

Python Kurs

0. Grundlagen

- **Testing for Equality == Testing for inequality !=**
- assign a value to a variable =
- **Integer Division //**
- **Getting the type of a variable: type ()**
- **List** [blau, rot]
- length (using the **len()** function)
- Accesssing Items in a List: `my_list[1]`, Using negative indices, you can access the last (-1), second-to-last (-2)
- `my_list[0:3]`-> elements at index 0, 1, and 2
- `my_list[0:6:2]` # like this: [including start, excluding stop, step size]
- Checking for Membership in a List: `in`, `not in`
- `.append()`, `.remove()`, `.insert`
- **pop()** by default the last element is removed
- **Sets** care only about membership: `set_1 = {1,4, "apple", "apple", 4}`
- The union of sets: using the `|` sign
- **Dictionary**: stores key-value pairs, curly braces `{}` key-value pairs are separated by a **colon** `:` `my_dict = {"key_1": "value_1", "key_2": "value_2"}`
- **Keys**: immutable data type, **Values**: Values can be of any data type
- Removing Items from a Dictionary: `del`
- **Operating on Logical Values**: `and`, `or`, `not`, `^` exclusive or
- **Control Flow**:
 - **If** specifies a condition after which a certain block of code is executed or not
 - **Elif** previous if statement is not true, but the elif statement itself is true
 - **Else** neither of the previous statements is true
- **For loop**: operate over a specified set of elements, number of iterations is known ahead of time
- **While loop**: execute a block of code as long as a specified condition remains true, useful when the number of iterations is not known in advance
- **range(start, stop, step)**: doing something a predefined number of times

Beispielfragen

(1) What is the output of the following code?

```
x = 5
y = "5"
print(x + y)
```

- a) 10
- b) 55
- c) TypeError
- d) 5

c)

(2) What is the value of x after executing the following code?

```
x = 10
if x > 5:
    x = x * 2
else:
    x = x - 2
```

- a) 5
- b) 10
- c) 20
- d) 8

c)

(3) Which of the following is an example of a correct if statement in Python?

- a) `if x == 10 then:`
- b) `if x == 10:`
- c) `if (x = 10):`
- d) `if x => 10:`

Correct Answer:

b)

(4) What happens if you try to access an index that is out of range for a list?

- a) It returns `None`
- b) It raises an `IndexError`
- c) It returns an empty string
- d) It raises a `ValueError`

Correct Answer:

b)

(5) What does `my_list[1:5:2]` do if `my_list = [10, 20, 30, 40, 50, 60]`?

- a) Returns all elements from index 1 to 5
- b) Returns elements at every second index from index 1 to 5
- c) Returns a single element at index 2
- d) Raises an `IndexError`

Correct Answer:

b)

(6) What will the following code print?

```
x = [1, 2, 3]
y = x
y.append(4)
print(x)
```

- a) `[1, 2, 3]`
- b) `[1, 2, 3, 4]`
- c) `Error`
- d) `[4]`

Correct Answer:

b)

(7) What is the purpose of a for-loop in Python?

- a) To repeatedly execute a block of code until a condition is True
- b) To execute a block of code a specific number of times or iterate over a sequence
- c) To define a function
- d) To check multiple conditions at once

Correct Answer:

b)

(8) What does the `and` operator do in Python?

- a) Checks if both conditions are True
- b) Assigns a value to two variables at once
- c) Compares two strings for equality
- d) Ensures that one condition is True

Correct Answer:

a)

(9) Why are comments used in Python code?

- a) To add debugging information that helps the program run
- b) To temporarily disable sections of code
- c) To explain what the code is doing, making it easier for humans to understand
- d) To mark important lines of code for the interpreter

Correct Answer:

c)

practically also b), but it's bad practice.

(10) Which of the following best describes a variable in Python?

- a) A reserved word used for built-in functions
- b) A name that holds data and can be changed during program execution
- c) A value that cannot change once assigned
- d) A keyword that defines a function

Correct Answer:

b)

(11) What is the main difference between `==` and `=` in Python?

