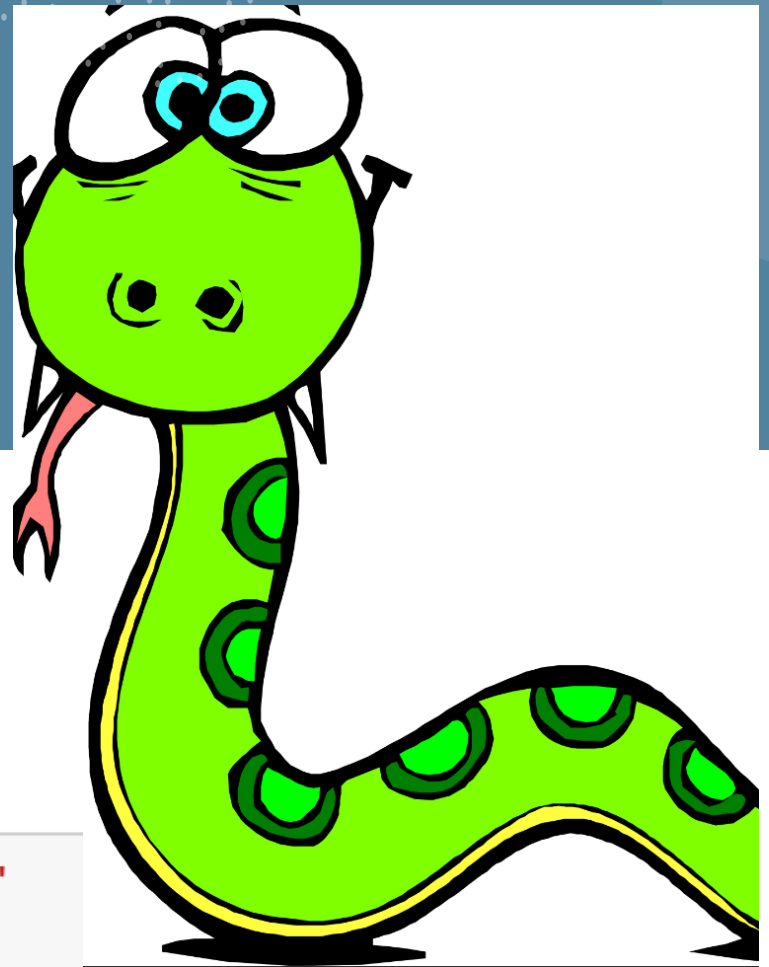


Welcome to my 3rd Python Lecture

Lutz Plümer

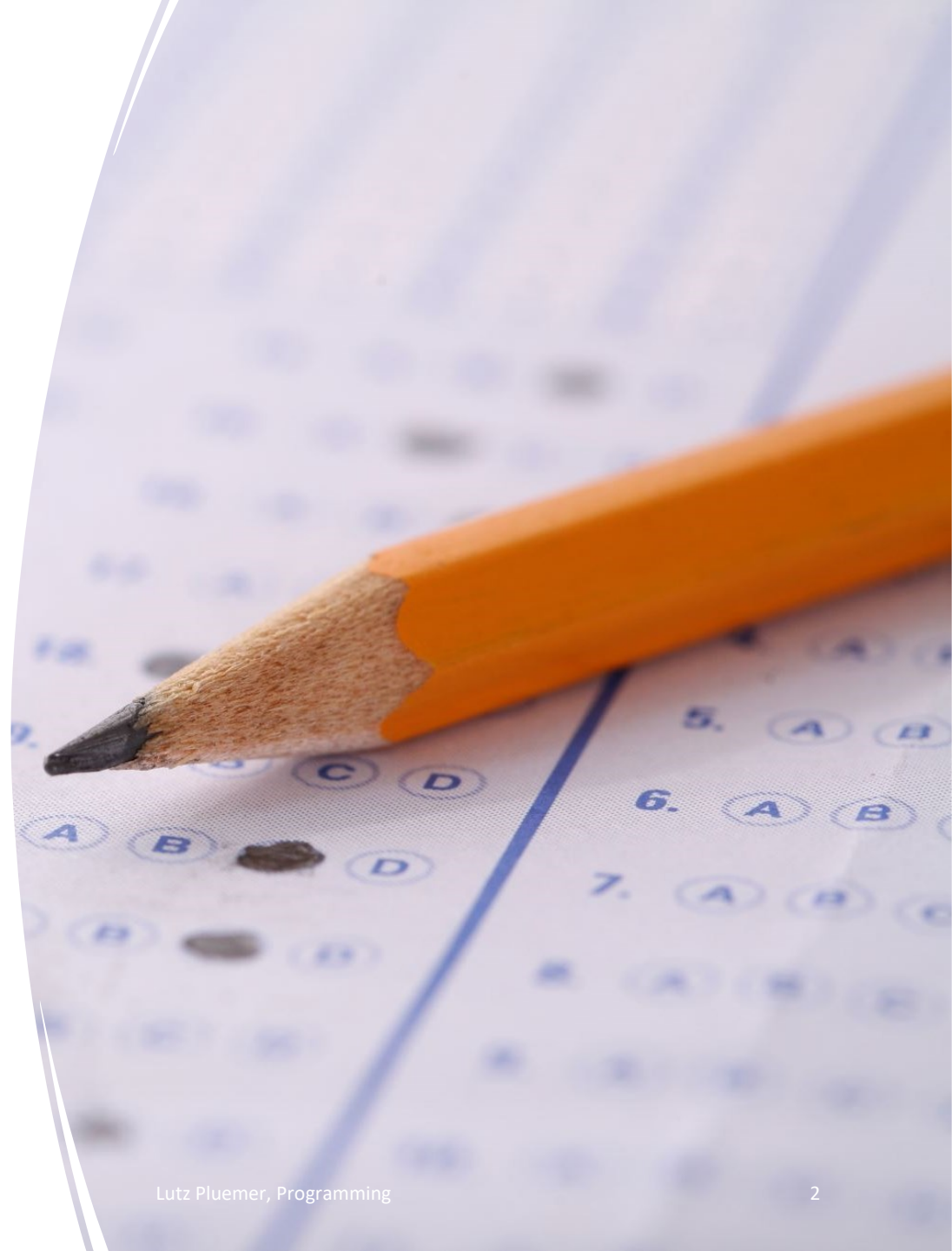


```
▶ WelcomeToMyLecture = "欢迎来到我的讲座"  
print(WelcomeToMyLecture)
```

欢迎来到我的讲座

Midterm Exam

April 13



Requirements and Preparation

Requirements:

- You are able to solve the same resp. similar exercises as in your homework
- But in an Exam setting: for your own, no other aids

Preparation:

- But: you can – and should prepare – one sheet of paper, both sides, with your own notes from your own hand
- You should have the most important **procedures** (print), **functions** and (list) **methods** in mind, as well as **loops**
- You should work through the lectures and the **exercises again** and again
- You need to be familiar with **Jupyter**
- You will use your own **laptops**, so the work will be rather familiar for you.
- Individual Assessment

Rules for the Midterm Exam (I)

1. The exam will take place on April 13, 2023 and will last 2 hours (120 minutes) in X4350

2. Each student brings his/her own **laptop** with **Jupyter** installed and running reliably.

This must be **ensured** before the exam.

