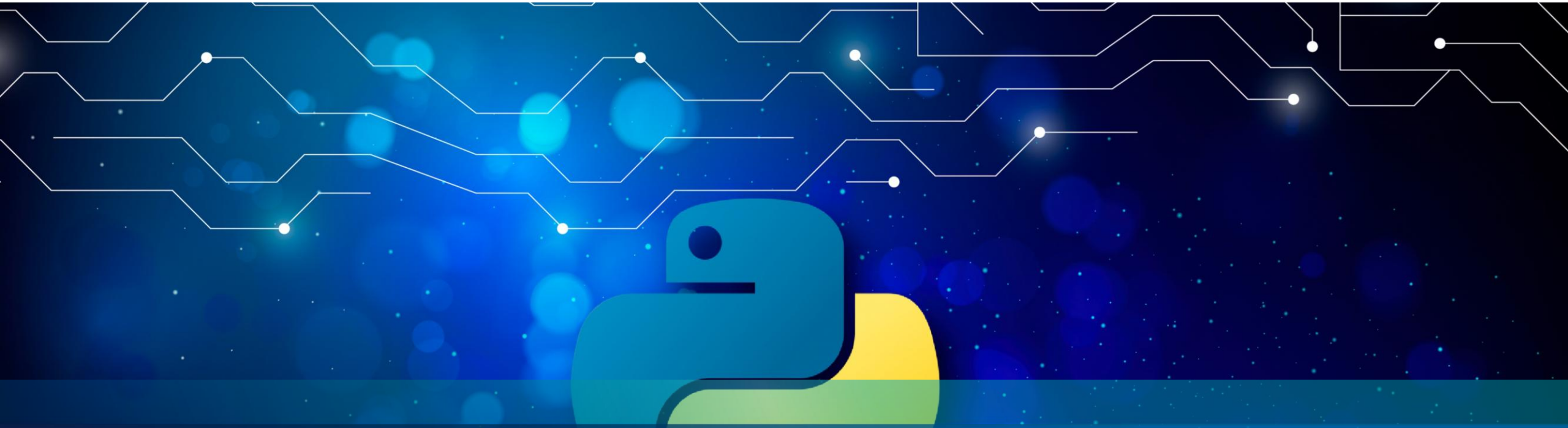
HOCHSCHULE HEILBRONN



› AKIB2 – GRUNDLAGEN DER KI-PROGRAMMIERUNG

07 – Beginner Python 6 – Functions, Modularization & PEP 8

P. Graf / TE / Angewandte KI

HOCHSCHULE HEILBRONN

› FUNCTIONS IN PYTHON

07 Beginner Python 6 | P. Graf / TE / Angewandte KI

WHAT IS A FUNCTION?

- A function is a reusable block of code that performs a specific task.
- Functions help break programs into smaller, manageable parts.
- Why use functions?
  - Increases code reusability
  - Improves readability & organization
  - Makes debugging easier

Function Definition

```
def greet(name):
    print("Hello, {}!".format(name))
greet("Pascal")
```

Functions make the code DRY (Don’t Repeat Yourself)!

TYPES OF FUNCTIONS & FUNCTION STRUCTURE

- Built-in functions
  - print(), len(), max(), round()
- User-defined functions
- Key Parts of a Function:
  - def → Defines the function
  - function name → The function's name
  - parameters → Optional inputs for the function
  - return → Optional output

Function Definition

```
def function_name(parameters):
    """Optional docstring explaining the function."""
    value = parameters**2
    return value  # (Optional)
```

Join at slido.com
#AKIB2_07

What are the main benefits of using functions in Python?

What are the correct components of a function definition in Python?

What happens if a function does not have a return statement?