- a) `==` is used to assign values, and `=` is used for comparison
- b) Both are used for comparison, but `=` is stricter
- c) `==` checks for the same data type, while `=` allows conversion
- d) `==` compares two values, and `=` assigns a value to a variable

Correct Answer:

d)

1. Functions and Classes

- **DRY! (standing for "Don't Repeat Yourself")**
- Def, my_function_name, parameters e.g. def my_function():
- **Return** values: defines which variable(s) is/are returned
- Documentation: **docstrings** (brief description of what your function does, inside triple quotes (""" """), or ('' ''))
- **Mutable types:** objects whose values can be changed 'in place' after the object is created e.g. lists
- **Immutable types:** These are objects whose values cannot be changed in place

Immutable Types	Mutable Types
int (Integer)	
float (Floating-point)	
complex (Complex number)	
bool (Boolean)	
str (String)	
tuple (Tuple)	list (List)
frozenset (Frozen set)	set (Set)
	dict (Dictionary)

- Identity of variables: **id()**, keyword **is** test for identity in terms of memory mapping
- a_tuple = (1,2,3)
- Updating variables:
 - 1. variable += something (modifies object in place)
 - 2. variable = variable + something (creates new object)
- **Classes** allow you to group related properties and functions into a single structure.
-> by keyword class followed by the Name of the class and : (class Student:)
- **The __init__() method:** constructor of the class, set individual values when we create an object, first parameter is always **self**
 - def __init__(self, full_name):
self.full_name = full_name
- **class variables:** are the same for all objects, **instance variables:** differ between all instances of the class
- **set ()** instance variable that we want to create later, do input checking, **get()** retrieve the value of an instance variable
- **__str__()** specifies the string representation of our Objects
- **Data classes** make defining simple classes for data storage easier and cleaner
 - __init__() --> the constructor
 - __repr__() --> an automatic object representation with all attributes
 - __eq__() --> an object comparison
- **Type hints**
 - : **type** after the variable name
 - : **any** variable can hold any type of value
- **Field ()** allows fine-tuned control over attributes
- Making Dataclasses Immutable with frozen=True
- Excluding Fields from repr() in Dataclasses repr=false

(1)

Which one of these is a valid function definition?

(a)

```
def fun():  
    print("hello")
```

(b)

```
def fun()  
    print("hello")  
    return
```

(c)

```
def fun():  
    print("hello")
```

1c)

(2)

What is the value of `avg` after running the following code?

```
def avg_grade(grades):  
    """Produces the average of a list of numerical grades (1 = best, 4 = worst)."""  
    if not grades:  
        return 0  
    avg_grade = sum(grades) / len(grades)  
  
grades = [1, 4, 2, 3]  
avg = avg_grade(grades)
```

(a) None

(b) 2.5

(c) SyntaxError: expected return statement

2a)

(3)

What is the value of `my_list` after the following block of code?

```
my_list = [0, 1, 2, 3, 4]  
  
def modify_list(number_list):  
    number_list = [-2, -1, 0]  
    return number_list  
  
modified_list = modify_list(my_list)
```

(a) [-2, -1, 0]

(b) [0, 1, 2, 3, 4]

3b)

Classes and Objects

(4)

Which of the following code blocks correctly defines a class named `Car` with an attribute `color` and a method `display_color`?

(a)

```
class Car:  
    color = "red"  
  
    def display_color():  
        print(self.color)
```

(b)

```
class Car:  
    def __init__(self, color):  
        self.color = color  
  
    def display_color(self):  
        print(self.color)
```

(c)

```
Car:  
    def __init__(self, color):  
        self.color = color  
  
    def display_color(self):  
        print(self.color)
```

4b)

(5)

What is the output of the following line of code?

```
class Dog:
    def __init__(self, name):
        self.name = name

    def bark(self):
        return f"{self.name} says woof!"

my_dog = Dog("Buddy")
print(my_dog.bark())
```

- (a) 'Buddy says woof!'
- (b) 'Dog says woof'
- (c) None

5a)

(6)

The following implementation of a `Rectangle` class contains several issues. Identify what needs to be fixed to make it work correctly. Multiple answers are possible.

