Prof. Dr. Gemma Roig
M.Sc. Alperen Kantarcı
M.Sc. Gamze Akyol

Programmieren für Studierende der Naturwissenschaften

Lecture 1 - Introduction

Organization

- Lecture
  - Alperen Kantarcı and Gamze Akyol
  - Available until 6 p.m. through Moodle and e-mail
  - Emails: kantarci@em.uni-frankfurt.de and akyol@em.uni-frankfurt.de

Objective

- Basics of programming using Python and similar languages
- Common (beginner) concepts and structures of programming.
- Python syntax
- Basic handling of data (records)
- Ability to work with documentation
- Ability to work independently also with other programming languages
- Companion effects should not be the only motivation: Certificates and CPs

Objective

- What cannot be covered?
- Other programming languages (e.g. C++,FORTRAN, R etc.)
- Subject-specific knowledge (e.g. DNA sequence analyses, etc.)
- More advanced topics (efficiency/parallelism/network)
- Object-oriented programming

Note: How do you actually learn how to program?

- Through practical application and research! Similar to foreign languages.
- Active and implicit knowledge acquisition through practical applications

Reasons for Participation

3 CP (90hours) in the format VL (2 hours) + P (3 hours)

- In order to pass 80% of the mandatory tasks should be presented
- There are 8 exercise sheets with a total of 23 compulsory tasks ⇒You need 19 tasks presented.
- Ask for credit from the examination office!
- If credit for CP is not possible : certificate (under the same conditions)

Contents

- L1: Basics of programming
  P1: Exercise 1 and help to installments.

- L2: Elementary data types and control structures
  P2: Exercises

- L3: Aggregated data types
  P3: Exercises

- L4: Aggregated data types and functions
  P4: Exercises

- L5:Testing,error messages and self-help
  P5: Exercises

Contents

Timing

- Daily:
  - Lecture-up to 2 hours (live)
  - Exercise: 3 hours in small groups → Very much your own work

- Lecture time
  - 10-12 am
- Lunch break?
- Closing at 4.00 - 4.30 pm at the latest
- Leave early (before 4:00pm) ?
- • (Mandatory) tasks completed
  • Peer teaching: Help at least one other group member with one of the tasks

Cooperation in the practical phase

- Groups of two to three
- Each group member should present one solution **per exercise sheet.**
- Group composition
  - If possible one experienced person in a group
  - No groups with only experienced participants (peer learning)
- Plagiarism – the exercise will be considered as failed

Things to consider during the practical part

- Take short breaks, let your eyes relax
- Swap roles in the group (peer coding)
- Drink enough water to stay hydrated

- Inside the classroom
  - You can listen/watch music, audio books, videos
  - Use headphones
  - Do not distract or disturb other participants with loud talk
  - Do not bring smelly food or drink
  - You can go outside the room and come back whenever you want

Questions about the organization?

Higher Level Programming Languages

Developer implements the program in a (platform independent?) programming language that is human readable.

- Alphabet: The underlying character set
- Syntax: Permissible character combinations (keywords and commands) and their grammar
- Semantics: Meaning of formally accepted Syntax

- A computer translates the program into machine language for the processor

- Attention: Information = Syntax + Semantics + Pragmatics
  - A program cannot capture the pragmatic meaning of a piece of information
  - ...where by the information available to us as humans is transmitted in completely
  - →Maximum precision is required in programming. A computer executes exactly what is required and needs regarding the corresponding instructions.

Python

- Good readability
- Great beginner language
- Detailed documentation and widely available tutorials
- Many special modules and libraries for natural sciences ,e.g.
  - NumPy
  - SciPy
  - BioPython
  - etc.

How do we get an executable Python program?

How to program?

- Describe and analyze the problem
- Reading and structuring the meaning
- Selection, if necessary development and description of the required algorithms
- Transfer/conversion into a programming language
- Testing

Algorithm

• **[Theoretical Computer Science]** A common formal definition: a computational rule for solving a problem is called an algorithm if and only if there exists a Turing machine equivalent to that computational rule that stops for every input that has a solution.

- Simplified definition: An algorithm describes an executable finite sequence of steps to correctly solve a given task using a finite memory volume.
- We will not delve further into variations and classes of algorithms.