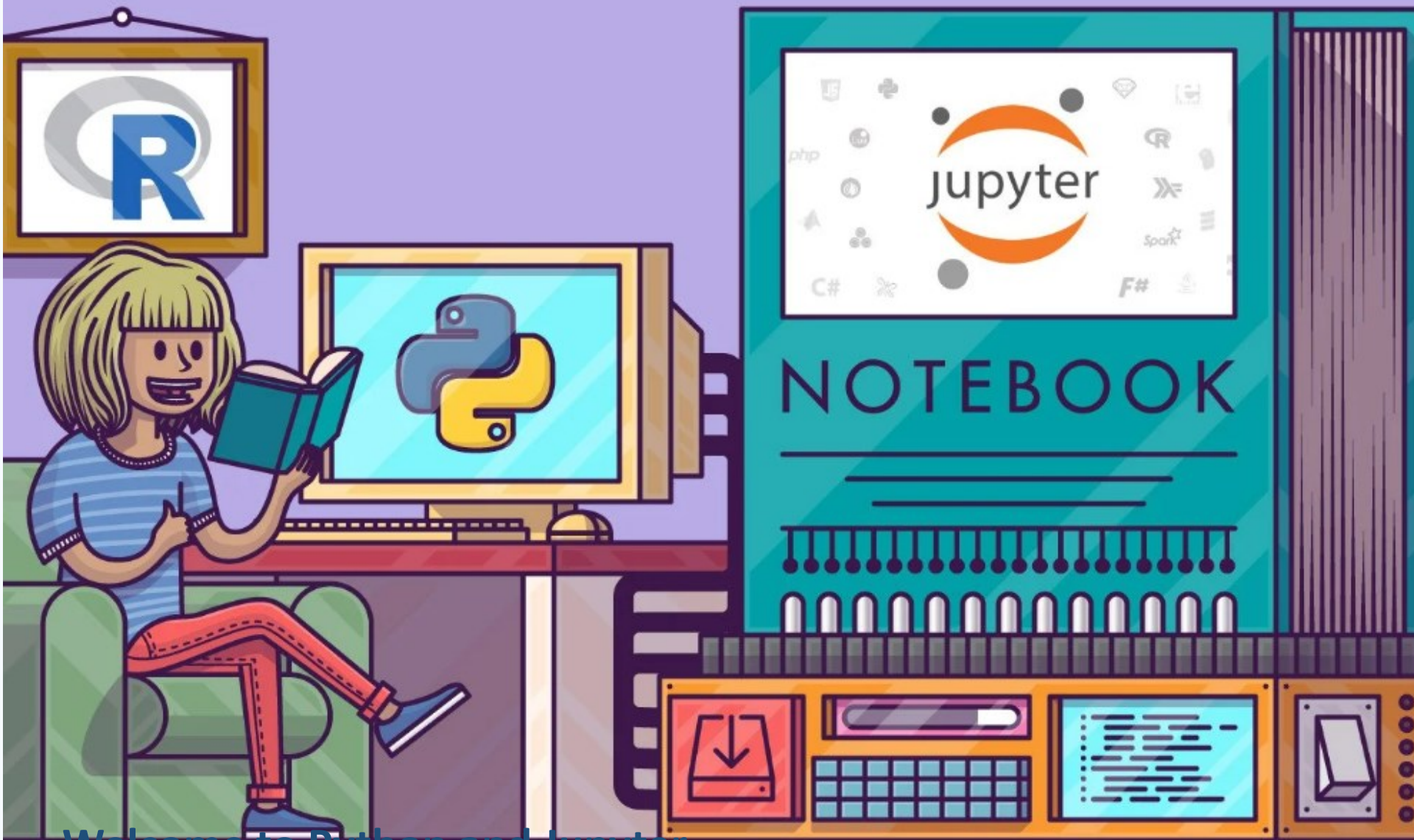
3. Each student is allowed to bring **one handwritten DIN A 4 sheet**, written on both sides, with own notes about the lecture. No printout, no copy. The sheet must contain the name and ID of the student on both sides in the upper right corner.

4. The exam questions will be handed out in written form. They are edited in a notepad in Jupyter and saved as a file consisting of the student's name and matriculation number, e.g.

