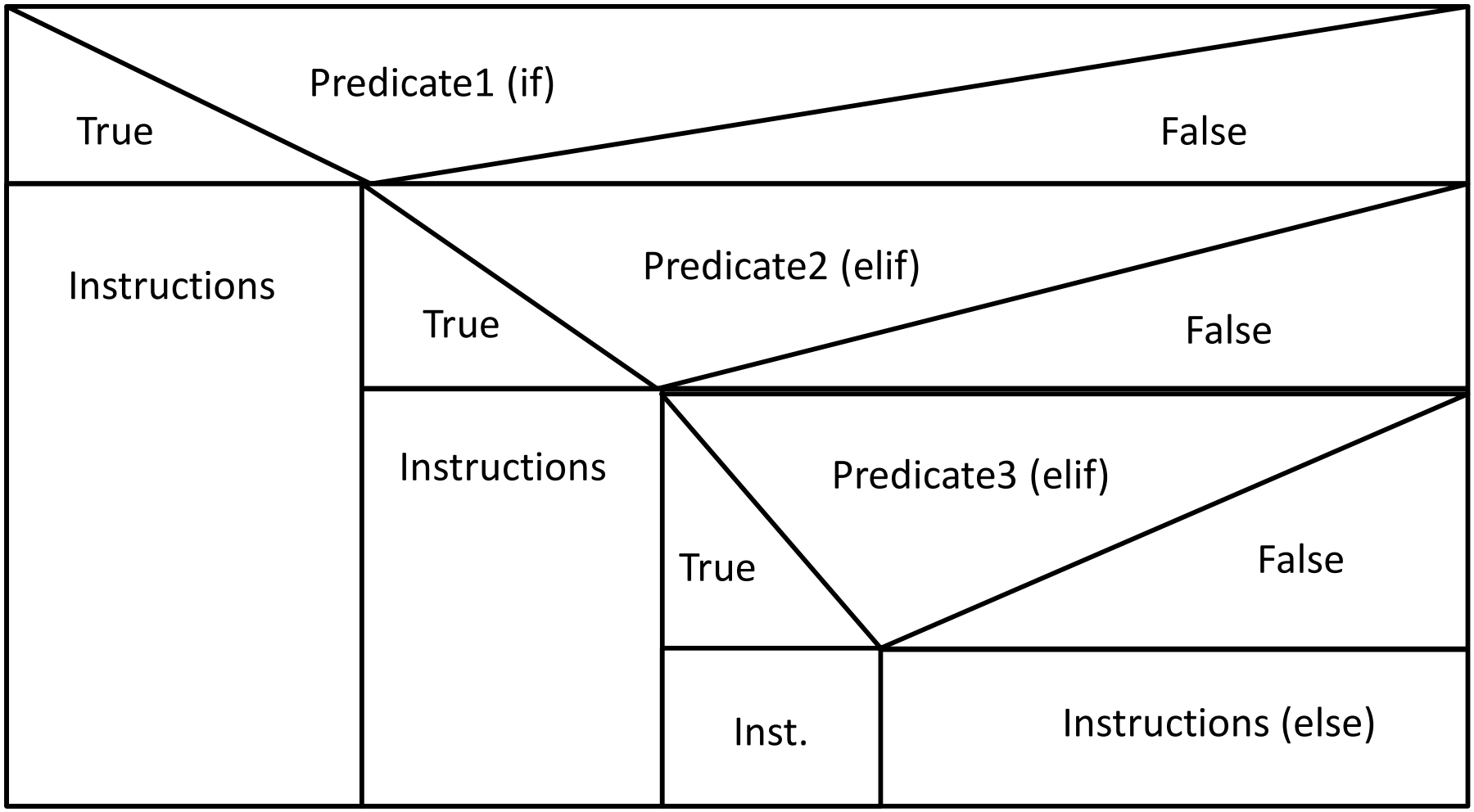


The call of functions are simply instructions

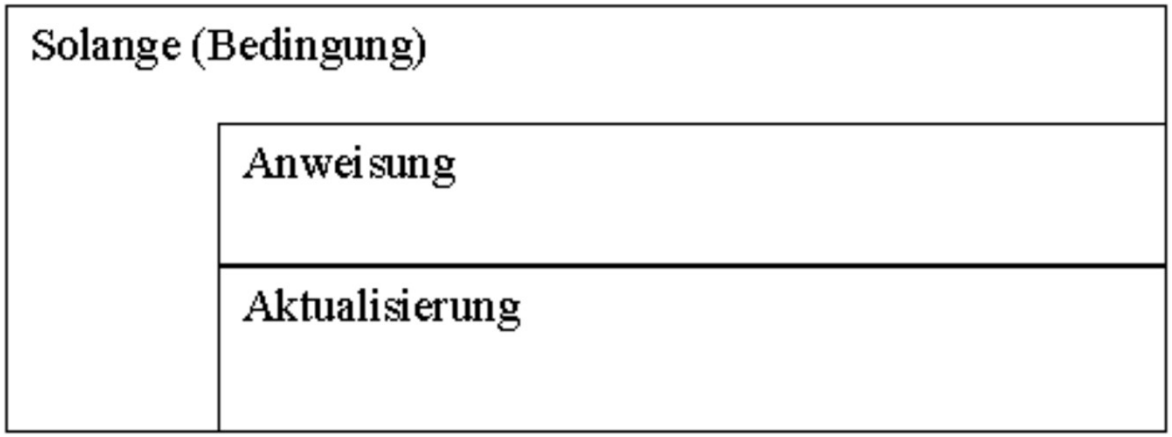
IF clauses


Zweiseitige Verzweigung
Bedingung
V
F
Mögliche Anweisung
Mögliche Anweisung2

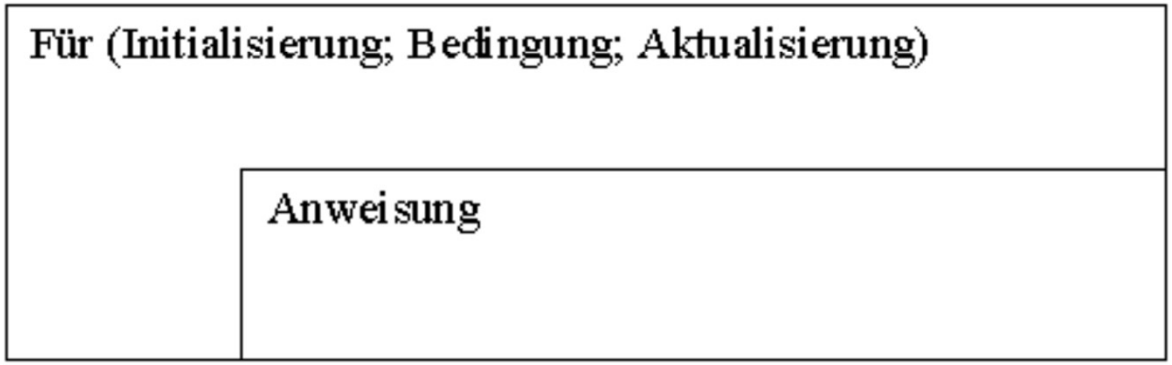
If clauses multiple ifs



Loops

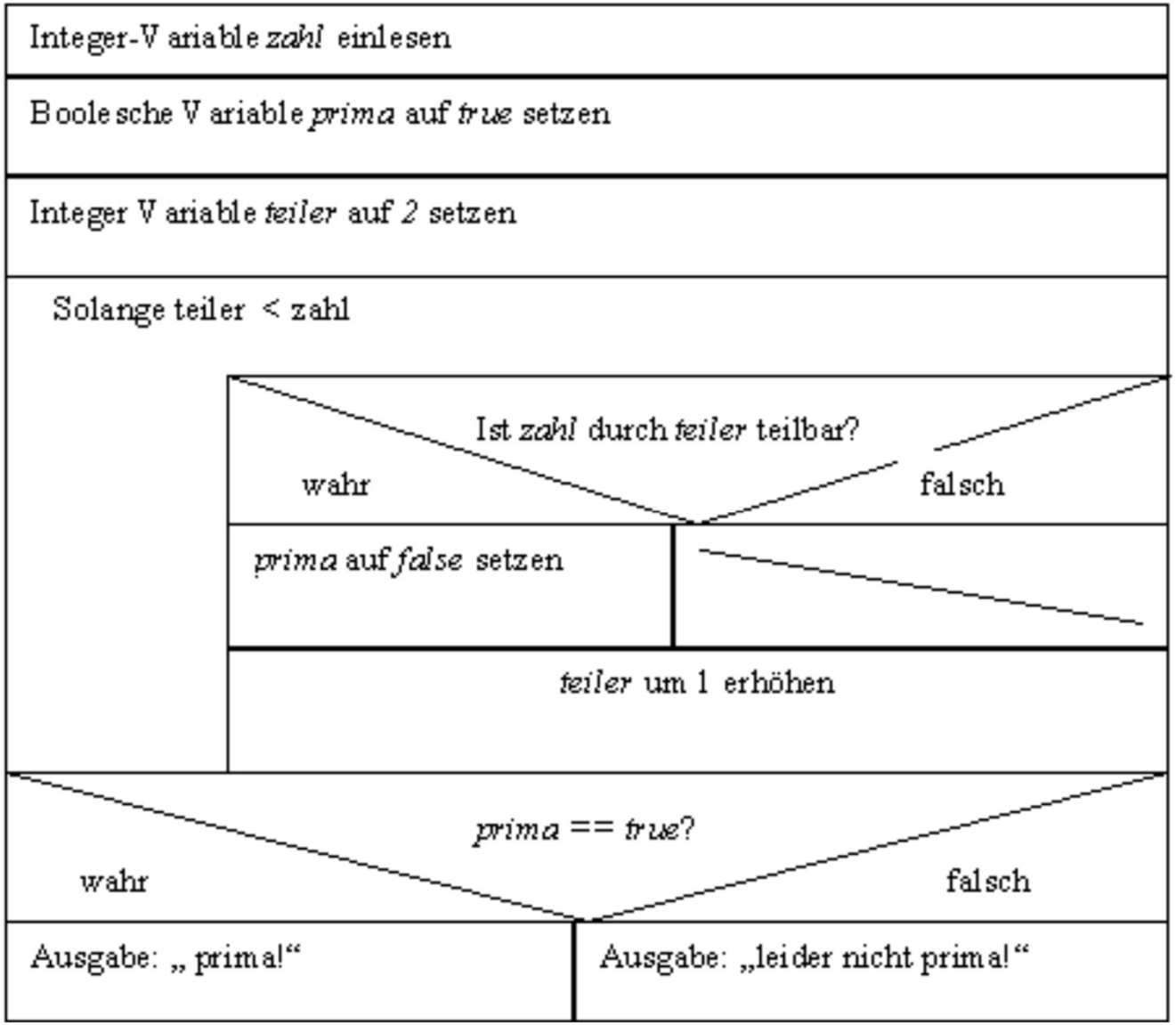


While



For

Example


Integer-Variable zahl einlesen
Boolesche Variable prima auf true setzen
Integer Variable teiler auf 2 setzen
Solange teiler < zahl
Ist zahl durch teiler teilbar?
wahr
falsch
prima auf false setzen
teiler um 1 erhöhen
prima == true?
wahr
falsch
Ausgabe: „prima!“
Ausgabe: „leider nicht prima!“

Exercise 26 – Structograms

- Create some structograms on paper for the programs you created for the exercises 25, 24, 23 and 6