Structure of an algorithm

Description: does not matter (implementation independent)

- Flowchart
- `Pseudocode (if Joke=true : evaluate(Stud, grade points+1))`
- Mathematical(E=mc^2)

Building blocks:

- Data input
- Instructions
- Data output

Program:

- concrete form of the algorithm
- adapted to the necessities and possibilities of the real machine

Algorithm for making coffee

Proposal 1
- Boil water
- Put coffee powder in the cup
- Fill water into the cup

Proposal 2
- Put coffee powder in the cup
- Boil water
- Fill water into the cup

More realistic example

Write a program that calculates the area of a circle. The radius of the circle is to be entered by the user.

- The most important step: analysis and description
- Ask the user for the size of the radius.
- (Optional, do I want to implement this?) Prevent incorrect entries

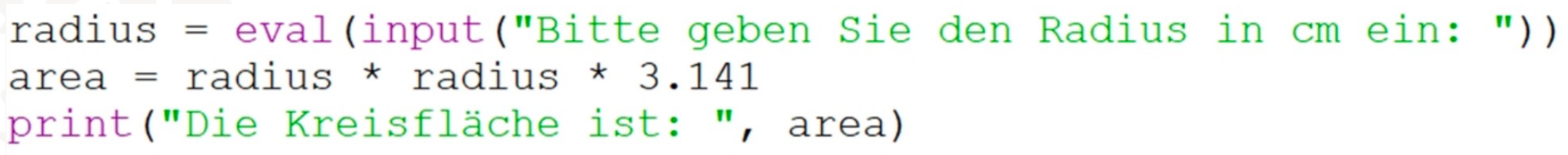
- Calculate the area of the circle using the following formula: Circle area=Radius*Radius*π
- Own research belongs naturally again and again to it
- (Not required in the task) Output the circle area in the console
- Let external clients enrich requirements with user stories

Let‘s Code

Save the code

Best in syntax-driven editor/development environment ,e.g.

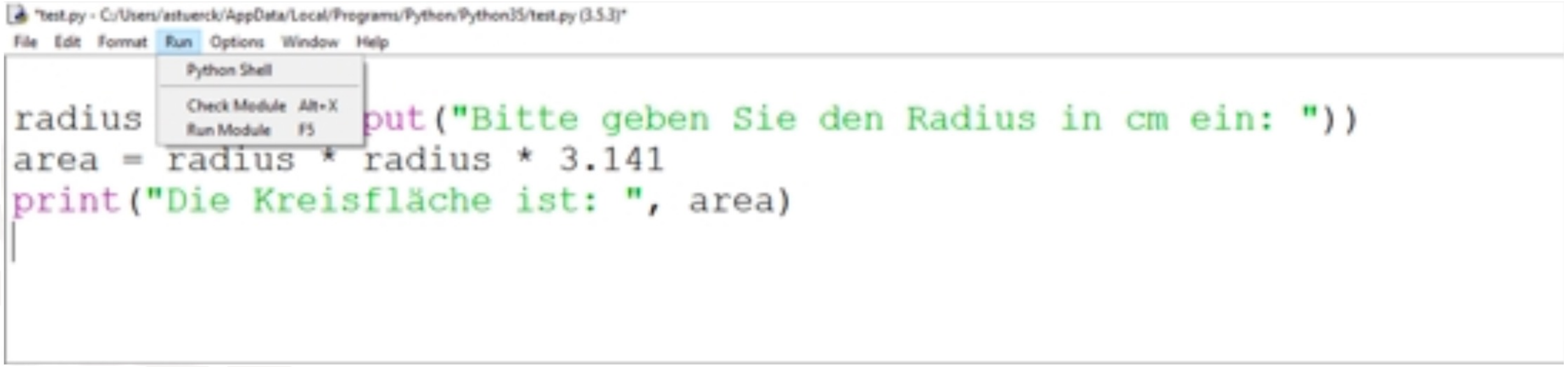
- **IDLE** (Python integrated environment)
- Spyder
- **Visual Studio Code**, Eclipse etc.
- If necessary, any kind of text editor (Notepad, nano, vim, etc.)



```
radius = eval(input("Bitte geben Sie den Radius in cm ein: "))
area = radius * radius * 3.141
print("Die Kreisfläche ist: ", area)
```

Execute

In the development environment


test.py - C:/Users/astuerck/AppData/Local/Programs/Python/Python35/test.py (3.5.3)
File Edit Format Run Options Window Help
Python Shell
Check Module Alt+X
Run Module F5

```
radius        put("Bitte geben Sie den Radius in cm ein: "))
area = radius * radius * 3.141
print("Die Kreisfläche ist: ", area)
```

From the terminal (for Linux/MacOS)
In the directory where the program is located with the command

```
python3 <name>.py
```

Basic language elements

- The instructions are separated from each other by a paragraph (ENTER)

- Instructions are executed in sequence during program execution, from the first to the last instruction

- After that the interpreter stops – the program is finished

Note: If you are programming in a compiled language (C, C++, Java etc.) compilation and execution are different operations. Interpreted languages like Python uses runtime interpretation.