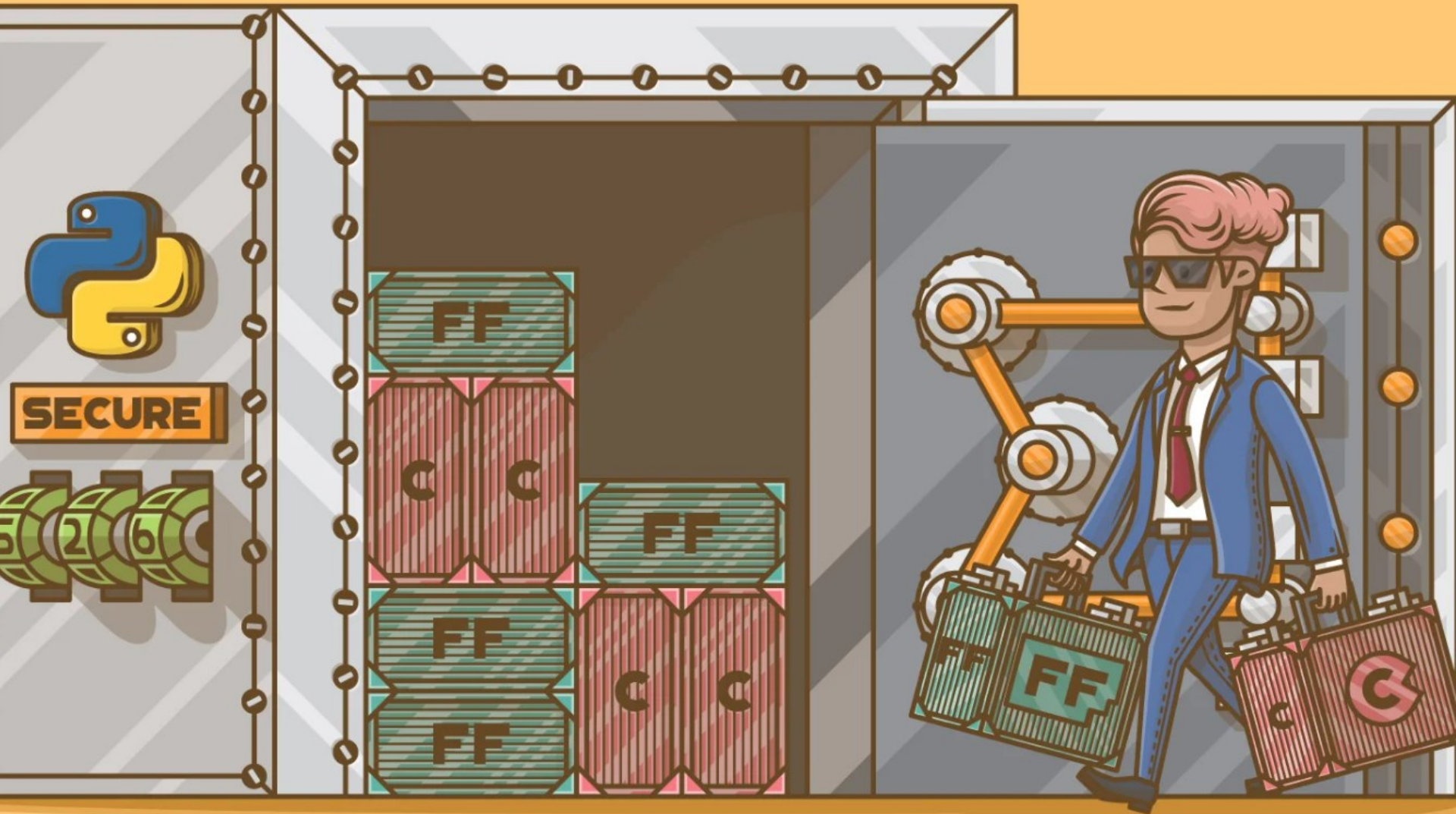
Apart from Jupyter and the student's own handwritten DIN A 4 sheet, no other aids may be used.

Rules for the Midterm Exam (II)

- In particular, the following are not allowed
 - a. Any form of contacting others, either in person or by WeChat, email, etc.
 - b. Use of the Internet to search for solutions
 - c. Use of smartphone. Smartphone is turned off.
 - d. The WiFi must not be used while working on the written assignments.
- 6. The end of the exam will be documented by signing the exam paper. Afterwards, the notebook with the solution of the tasks will be downloaded in the specified file format via **hotspot** of the three counsellors. (There is no Wifi during Exam)
- 7. The notepad has in the first line name and id of the student, in a markdown cell.
- 8. In case of attempted **cheating**, the exam is considered **failed**.



Welcome to Python and Jupyter



Welcome to Python's Lists and Functions

Lutz Pluemer, Programming

Real

Done so far - Keywords

- Syntax and Semantics
- Assignments and Expressions
- Types
- While- and For-Loops
- Factorial
- Greatest Common Divisor

- Today: More on Lists and Functions
- **Functions (and Modules) structure Programs**

What is a Function / Procedure in Python?

A function is a block of **Code** defined with a **name**

It is **defined** first.