What can be said about this function definition?

```
def add_numbers(x: int, y: int) -> int:
    return x + y
```

TechCampus

HOCHSCHULE HEILBRONN

› FUNCTION PARAMETERS

07 Beginner Python 6 | P. Graf / TE / Angewandte KI

POSITIONAL VS KEYWORD ARGUMENTS & DEFAULT VALUES

- **Positional Arguments:** Assigned by position in the function call, order matters and must match the function signature
- **Keyword Arguments:** Assigned using the parameter name, can appear in any order, as long as names are correct

Positional vs Keyword Arguments

```
def greet(name, age):
    print(f"Hello {name}, you are {age} years old.")

greet("Pascal", 30)  # name = "Pascal", age = 30
greet(age=30, name="Alice")
```

POSITIONAL VS KEYWORD ARGUMENTS & DEFAULT VALUES

- **Default Arguments:** Provide default values for parameters, these arguments become optional
- Positional arguments must come before keyword arguments
- Parameters with default values must come after those without in function definition

Default Arguments

```
def greet(name, age=18):
    print(f"Hello {name}, you are {age} years old.")

greet("Bob")          # uses default: age = 18
greet("Eve", 25)
```

ARBITRARY ARGUMENTS (*ARGS)

- Use *args to accept multiple positional arguments as a tuple
- Allows flexible function calls with arbitrary number of arguments

*args

```
def sum_numbers(*args):
    return sum(args)

print(sum_numbers(1, 2, 3))      # Output: 6
print(sum_numbers(10, 20, 30, 40)) # Output: 100
```

ARBITRARY KEYWORD ARGUMENTS (**KWARGS)

- Use **kwargs to accept multiple named arguments as a dictionary
- Useful when passing dynamic key-value pairs

*kwargs

```
def print_info(**kwargs):
    for key, value in kwargs.items():
        print(f"{key}: {value}")

print_info(name="Alice", age=25, city="New York")
```

DOCSTRINGS

- A docstring describes what a function does including:
  - Parameters (inputs the function accepts)
  - Return values (what the function outputs)

Docstrings

```
def add(a: int, b: int) -> int:
    """
    Adds two numbers and returns the sum.

    Parameters:
    a (int): The first number.
    b (int): The second number.

    Returns:
    int: The sum of a and b.
    """
    return a + b
print(add.__doc__)  # Prints the docstring
```

TYPE ANNOTATIONS IN FUNCTIONS

- Improves readability and helps with static analysis tools
- Type annotations are **not enforced** at runtime
- They are hints for developers and tools
- Use **Union[X, Y]** when a value could be either type X or type Y (with Python >3.10 you can use the | (pipe) operator instead.

Type Annotations

```
def function_name(param1: Type, param2: Type) -> ReturnType:
    ...

def greet(name: str | None = None) -> str:
    return "Hello {}".format(name) if name else "Hello Stranger"
```

Which function calls are valid?

```
def greet(name, greeting="Hello"):
    print(f"{greeting}, {name}!")
```

What is the correct order of arguments in a function definition?

What does *args collect?

Which statements about **kwargs are true?

Which docstring examples follow best practices?

What does this function return?

```
def mystery(a, b=2, *args, **kwargs):
    return a + b + len(args) + len(kwargs)

mystery(3, 4, 5, 6, x=1, y=2)
```

TechCampus

HOCHSCHULE HEILBRONN

› ADVANCED FUNCTION CONCEPTS

07 Beginner Python 6 | P. Graf / TE / Angewandte KI

RECURSIVE FUNCTIONS

- A recursive function calls itself within its definition
- Used for problems like factorial, Fibonacci sequence, and tree structures

Recursive Functions

```
def factorial(n):
    if n == 1:
        return 1
    return n * factorial(n - 1)

print(factorial(5))  # Output: 120
```

LAMBDA FUNCTIONS

- A lambda function is a small, single-expression function without a name
- Useful for short, one-time-use functions
- *lambda arguments: expression*
- Often used with *map, filter, sorted*

Lambda Expressions

```
numbers = [1, 2, 3, 4]
squared = list(map(lambda x: x ** 2, numbers))
print(squared)  # Output: [1, 4, 9, 16]
```