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

    def area(self):
        return width * height
```

- (a) The `width` and `height` attributes are not properly assigned to the instance.
- (b) The `area` method should use `self.width` and `self.height` instead of `width` and `height`.
- (c) The class lacks a method to calculate the perimeter of the rectangle.
- (d) The parameters `width` and `height` should be declared as class variables.

6a,b)

Modules

(7)

Which of the following code blocks correctly imports the `math` module and uses it to calculate the square root of 25? Multiple answers are possible.

(a)

```
import math
result = math.sqrt(25)
print(result)
```

(b)

```
from math import sqrt
result = sqrt(25)
print(result)
```

(c)

```
import math.sqrt
result = math.sqrt(25)
print(result)
```

(d)

```
import math as m
result = math.sqrt(25)
print(result)
```

7a,b)

2. Numpy

- `[0,3]` First element in fourth column, `[0,:]` Get all elements in the first row, `[:,1]` Get all elements in the second column
- **all entries must be of the same data type!**
- **object` dtype** allows you to store different types
- `.astype()` convert array between types
- **Reshaping arrays:** transpose, flatten, reshape
- **Broadcasting** enables NumPy to work with arrays of different shapes in element-wise operations
- **Mutability:** slicing returns a copy
- check whether something is a view or a copy **.base -> None** if something is a copy

3. Matplot

- `plt.plot(numbers), plt.show()`
- **Figure:** The main plotting space, created with `plt.figure()`
- **Axes:** An Axis object (singular of Axes) of a figure in Matplotlib is a part of the figure that contains a single (sub-)plot itself.
- **Plotting Data:** Commands like `plt.plot()`, `plt.scatter()` create line or point plots.
- Styling: `plt.plot(x,y, color = "blue")`
- **Supporting Elements:** label, title, legend
- **Displaying the Plot:** `plt.show()` renders the figure, displaying all the elements you've added.

4. Timeseries

- Neural data are typically stored as a **2D array**, where each **row corresponds to one channel** and each **column corresponds to a time point**: $n_channels \times n_timepoints$
- Create **time vectors** that define discrete sampling points
- **White noise** is a signal composed of random, uncorrelated samples
- In the frequency domain, white noise has equal power at all frequencies
- **1/f slope:** show higher power at low frequencies and lower power as frequency increases
- **Brown noise:** integrate white noise, drifts more smoothly in time
- **Sampling Rate** The (average) rate at which individual samples are acquired
- **Regular vs Irregular Sampling Interval**
- **Univariate vs Multivariate** Whether only a single channel is acquired, or whether concurrent data from multiple channels is acquired
- **Autocorrelation** values at time $t + k$ are correlated with values at time t , indicating dependence of future on past observations.
- **Periodicity or Seasonality**
- **Stationarity** statistical properties (mean, variance, autocorrelation) remain constant over time.
- **Ergodicity**, i.e. statistical properties estimated from a single sufficiently long time series are representative of the entire process

5. Event-Related Potentials

P300 -> positivity approximately peaking after 300 ms

Analysis Steps:

1. Data Preprocessing
2. Epoching
3. Baseline Correction
4. Artifact Rejection
5. Averaging
6. P300 Identification
7. Plotting ERP
8. Interpretation

6. MNE:

Compute an Event-Related Potential (ERP)[1](#)

- **Data types**

Stage	MNE class	Description
Continuous data	<code>mne.io.Raw</code>	The full continuous EEG recording. Contains all channels and the time series for the whole session.
Trial-based data	<code>mne.Epochs</code>	Segments (short time windows) cut out from the Raw data around each event — one per trial.
Averaged data	<code>mne.Evoked</code>	The average of multiple epochs belonging to the same condition — the ERP.