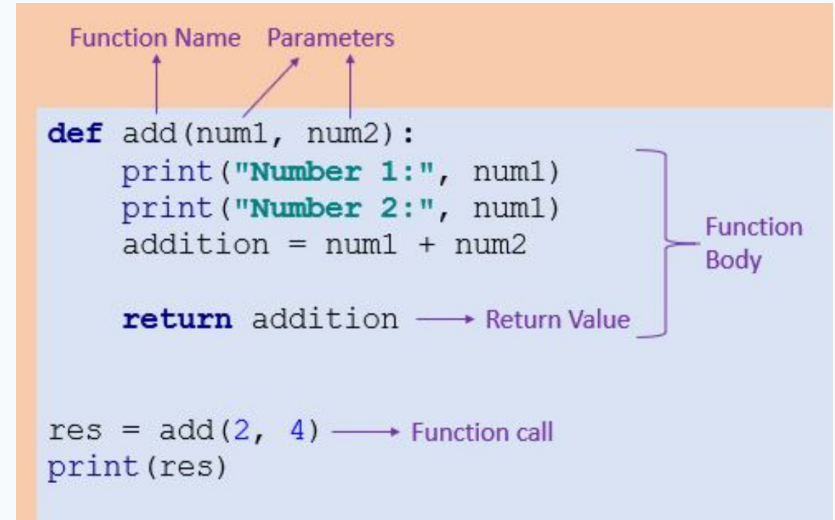
It runs whenever it is **called**

You pass **data**, known as **parameters**, or **arguments**, into a function

Functions deliver a **result**

Procedures are similar, they do an action, but **don't** deliver a result. They do some useful actions (such as print)

Functions reuse code: you **define** it **once** and **apply** it several times.



Function Name Parameters

```
def add(num1, num2):  
    print("Number 1:", num1)  
    print("Number 2:", num1)  
    addition = num1 + num2
```