Variables and assignment

A variable can conceptually also be considered as a container in memory

- Three attributes: Name, Type, Value

1. Name: a unique word in a namespace (here initially the entire program) under which the variable can be addressed in the program text
2. Type: Describes the type of the variable to indicate the possible set of operations
3. Value: Content of the variable (depends on the type)

   Example: DIX (surname of Otto DIX or Roman numeral 509?)

Data types

• **Integer:**	Integers	E.g.	13	0	1
• **Float:**	"Commas"	E.g.	-5.38	0.0	1.0
• **String:**	String (=Text)	E.g.	"Hello World"	"Hello."	"123"

- Note: Even though we say "comma numbers", in most cases we write them with a period in a source code

Dynamic Typing in Python

Dynamic typing in Python

- Type assignment of variables at **runtime** of a program
- Assignment creates a variable and assigns a value to it. No declaration necessary.
- The type() function returns the type.

- Let’s try

The assignment operator

- = is used in Python to change the type and value of a variable, e.g.

```
radius = 5
```

- In Python you don’t have to do anything else: Any memory management (create,free,...) of memory areas is done for you by the Python interpreter.
- In other programming languages, additional steps may be required.

What can be to the right of the assignment operator?

Expressions are terms (in mathematics)
- e.g.: 3+5, 7*3, B*3, 7*3-B*3...

- Valid operators:
    - e.g.: +, -, **
    - But also functions
        - e.g.: input(), print(), eval()

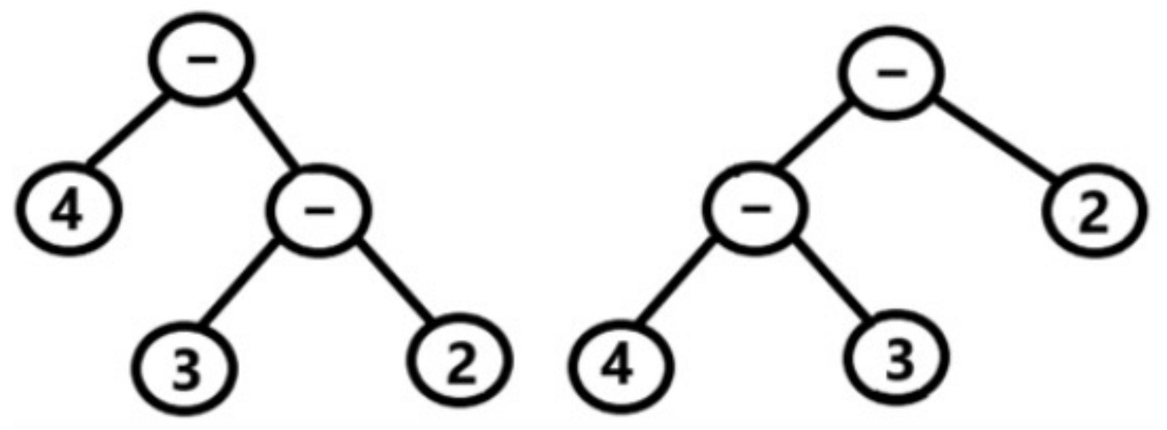
- Or mathematical functions like
    - e.g.: math.sin(0.5) (available only after importing the math module)

Learn more about modules in the coming days

Evaluation Sequence

- Mathematics: "Point before dash” calculation
- Computer Science (General):
  - If there are several operators in an expression, it must be regulated which one is evaluated first (see handout)
  - Round brackets control the evaluation order
  - For operators with the same binding strength, generally from left to right (left-associative).
  - Example subtraction:
    - 4 - 3 - 2 would not always be the same:
    - (4-3)-2=-1
    - 4-(3-2)=3
- Exception:**(power) as in mathematics right-associative.
  - 4**3**2 = 4**(3**2) = 262144

4 - 3 - 2=?