- Exercise 6 – The if statements for four inputs:
  - Input 1 = Input 2 and Input 3 = Input 4
  - Input 1 = Input 3 and Input 2 = Input 4
  - Input 1 = Input 4 or Input 2 = Input 3
- Exercise 23 – for your for loop that calculates the sum
- Exercise 24 – for your while loop that searches for a specific number
- Exercise 25 – for your loops that find the maximal value

Pseudocode

- Displays exemplarly certain parts of an algorithm or concepts
- Contains only parts of the programing language that are essential for this concept
- Simplified programing language that can‘t be translated into machine code
- There is no standard for Pseudocode

Pseudocode example


```
General Search Algorithm:

input:    a properly formulated search problem
          a function "insert" to place new nodes into a queue

1 fringe := make-queue(initial-node)
2 loop
3    if empty?(fringe) then return FAIL
4    else
5      X := remove-front(fringe)
6      if satisfies-goal(state(X)) then return X
7      else
8         fringe := insert(fringe,expand(X))
9  end loop
```

Figure 3: Pseudo-code for General Search Algorithm

Pseudocode Python if

- For if statements you can use simplified version

```
get 4 inputs in1, in2, in3 & in4

if in1=in2 & in3=in4: print1

elif in1=in3 & in2=in4: print2

elif in1=in4 or in2=in3: print3

else: print4


if empty: …

if type(str): …
```

Pseudocode Python loops

- For for loops you could use it like this:

```
sum=0
for range(..):
        add i to sum
```