CALLBACKS

- A callback is a function passed as an argument to another function
- Enables custom behavior or deferred execution inside functions
- Commonly used in:
  - Sorting (sorted(..., key=callback))
  - Event handling (GUI, web servers)
  - Modular design and testing

Callback

```
def greet_user(name: str, formatter):
    print("Hello,", formatter(name))

def uppercase(name):
    return name.upper()

def polite(name):
    return "Mr./Ms. " + name

greet_user("Alice", uppercase)  # Hello, ALICE
greet_user("Bob", polite)       # Hello, Mr./Ms. Bob
```

CALLBACKS

Callback 2

```python
import time

def download_file(url, callback):
    print(f"Downloading {url}...")
    time.sleep(1)  # Simulate delay
    callback(f"{url} finished!")

def on_download_complete(message):
    print("Callback received:", message)

# Usage
download_file("https://example.com/file", on_download_complete)
```

TechCampus

HOCHSCHULE HEILBRONN

› MODULARIZATION

07 Beginner Python 6 | P. Graf / TE / Angewandte KI

MODULES

- Modularization means breaking code into multiple files (modules) for better structure.
- Why?
  - Increases Maintainability – Easier to read and update code.
  - Reusability – Use the same module across multiple projects.
  - Avoids Code Duplication – Write once, use multiple times.

Modularization

```
project/
|— main.py  # Main script
|— utils.py  # Helper functions
|— config.py  # Configuration settings
```

DIFFERENT WAYS TO IMPORT MODULES

Imports

```
import math  # Import the whole module
print(math.sqrt(16))

from math import sqrt  # Import a specific function
print(sqrt(16))

import math as m  # Use an alias
print(m.sqrt(16))

from math import *  # Import everything (not recommended)
print(sqrt(16))
```

WORKING WITH PACKAGES (__INIT__.PY)

- A package is a collection of modules inside a folder.
- A package needs an __init__.py file (can be empty) to be recognized (not strictly needed since Python 3.3).

Packages

```
mypackage/
|— __init__.py  # Required for a package
|— module1.py
|— module2.py

from mypackage import module1
module1.some_function()
```

What happens when you call this function?

```
def repeat(word: str, times: int = 3) -> str:
    return word * times

repeat("hi", times="3")
```

What does this function print?

```
def printer(*args, sep=", ", end="."):
    print(sep.join(args), end=end)

printer("a", "b", "c", sep="-", end="!\n")
```

What are benefits of modularizing code into functions and modules?

How do you import the function foo() from a file called tools.py in the same directory?

Which of the following statements about lambda functions are true?

What is printed by this code?

```
def mystery(n):
    if n == 0:
        return 0
    return n + mystery(n - 1)

print(mystery(3))
```

What is a potential danger of using recursion without a proper base case?

What is a callback function in Python?

Which line correctly passes a callback to calculate()?

```
def calculate(a, b, operation):
    return operation(a, b)

def subtract(x, y):
    return x - y
```

HOCHSCHULE HEILBRONN

PEP 8 – THE PYTHON STYLE GUIDE

07 Beginner Python 6 | P. Graf / TE / Angewandte KI

WHY FOLLOW PEP 8?

- What is PEP 8?
  - PEP 8 stands for Python Enhancement Proposal 8 and defines the official style guide for Python.
  - Helps maintain consistency, readability, and best practices across projects.
- Why is PEP 8 Important?
  - Improves code readability and makes collaboration easier.
  - Ensures code is easy to understand, debug, and maintain.
  - Helps avoid common pitfalls & bad practices.

NAMING CONVENTIONS

- Variable & Function Names - Use **snake_case**
- Class Names - Use **PascalCase** (CapitalizedWords format)
- Constants - Use **UPPER_CASE_WITH_UNDERSCORES**
- Private Variables - Use a **single underscore _var**

OTHER RULES

- Use 4 spaces per indentation level
- Keep lines ≤79 characters and structure imports properly
- Write clear docstrings & comments
- Use two blank lines between top-level functions/classes.
- Use one blank line inside functions to separate logic.