Comments

Source code without comments –introduced with #- quickly becomes a spaghetti

- Support after long breaks
- Assistance for collaborative work
- NO significant delays in code execution in Python
- No impact on compiles due to internal mechanisms of the compiler (compiler)

- Quality criteria:
  - Short,but more meaningful than the code itself
  - Simple English sentences
- Placement:
  - In a separate line: `#this is a comment`
  - In the code line as inline comment
  - As block comments,so-called docstrings,respecting the indentation:

```
"""this is a multi line comment to describe something""
```

Variable name - Conventions

Naming is not always trivial:
- Team internal style guides
- E.g. programming manual EPI(FB12,Prof.Krömker) for Python3.x

PEP8-Style Guide for Python Code
- https://www.python.org/dev/peps/pep-0008/#id17

- Own memorized notation

Variable name - Conventions

Independently defined names start with a letter (a,...,z, A,...Z) or an underscore"_".

- Names can be of any length and from the 2nd character also containdigits(0,...,9).
- Upper and lowercase is always relevant! myvariable1 != myVariable1
- Umlauts like   ö,ü or Ö,Ü or !,§,$,...are not allowed as characters in names.
- Certain keywords are forbidden as variable names(e.g.and, or, except, import, if, in...)

Variable name - Conventions

- **Variables:** `noun_with_underscore`
  - Example: `current_index, radius, ...`
- **Constants:** `NOUN_IN_CAPITAL_LETTERS`
  - Example: `MAX_LENGTH`
  - Note: In Python there is no way to leave constants definitely unchanged
- **Tips:**
  - Get into the habit of choosing English names right away
  - Choose names that speak as much as possible: e.g. `current_line` instead of `cl`
  - singular for single objects -- plural for collections (`student_name` vs. `student_names`)
  - NEVER "l" (small L) or I (large i) or O (large o)! Often hardly distinguishable from the digits "1" (one) or "0" (zero)

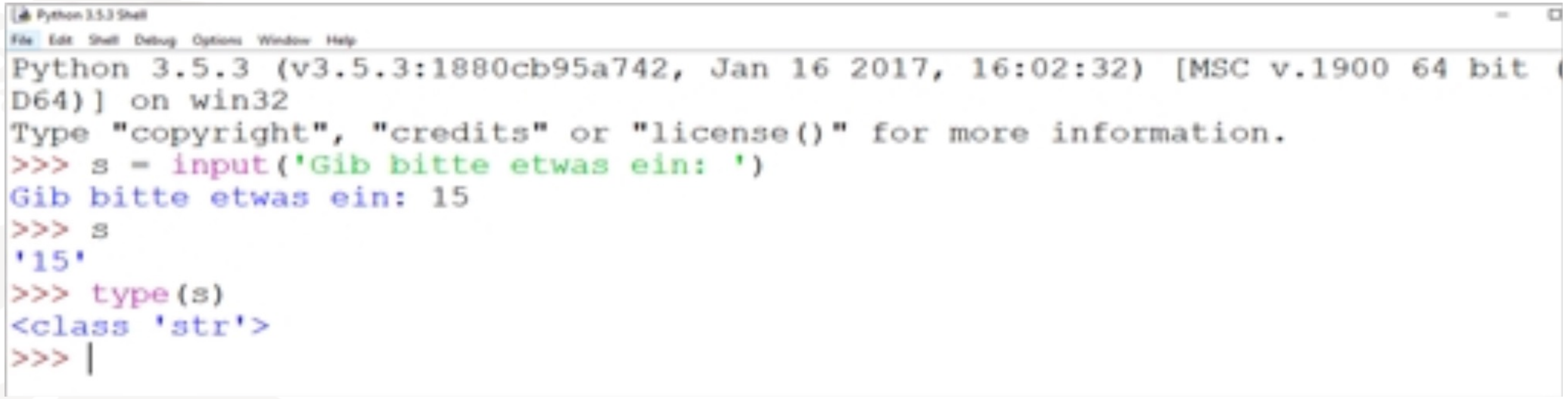
First programs: Input and output functions

`input()`

When the `input()` function is called, the program sequence stops until the user makes an input from the keyboard and completes it with the return key

- `input()` always returns a value of the type string (character string)

If you want to make a number out of the string, you have to convert it explicitly