- For while loop you could use it like this:

```
product=0
while in list:
        multiply product with current entry of list
```

Exercise 27 – Pseudocode

- Write a Pseudocode for the programs you created for the exercises 25, 24 and 23
- Exercise 23 – for your for loop that calculates the sum
- Exercise 24 – for your while loop that searches for a specific number
- Exercise 25 – for your loops that find the maximal value of a matrix

Functions [8]

- Programmers hate to repeat things
- Therefore each repeating process should be written as a function
- Saves multiples in the source code for repeating processes during the program
- Results in a better structure and outline of the code
- Describes the tools of programing
- Purifies the memory management

Signature

The signature of a function describes only, which name the function has, which parameters it uses and which data type the return value of the function and thus the function itself has. The programing of a function isn‘t displayed in a signature.

Parameter

- Are given to the function on call back
- Determine how to use the function
- Types and order of the parameters are dictated explicitly by the signature
- Types are fixed, but variable in the declaration
- Call back of a function with parameters of the wrong type leads to compiler errors

Program execution

- Like a normal instruction block with all possibilities for branchings and all loops etc.
- It is possible to call another function within a function
- IMPORTANT: Except for the return value, all variables and structures, that are used within a function, exist only during the call of the function and are lost afterwards
  - Workspace before, during and after the call of a function

The feature def

- In Python function are defined with the feature def:
  - The statement def is followed by the name of the function with the parameters passed to function in the brackets
  - The statements after a “:” in the first line are executed when the function is used
  - This continues until either the idented statements end or a return is encountered

```
def sumup(a,b):
    sum = a+b
    return sum
```

Variables in functions

- When using functions you often have variables in this repeated code parts
- Python deals with them in a special way
- So far we encountered global variables
- Functions use a special type called local variables
  - These only exist when the function runs
  - When a local variable has the same name as another variable (e.g. a global variable), the local one hides the other

Rules for functions

- Avoid duplicate source code
- A function should only depend on her parameters
- One function is exactly one job
- The source code shouldn‘t contain too much lines

Exercise 28 – Your first functions

- Create a program that defines functions for the four mathematical basic operations (+, -, * and /)
- Call each of the functions at least three times with different parameters

Exercise 29 – Functions with simple loops

- Write a program that contains two different functions
- The first function should take a list as an argument and return the sum of all elements in the list, calculated by a loop
- The second function should also take a list and return a list in which the contents of the list that was used as parameter are reversed

Exercise 30 – Functions with nested loops

- In Exercise 25 we worked with nested loops to find the maximum of a matrix, calculate the sum of all entries of a matrix and create a new matrix that is the result of element-wise multiplication of two other matrices
- Write a program that contains three different functions, one for each of the three tasks above

Multiple returns

- Python allows you to return multiple values to the program

```
def sum_sub(a,b):

    sum = a + b

    sub = a - b

    return sum,sub
```

Exercise 31 – Multiple returns

- In exercise 28 you created four different functions, now combine them into a new function that returns all four results at once
- Call this function at least three times with different parameters and print out the results

Exercise 32 – Multiple returns from lists

- Take your functions from exercise 29 and create a new one that returns both results at once
- The function should include only one loop

Keywords

- In Python there exist 3 keywords you should know:
  - break
  - continue
  - pass
- Each of them has different uses which we will discuss on the following slides
- Try to use them only if you really need them

Break

- This command completely stops a loop
- You can use it if you are searching for something and don‘t need to run over the rest of the loop
- This is mainly used to reduce the running time of programs if you are only interested in certain parts of the calculation

Continue

- Continue can be used to skip parts of a loop
- All commands that follow a continue in a loop aren‘t executed
- You can use this if you don‘t want your program to go over multiple if statements or other loops nested inside your loop
- Use it only if you can be sure that you don‘t need the remaining calculations in the current iteration

Pass

- Pass can be used as a placeholder for future code
- In python empty code is not allowed in function definitions, loops, if statements or class definitions
- You should use this only in combination with comments that tell what you want to do in this place later on
- I use it often when I‘m constructing my programs
  - That means I create the loops, if statements and functions as empty parts and add the necessary code later on
  - Like this you can test out your code step by step, while having the overall structure ready

Exercise 64 – Keywords in Python

- Create a list with at least 15 random entries
- Iterate over it with a for loop and add up the elements
- Create one if statement in your loop that can’t be true and add a pass
- Include a second if statement in your loop that breaks the loop if your sum gets bigger than a threshold (you chose)
- Include a third if statement in your loop that uses continue if the current entry of your list is in a specific range, eg. 50 to 60, and prints out something in all other cases (use else)
- Print out the sum and the number of