TechCampus

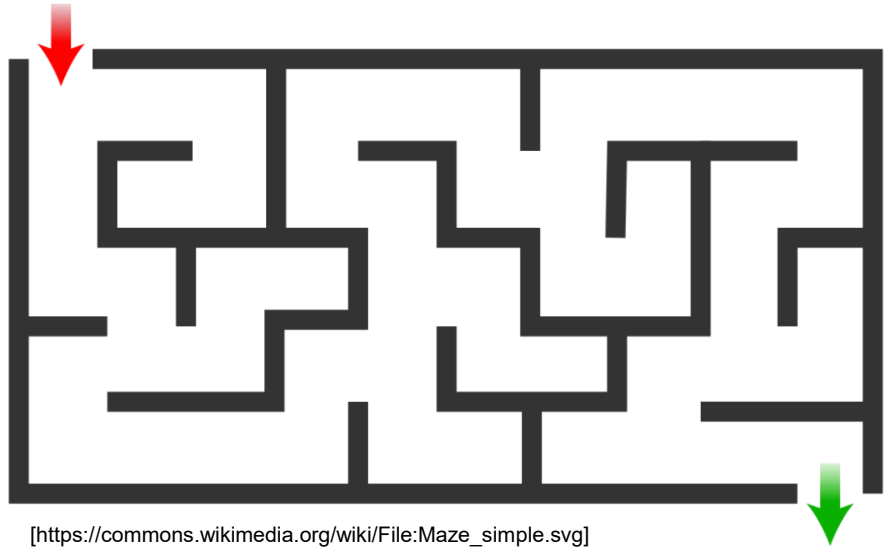
HOCHSCHULE HEILBRONN

› SEARCH / PATHFINDING ALGORITHMS

07 Beginner Python 6 | P. Graf / TE / Angewandte KI

PATHFINDING & SEARCHING

- Search algorithms explore graphs or problem spaces to find a path from a start to a goal
- Can be uninformed (e.g. BFS, DFS) or informed (e.g. A* uses heuristics)
- Pathfinding algorithms focus on finding the shortest or most efficient route
- Key trade-offs include speed, memory usage, and path optimality
- Common applications: maps, games, navigation