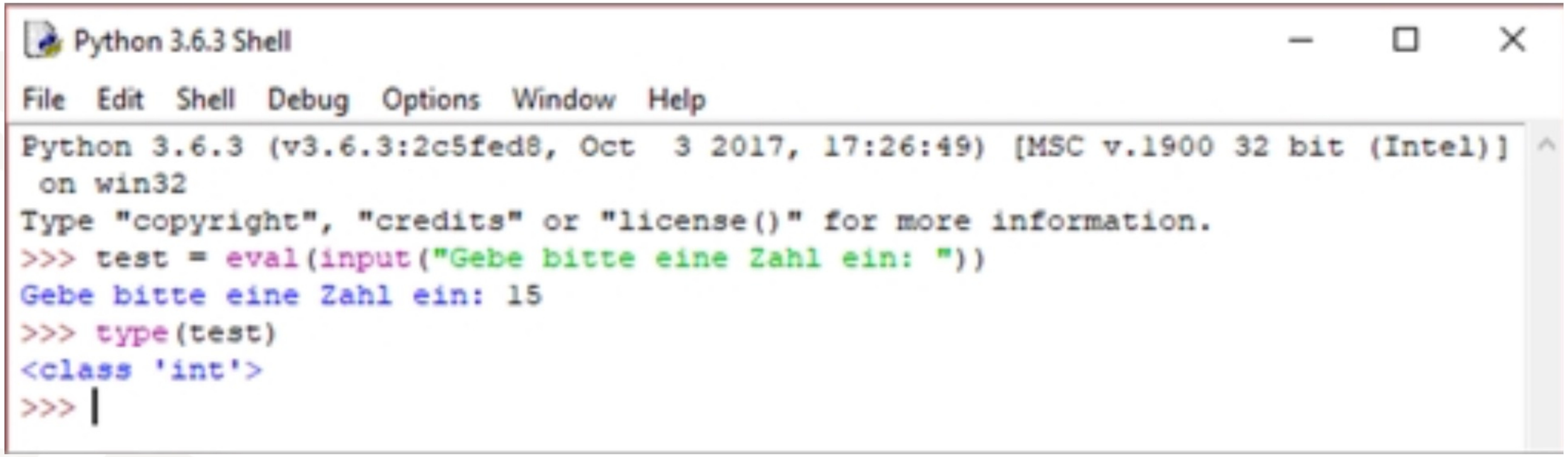


```
Python 3.5.3 (v3.5.3:1880cb95a742, Jan 16 2017, 16:02:32) [MSC v.1900 64 bit (
D64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> s = input('Gib bitte etwas ein: ')
Gib bitte etwas ein: 15
>>> s
'15'
>>> type(s)
<class 'str'>
>>> |
```

First programs: Input and output functions

eval()

This function does not have the goal to make an integer number out of a string!

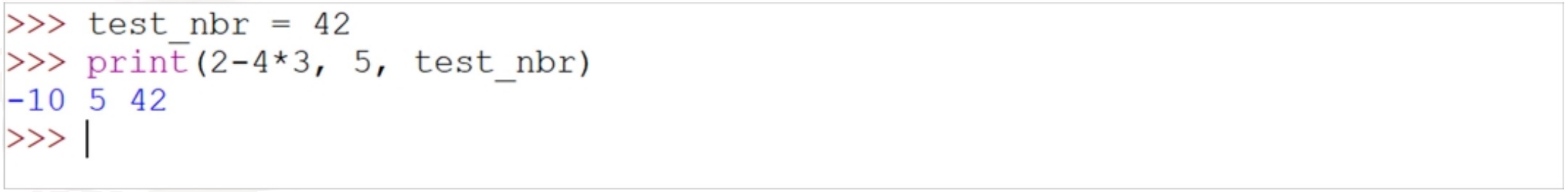


```
Python 3.6.3 (v3.6.3:2c5fed8, Oct  3 2017, 17:26:49) [MSC v.1900 32 bit (Intel)]
 on win32
Type "copyright", "credits" or "license()" for more information.
>>> test = eval(input("Gebe bitte eine Zahl ein: "))
Gebe bitte eine Zahl ein: 15
>>> type(test)
<class 'int'>
>>> 
```

Research what it does and why it is considered particularly error-prone!