Using modules [8]

- You can use more functions by using additional modules in Python
- With the help of the import operator you can read in external functions
- Note that programs should not have the same name as modules, e.g. calendar, since the operator import would look for a file called calendar.py for example
  - The program would try to read in itself, which works poorly at best

Finding commands for standard modules

- To see which command are available for standard modules either look in the library reference (if you downloaded it) or go to https://docs.python.org/3/library/
- For the module calender you can find the function prcal that prints a calendar for a year
  - To use this function you need the call `calendar.prcal(year)`
  - If you don’t want to use all functions you can use:
    - `From calendar import prcal`
    - `From time import time, ctime`

Exercise 33 – New modules

- Write a program that asks the user to guess a specific number
- Generate a random number between 0 and 99, search the library to find a module and a function for this task
- Generate a while loop that asks for a guess as long as the user didn’t guess right
- Print out hints if the user is lower or higher than the random number
- Count the number of guesses the user needed and print them out after the user guessed correctly

os [9]

- Provides a portable way of using operating system dependent functionality
- Simple put you can use Linux/Bash commands in your python code
- Some functions have names analogous to bash commands you already know
- Simple import with the call:
  - `import os`

Examples os [9]

- `os.chdir(path)`
  - change directory, analogous to cd
- `os.getcwd()`
  - get a string representing the current directory
- `os.mkdir(path)`
  - create a new directory
- `os.rename(src, dst)`
  - rename file or directory src to dst
- `os.system("command")`
  - Run the given command in a subshell, you can use Linux commands as Strings to run from your program

Exercise 34 – Working with os

- Create a directory structure by using os
- Create one parent directory with at least five child directories
- Create a file in each directory simply by using os and by changing directories beforehand

math [10]

- Provides access to mathematical functions and constants
- Cannot be used with complex numbers
  - Use cmath instead
- The distinction between functions that can use complex numbers and those which don’t is made since most user don’t want to learn quite as much mathematics
  - Furthermore receiving an exception instead of a complex number can make the program more resilent
- All returned values are floats

Examples math [10]

- `math.isnan(x)`
  - Returns true if x is not a number
- `math.exp(x)`
  - Returns e raised to the power x
- `math.log(x[,base])`
  - With one argument returns the natural logarithm (to base e)
  - With two arguments returns the logarithm of to the given base
- `math.log2(x)`
  - Returns the logarithm of x to the base of 2
- `math.log10(x)`
  - Returns the logarithm of x to the base of 10

Examples math [10]

- `math.pow(x,y)`
  - Returns x raised to the power y
- `math.sqrt(x)`
  - Returns the square root of x
- `math.cos(x), math.sin(x), math.tan(x),`
  `math.acos(x), math.asin(x), math.atan(x)`
  - Trigonometric functions
- `math.pi, math.e, math.tau`
  - π = 3.141592…
  - *e* = 2.718281…
  - τ = 6.283185….

Exercise 35 – Working with math

- Write a program that takes two numbers (x and y) from the user and calculates new values with the help of the math functions on the previous slides (exp, log, log2, log10, pow, sqrt, cos, sin, tan, acos, asin, atan)
- Print out the results

Exercise 36 – Working with constants

- Write a second program that takes again two numbers from the user and calculates new values with the help of the same functions as in exercise 35
- This time the input variables have to multiplied by one of three shown constants (Pi, Euler number or Tau)
- Let the user choose which constant should be used
- Print out the results

random [11]

- As you should already know this module implements Pseudo-random number generators
- You already saw
  - `random.randint(a,b)`
  - Returns a random integer N such that a <= N < b
- Other methods are:
  - `random.random()` ➔ random number between 0 and 1
  - `random.gauss(mu,sigma)` ➔ normal distribution
  - `random.shuffle(x)` ➔ mixes list randomly

Exercise 37 – Random generator

- Write a program that generates several random numbers
- Let the user input the ranges for three random integers
- Let the user choose how many random floats between 0 and 1 should be created
- Let the user put in at least two mu and two sigma to create normal distribution values
- What happens if you switch these two values?
- Last but not least put all generated values in a list, print the list, shuffle the list and print it again

NumPy [12]