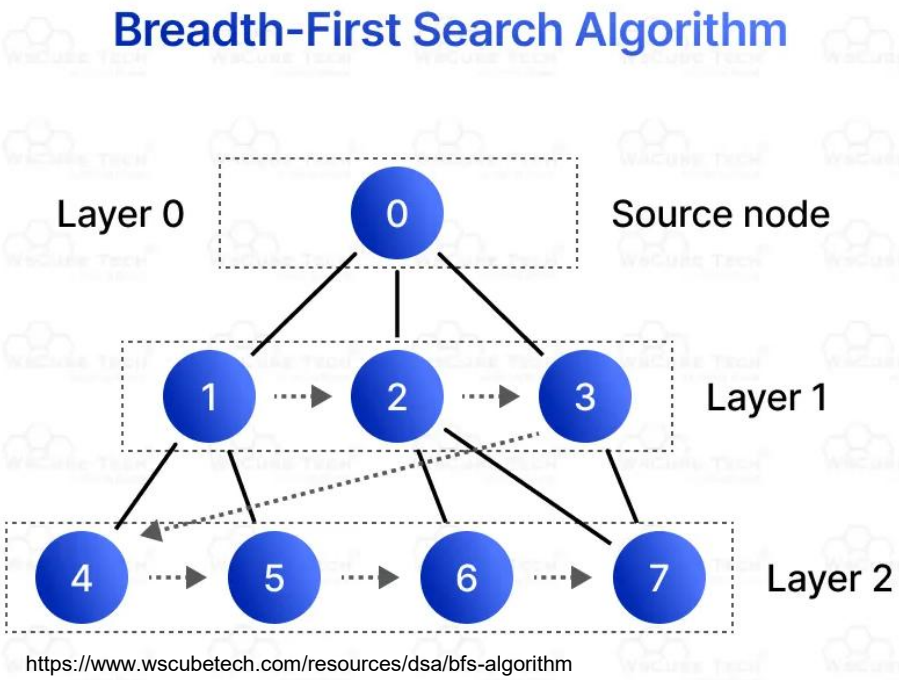


[https://commons.wikimedia.org/wiki/File:Maze_simple.svg]

```
maze = [
    [0, 1, 0, 0, 0],
    [0, 1, 0, 1, 0],
    [0, 0, 0, 1, 0],
    [1, 1, 0, 0, 0],
    [0, 0, 0, 1, 0],
]
```

BREADTH-FIRST SEARCH ALGORITHM (BFS)

- Explores all neighbors at the current depth before moving deeper (level-by-level search)
- Uses a queue (FIFO) to manage nodes to visit
- Guarantees the shortest path in unweighted graphs or mazes
- Complete and optimal, but can be memory-heavy in large graphs



https://www.wscubetech.com/resources/dsa/bfs-algorithm

BREADTH-FIRST SEARCH ALGORITHM (BFS)

Algorithm 3 BFS Maze Solver

```
1: procedure BFS_SOLVE_MAZE(maze, start, goal)
2:     Q ← empty queue
3:     enqueue start into Q
4:     visited ← {start}
5:     came_from ← dictionary with start ↦ None
6:     while Q is not empty do
7:         current ← dequeue from Q
8:         if current = goal then
9:             return RECONSTRUCT_PATH(came_from, start, goal)
10:        end if
11:        for all neighbor ∈ GET_NEIGHBORS(current) do
12:            if neighbor is in bounds and maze[neighbor] = 0 and neighbor ∉
   visited then
13:                enqueue neighbor into Q
14:                add neighbor to visited
15:                came_from[neighbor] ← current
16:            end if
17:        end for
18:    end while
19:    return empty list
20: end procedure
```

TechCampus

HOCHSCHULE HEILBRONN

› PYCHARM HOTKEY CHEAT SHEET

07 Beginner Python 6 | P. Graf / TE / Angewandte KI

PYCHARM HOTKEY CHEAT SHEET

➢ Cheat sheet can be downloaded from Ilias

PyCharm

Find any action inside the IDE	Ctrl + Shift + A

CREATE AND EDIT

Show intention actions	Alt + Enter
Basic code completion	Ctrl + Space
Smart code completion	Ctrl + Shift + Space
Type name completion	Ctrl + Alt + Space
Complete statement	Ctrl + Shift + Enter
Parameter information / context info	Ctrl + P / Alt + Q
Quick definition	Ctrl + Shift + I
Quick / external documentation	Ctrl + Q / Shift + F1
Generate code	Alt + Insert
Override / implement members	Ctrl + O / Ctrl + I
Surround with…	Ctrl + Alt + T
Comment with line comment	Ctrl + /
Extend / shrink selection	Ctrl + W / Ctrl + Shift + W
Optimize imports	Ctrl + Alt + O
Auto-indent lines	Ctrl + Alt + I
Cut / Copy / Paste	Ctrl + X / Ctrl + C / Ctrl + V
Copy document path	Ctrl + Shift + C
Paste from clipboard history	Ctrl + Shift + V
Duplicate current line or selection	Ctrl + D
Move line up / down	Ctrl + Shift + Up / Down
Delete line at caret	Ctrl + Y
Join / split line	Ctrl + Shift + J / Ctrl + Enter
Start new line	Shift + Enter
Toggle case	Ctrl + Shift + U
Expand / collapse code block	Ctrl + NumPad + / -
Expand / collapse all	Ctrl + Shift + NumPad + / -
Save all	Ctrl + S

VERSION CONTROL

VCS operations popup…	Alt + `
Commit	Ctrl + K
Update project	Ctrl + T
Recent changes	Alt + Shift + C
Revert	Ctrl + Alt + Z
Push…	Ctrl + Shift + K
Next / previous change	Ctrl + Alt + Shift + Down / Up