Function `print()`

- can be used to print something on the console
- Today (in exercise) you should have a look at the documentation for this function and experiment with the inputs

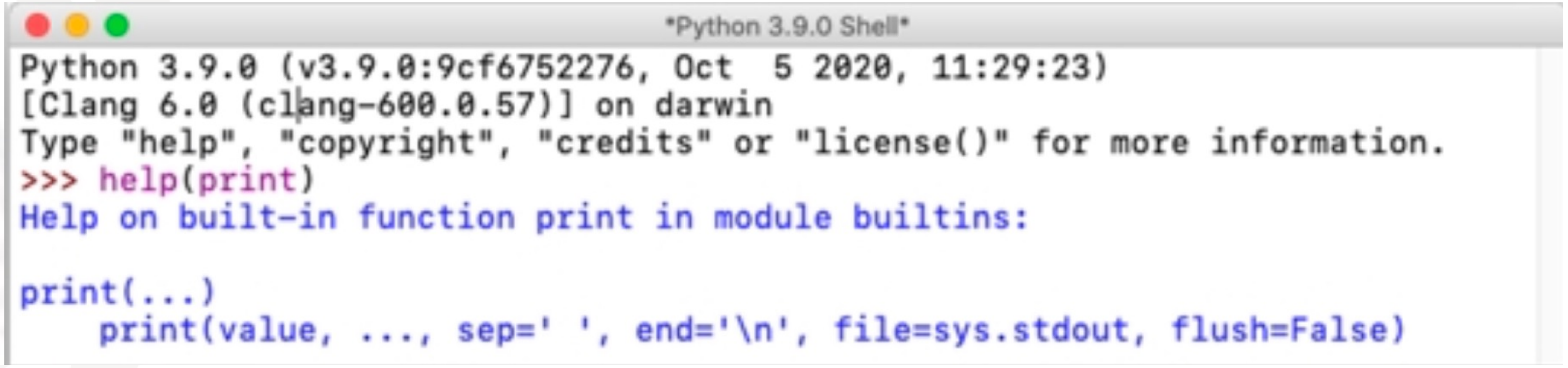


```
>>> test_nbr = 42
>>> print(2-4*3, 5, test_nbr)
-10 5 42
>>> |
```

Function `help()`

- can be used to call help texts for the modules, functions and data types

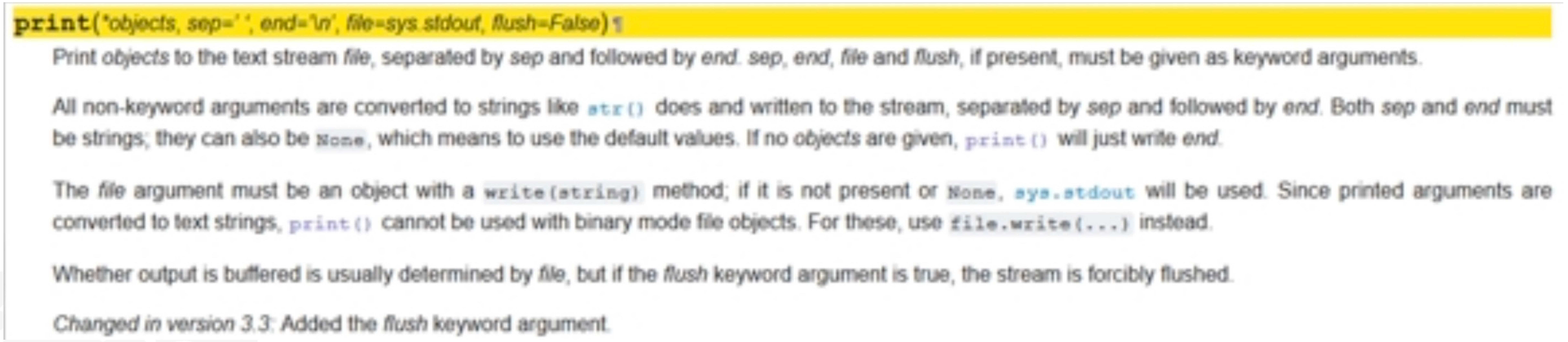
- Syntax: `help([object])`, e.g.:
- `help(print)`
- `help(int)`



```
Python 3.9.0 (v3.9.0:9cf6752276, Oct  5 2020, 11:29:23)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license()" for more information.
>>> help(print)
Help on built-in function print in module builtins:

print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)
```

Documentation



print(**objects, sep=' ', end='\n', file=sys.stdout, flush=False*)¶

Print *objects* to the text stream *file*, separated by *sep* and followed by *end*. *sep*, *end*, *file* and *flush*, if present, must be given as keyword arguments.