- The fundamental package for scientific computing in Python
- At the core of the NumPy package, is the ndarray object
- NumPy arrays have a fixed size at creation
- The elements in a NumPy array are all required to be of the same data type
- NumPy arrays facilitate advanced mathematical and other types of operations on large numbers of data
- import numpy as np ➔ convention for the import of the numpy module

Examples NumPy [12]

- Arrays in numpy can be created in various ways
  - np.array([4,3,2,1]) ➔ [4,3,2,1]
  - np.arange(4) ➔ [0,1,2,3]
  - np.zeros(4) ➔ [0,0,0,0]
  - np.ones(4) ➔ [1,1,1,1]
  - np.zeros((3,4)) ➔ [[0,0,0,0], [0,0,0,0], [0,0,0,0]]
  - np.arange(15).reshape(3, 5) ➔

    [[0,1,2,3,4],[5,6,7,8,9],[10,11,12,13,14]]
  - np.random.randn(6,4) ➔ ?

Basics NumPy [12]

- All operations on arrays apply elementwise
  - [1,2,3] * 3 ➔ [3,6,9]
  - [10,20,30,40] – [0,1,2,3] ➔ [10,19,28,37]
- There are basic functions like:
  - array.sum()
  - array.min()
  - array.max()
  - array.transpose()
  - array.shape

Exercise 38 – NumPy arrays

- Write a simple python program that generates the following arrays:
  - An array consisting of 25 zeros
  - An array consisting of 8 ones
  - An array consisting of numbers from 0 to 24 in order
  - An array consisting of 9 entries of your choice
  - An two-dimensional array TD consisting of 6 rows of arrays with 5 entries
  - A transposed version of the two-dimensional array TD
- Write furthermore an algorithm to print out the sum, maximum value, minimum value and shape of all created arrays

Scipy [13]

- SciPy is a collection of mathematical algorithms and convenience functions built on the NumPy extension of Python
- With SciPy, an interactive Python session becomes a data-processing and system-prototyping environment rivaling systems, such as MATLAB, IDL, Octave, R-Lab, and SciLab
- SciPy sub-packages need to be imported separately
  - from scipy import linalg, optimize

SciPy subpackages [13]

Subpackage	Description
Cluster	Clustering algorithms
Constants	Physical and mathematical constants
Fftpack	Fast Fourier Transform routines
Integrate	Integration and ordinary differential equation solvers
Interpolate	Interpolation and smoothing splines
Io	Input and Output
Linalg	Linear algebra
Ndimage	N-dimensional image processing
Odr	Orthogonal distance regression
Optimize	Optimization and root-finding routines
Signal	Signal processing
Sparse	Sparse matrices and associated routines
Spatial	Spatial data structures and algorithms
Special	Special functions
Stats	Statistical distributions and functions

Pandas [14]

- When working with tabular data, such as data stored in spreadsheets or databases, pandas is the right tool for you
- In pandas, a data table is called a DataFrame
- pandas supports the integration with many file formats or data sources out of the box
  - csv, excel, sql, json, parquet
- There is no need to loop over all rows of your data table to do calculations
  - Data manipulations on a column work elementwise
  - Adding a column to a DataFrame based on existing data in other columns is straightforward

Syntax Pandas [14]

- Standard import: import pandas as pd
- Two data types:
  - s = pd.Series([1, 3, 5, np.nan, 6, 8])
  - df = pd.DataFrame(np.zeros(2, 4), index=[“Test1”,”Test2”]
    columns=list("ABCD"))
- Some essential functions are:
  - df.sum()
  - df.describe()
  - df.sort_index(axis=1,ascending=False)
  - df.sort_values(by=“B”)
  - df.index or df.columns

Exercise 39 – Pandas DataFrames

Write a program that creates a DataFrame consisting of at least 4 rows and 6 columns. The DataFrame should contain random numbers generated via NumPy. Let the program create 6 new DataFrames by sorting the original DataFrame alongside the columns (df.sort_values()). Let the program print a description of the original DataFrame. What do these numbers mean?