MASTER YOUR IDE

Find action…	Ctrl + Shift + A
Open a tool window	Alt + [0-9]
Synchronize	Ctrl + Alt + Y
Quick switch scheme…	Ctrl + `
Settings…	Ctrl + Alt + S
Jump to source / navigation bar	F4 / Alt + Home
Jump to last tool window	F12
Hide active / all tool windows	Shift + Esc / Ctrl + Shift + F12
Go to next / previous editor tab	Alt + Right / Alt + Left
Go to editor (from a tool window)	Esc
Close active tab / window	Ctrl + Shift + F4 / Ctrl + F4

FIND EVERYTHING

Search everywhere	Double Shift
Find / replace	Ctrl + F / R
Find in path / Replace in path	Ctrl + Shift + F / R
Next / previous occurence	F3 / Shift + F3
Find word at caret	Ctrl + F3
Go to class / file	Ctrl + N / Ctrl + Shift + N
Go to file member	Ctrl + F12
Go to symbol	Ctrl + Alt + Shift + N

NAVIGATE FROM SYMBOLS

Declaration	Ctrl + B
Type declaration (JavaScript only)	Ctrl + Shift + B
Super method	Ctrl + U
Implementation(s)	Ctrl + Alt + B
Find usages / Find usages in file	Alt + F7 / Ctrl + F7
Highlight usages in file	Ctrl + Shift + F7
Show usages	Ctrl + Alt + F7

REFACTOR AND CLEAN UP

Refactor this…	Ctrl + Alt + Shift + T
Copy… / Move…	F5 / F6
Safe delete…	Alt + Delete
Rename…	Shift + F6
Change signature…	Ctrl + F6
Inline…	Ctrl + Alt + N
Extract method	Ctrl + Alt + M
Introduce variable/ parameter	Ctrl + Alt + V / P
Introduce field / constant	Ctrl + Alt + F / C
Reformat code	Ctrl + Alt + L

6597

ANALYZE AND EXPLORE

Show error description	Ctrl + F1
Next / previous highlighted error	F2 / Shift + F2
Run inspection by name…	Ctrl + Alt + Shift + I
Type/ call hierarchy	Ctrl + H / Ctrl + Alt + H

NAVIGATE IN CONTEXT

Select in…	Alt + F1
Recently viewed / Recent locations	Ctrl + E / Ctrl + Shift + E
Last edit location	Ctrl + Shift + Back
Navigate back / forward	Ctrl + Alt + Left / Right
Go to previous / next method	Alt + Up / Down
Go to line / column…	Ctrl + G
Go to code block end / start	Ctrl +] / [
Add to favorites	Alt + Shift + F
Toggle bookmark	F11
Toggle bookmark with mnemonic	Ctrl + F11
Go to numbered bookmark	Ctrl + [0-9]
Show bookmarks	Shift + F11

BUILD, RUN, AND DEBUG

Run context configuration	Ctrl + Shift + F10
Run / debug selected configuration	Alt + Shift + F10 / F9
Run / debug current configuration	Shift + F10 / F9
Step over / into	F8 / F7
Smart step into	Shift + F7
Step out	Shift + F8
Run to cursor / Force run to cursor	Alt + F9 / Ctrl + Alt + F9
Show execution point	Alt + F10
Evaluate expression…	Alt + F8
Stop	Ctrl + F2
Stop background processes…	Ctrl + Shift + F2
Resume program	F9
Toggle line breakpoint	Ctrl + F8
Toggle temporary line breakpoint	Ctrl + Alt + Shift + F8
Edit / view breakpoint	Ctrl + Shift + F8

jetbrains.com/pycharm
jetbrains.com/help/pycharm
blog.jetbrains.com/pycharm
@pycharm

JET BRAINS

THANK YOU!

Please contact for questions:

Pascal Graf
Fakultät TE | Angewandte KI
pascal.graf@hs-heilbronn.de