All non-keyword arguments are converted to strings like `str()` does and written to the stream, separated by *sep* and followed by *end*. Both *sep* and *end* must be strings; they can also be `None`, which means to use the default values. If no *objects* are given, `print()` will just write *end*.

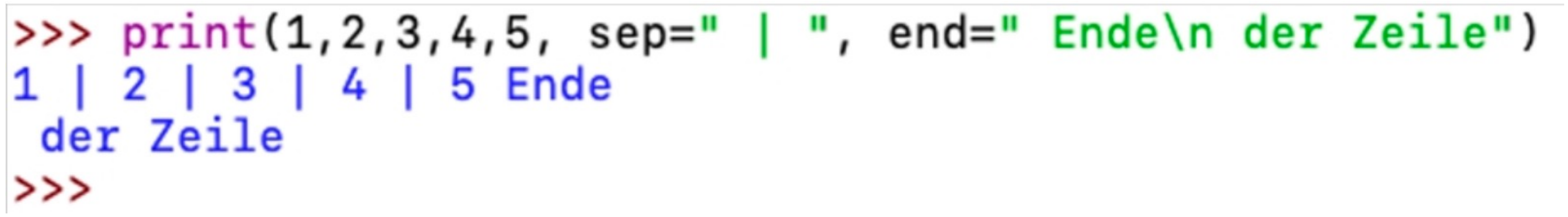
The *file* argument must be an object with a `write(string)` method; if it is not present or `None`, `sys.stdout` will be used. Since printed arguments are converted to text strings, `print()` cannot be used with binary mode file objects. For these, use `file.write(...)` instead.

Whether output is buffered is usually determined by *file*, but if the *flush* keyword argument is true, the stream is forcibly flushed.

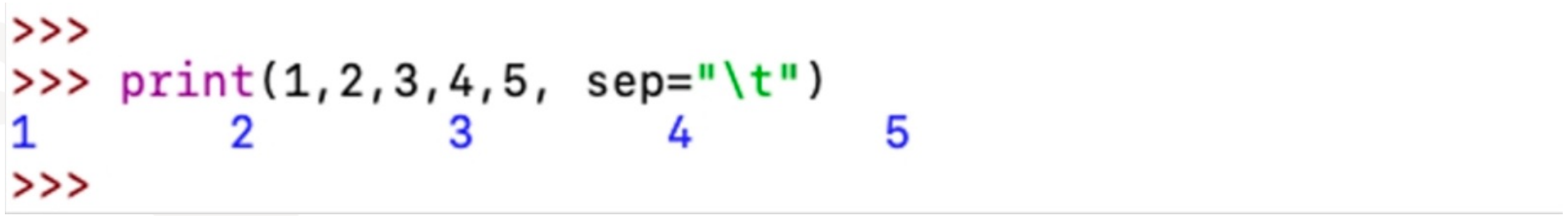
Changed in version 3.3: Added the *flush* keyword argument.

“translation":
- Pass any number of expressions `(*objects) to` the function
- The separator (separator, `sep`) between the values is a `‘ ’`(blank) by default, but can be set by `sep='anything'` changeable for example `sep=‘,'`
- The termination control character (end) is `\n` (i.e. `NewLine`) as default and can also be changed
- The output stream is set as default `file = sys.stdout` but can also be changed (e.g. for writing to files)
- `flush` (a bit more complicated and will not be discussed here)