BioPandas [15]

- it is a bit cumbersome to work with PDB files in "modern" programming languages
- Takes pandas to the structural biology world
- Working with molecular structures of biological macromolecules (from PDB and MOL2 files) in pandas DataFrames is what BioPandas is all about
- Please install biopandas with $ pip install biopandas via your terminal before trying to run the script below
- *BioPDB.py* *

Matplotlib [16]

- Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python
  - Create publication quality plots.
  - Make interactive figures that can zoom, pan, update.
  - Customize visual style and layout.
  - Export to many file formats .
  - Embed in JupyterLab and Graphical User Interfaces.
  - Use a rich array of third-party packages built on Matplotlib
- import matplotlib.pyplot as plt
  - International convention for importing matplotlib

argparse [17]

- Makes it easy to write user-friendly command-line interfaces
- The program defines what arguments it requires
- The argparse module also automatically generates:
  - Help messages
  - usage messages
  - issues errors when users give the program invalid arguments

arguments.py

- You have an program called *arguments.py*
  - Try run it ➔ What happens?
  - Now try to run it again but type in at least two numbers behind the start commando (python arguments.py 5 6)
  - Try different combinations of numbers ➔ What do you think the program does?
  - Now try the command python arguments.py –h

Arguments of argparse [17]

- There are two types of arguments that argparse can handle
  - Optional arguments which are created with parser.add_argument('-f', '--foo’)
    - Allow the user to let the program make something different like calculating the sum instead of giving out the max
  - Positional arguments which are created with parser.add_argument('bar’)
    - Have to be given to the program on the start
    - As you saw with *arguments.py* if you don’t give the argument the program won’t be able to start

Exercise 40 – argparse

- Write a program using the module argparse that takes some strings as input and concatenates these strings in the order of input to one new string.
- Add at least three other arguments besides the strings that modify the program in different ways.
  - The program could add spaces or could reverse the order of the input strings for example.

Writing Files

- You can work with files in Python
- To write something into a new file you need the open command in combination with a filename and and operator
  - `datei = open("neue_datei.txt","w")`
  - „w“ stands for writing
  - „a“ stands for appending
  - The first one would overwrite contents included in the file and the second one would add the contents at the end of the file
- After you wrote everything you have to close the file
  - `datei.close()`

Writing Files

- If you want to open a file that doesn‘t exist it is created
- You can also work on Files with the help of the „with“ statement
- By using this statement you don‘t have to close the file

```
with open("bla.txt","w") as with_datei:
    with_datei.write("Dies ist ein Test\n")
```

Exercise 43 – Writing Files

- Write a program that creates two different files.
- The first file should be opened normally and two lines should be written in it.
- Afterwards close the file, open it again and add another three lines of text to the file
- The second file should be generated by using „with“ and a loop to write a countdown from 10 to 0
- Each number should be in a separate line

Reading Files

- There are three variants to read in files
- The first one reads in the whole file at once:

```
with open("eins_zu_zehn.txt","r") as lese_datei:
    inhalt = lese_datei.read()
print(repr(inhalt))
```

- The repr() command can show a representation of the contents of the file
- The version above should be used only with small files because the whole contents are put into the memory and you can‘t work with them

Reading Files

- The other two options read in files line by line

```
with open("eins_zu_zehn.txt","r") as lese_datei2:
    zeilen = lese_datei2.readlines()
print(zeilen)
```

- In the case above the whole file is read in at once, but each line is a separate entry in the list `zeilen`
- This should also only be used with small files because you read in the whole file into the memory

Reading files

- The last version works with a generator and has the advantage that only one line at a time is read into the memory

```
with open("eins_zu_zehn.txt","r") as lese_datei3:
    for zeile in lese_datei3:
        print(repr(zeile))
```

Exercise 44 – Reading a File

- Read in your second file from exercise 43, the one with the countdown, by using the generator method
- Calculate the sum of the numbers and print out the result

Exercise 45 – Copying files

Write a program that takes the name of a file from the user and generates a copy of this file. The Copy should be generated by writing the contents of the original file into a new file line by line. Use generators to optimize your memory usage for this exercise.

Raising Exceptions

- Python allows you to raise exceptions, so that errors occurring during the run of the program don’t stop it
- Consists of four operators
  - try: the program tries to run the commands afterwards
  - except: raises exceptions in combination with specific errors like RunTimeError, ValueError or ZeroDivisionError
  - else: the instructions after this are only run if the try didn’t raise an error
  - finally: instructions after this run at the end, whether errors occurred or not

Exercise 41 – Exceptions on inputs

- Write a program that runs a simple division
- Generate a random integer and divide it through an integer inputted by the user
- Create an exception for wrong inputs (a string for example)
- Create a second exception to handle division by zero
- Both exceptions should belong to the same try
- Print out the result of the division at the end and try several different inputs