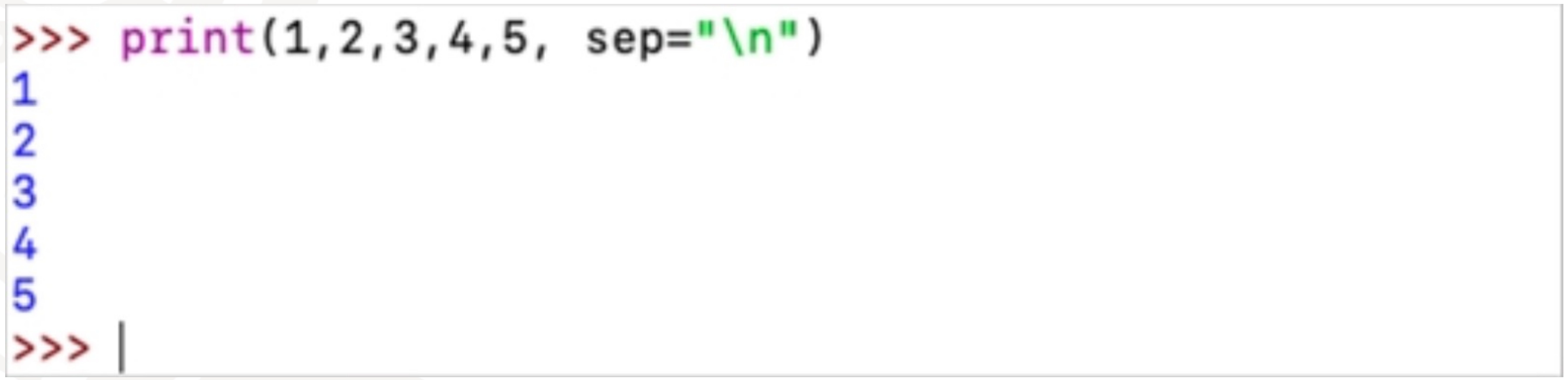
Example



```
>>> print(1,2,3,4,5, sep=" | ", end=" Ende\n der Zeile")
1 | 2 | 3 | 4 | 5 Ende
 der Zeile
>>>
```



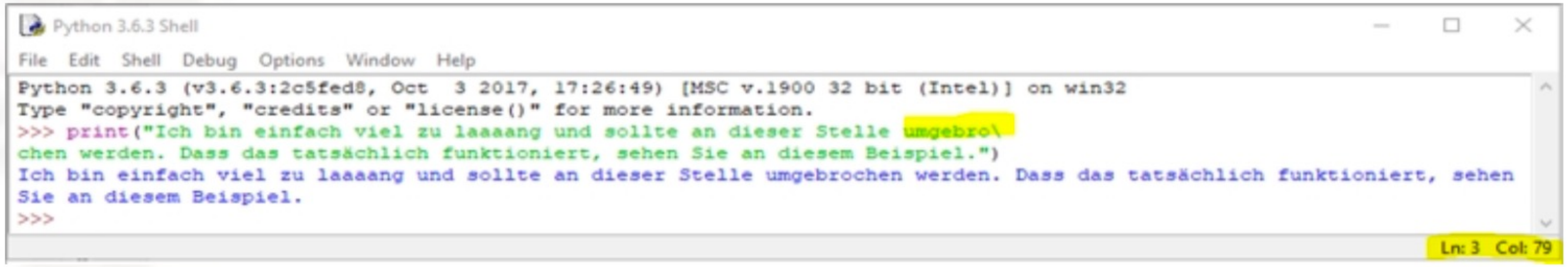
```
>>>
>>> print(1,2,3,4,5, sep="\t")
1	2	3	4	5
>>>
```



```
>>> print(1,2,3,4,5, sep="\n")
1
2
3
4
5
>>> |
```

Programming Conventions

- are agreements/guidelines that programmers make with each other
  - Readability
  - Unity
  - Overview
- Limit a statement line to a maximum of 79 characters.
  - If necessary: \ (backslash) at the end of a line extends this line "logically" with the help of the next line:



```
Python 3.6.3 (v3.6.3:2c5fed8, Oct  3 2017, 17:26:49) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> print("Ich bin einfach viel zu laaaang und sollte an dieser Stelle umgebro\
chen werden. Dass das tatsächlich funktioniert, sehen Sie an diesem Beispiel.")
Ich bin einfach viel zu laaaang und sollte an dieser Stelle umgebrochen werden. Dass das tatsächlich funktioniert, sehen
Sie an diesem Beispiel.
>>>
```

  - For the Python interpreter empty lines have no meaning.
    - Structure your program text into "logical blocks", "paragraphs" as it were, by using "blank lines"

Programming Conventions

Before and after an operator (=,+,-,...) there is a blank (space character)

After a comma there is a space

There is no space before and after a bracket()

If operators of different binding strength (priority) are used in an expression, one should visualize the higher binding strength by leaving out the blank, e.g.:

```
x = x*2 - 1          or          c = (a+b) * (a-b)
```

Learn Programming

- **PRACTICE** (Practice, practice, practice...)

- Today: Exercise sheet 1(will be released soon)

- Forum for group search is unlocked in Moodle course (or find a group just now)

- I am available until 4 pm at any time for questions of anykind, as well as if you want to present your solutions