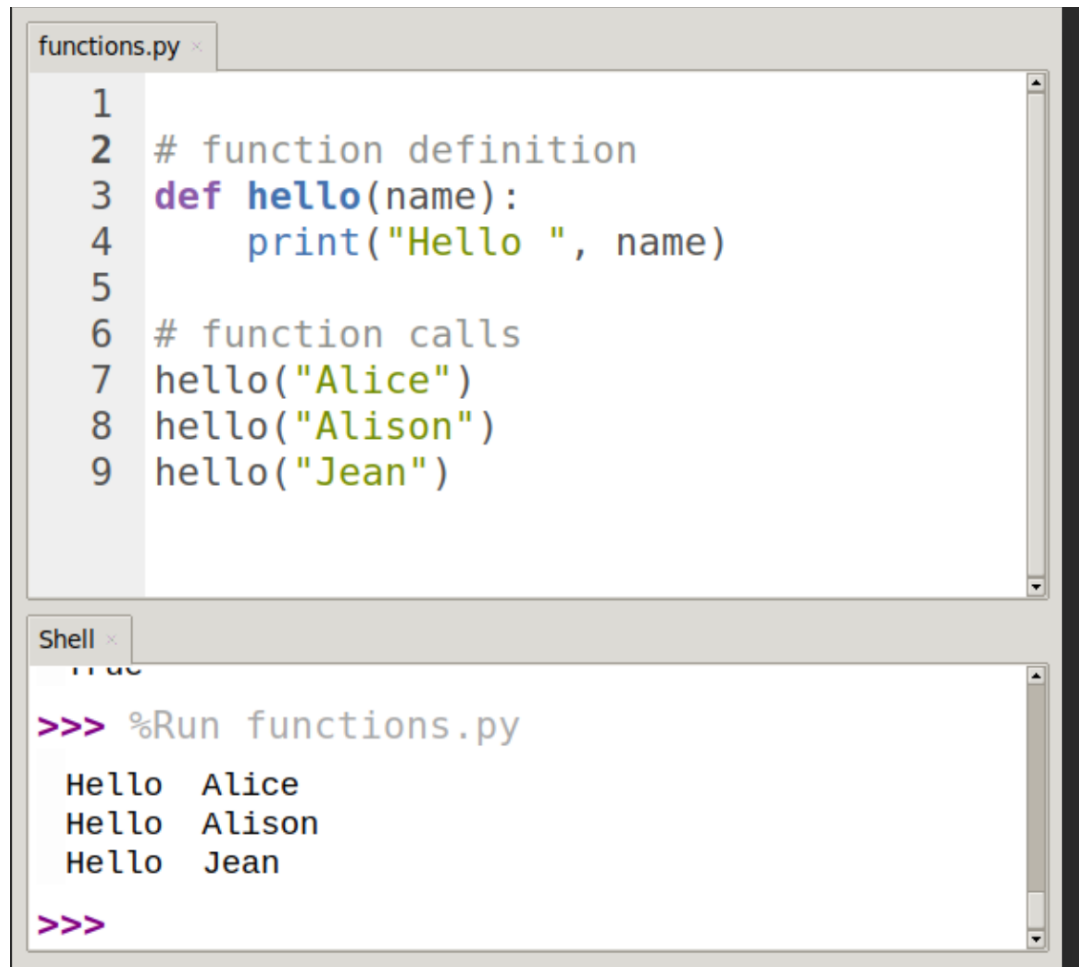
```
    return addition
```

→ Return Value

} Function
Body

```
res = add(2, 4) → Function call  
print(res)
```

Reuse Code



The image shows a screenshot of a Python IDE with two panels. The top panel, titled 'functions.py', contains a Python script. The bottom panel, titled 'Shell', shows the output of running the script.

```
1
2 # function definition
3 def hello(name):
4     print("Hello ", name)
5
6 # function calls
7 hello("Alice")
8 hello("Alison")
9 hello("Jean")
```

The Shell panel shows the command prompt with the command `>>> %Run functions.py` and the output:

```
Hello Alice
Hello Alison
Hello Jean
>>>
```

```
▶ def sum_of(n) :  
    sum = 0  
    for i in range(n+1) :  
        sum += i  
    return sum  
  
print(sum_of(5))
```

15

See how it
works

```
▶ def sum_of(n) :  
    sum = 0  
    for i in range(n+1) :  
        sum += i  
        print(sum)  
    return sum  
  
print(sum_of(5))
```

0
1
3
6
10
15
15

Look at GCD

```
▶ def GCD(x,y):  
    while y > 0 :  
        x, y = y, x % y  
    return x  
print(GCD(30,35))
```

5

```
▶ print(GCD(2*3*5*7,5*7*11*13))
```

Built-in Functions, Keyword Arguments

- We have already seen and used Python Built-in Functions such as `type`, `int`, `str`, ... , but especially **`print`**
- note that `print` always finishes with a new line and carriage return, this is the default value of the parameter named **`end`**
- This is the default value of `print`, but you can change this default value **`end = " "`**, using redefining the parameter name **`end`**
- Above, you can use an arbitrary number of arguments
- But – **how** and **why** does that work?
See it bigger on the next slide

```
print("Hello Python")  
print("Hello Lutz")
```

```
Hello Python  
Hello Lutz
```

```
print("Hello Python", end = " ")  
print("Hello Lutz")
```

```
Hello Python Hello Lutz
```

```
print("Hello Lutz", "Hello Bob", "Hello Alice", sep = " ++")
```

```
Hello Lutz ++ Hello Bob ++ Hello Alice
```

```
print("Hello Python")  
print("Hello Lutz")
```

Hello Python
Hello Lutz

```
print("Hello Python", end = " ")  
print("Hello Lutz")
```

Hello Python Hello Lutz

```
print("Hello Lutz", "Hello Bob", "Hello Alice", sep = " ++ ")
```

Hello Lutz ++ Hello Bob ++ Hello Alice

Function Parameters (or Arguments)

- Default argument
- Keyword arguments (named arguments)
- Positional arguments
- Arbitrary arguments (variable-length arguments `*args` and `**kwargs`)

Default arguments

- A [default argument](#) is a parameter that assumes a default value if a **value** is **not** provided in the function call for **that argument**.

```
▶ def myFun(x, y=50):  
    print("x: ", x)  
    print("y: ", y)
```

```
▶ myFun(10)
```

```
x:  10  
y:  50
```

```
▶ myFun(10,20)
```

```
x:  10  
y:  20
```

```
▶ myFun(10,y= 100)
```

```
x:  10  
y: 100
```

```
▶
```

Keyword Arguments versus Positional Arguments

- Python allows the programmer to specify the argument name with values so that he does not need to remember the order of parameters.

```
▶ def division(x,y):  
    print(x/y)  
    return x/y
```

```
▶ division(10,3)
```

```
] : 3.3333333333333335
```

```
▶ division(y = 4, x = 12)
```

```
] : 3.0
```

Arbitrary Keyword Arguments

Arbitrary Keyword Arguments, such as `*args` can pass a variable number of arguments to a

There is also something like `**kwargs` in Python which is beyond our scope here

```
| def myPrintArgs(*argv):  
    for arg in argv:  
        print(arg)
```

```
| myPrintArgs('Hello', 'Welcome', 'to', 'my', 'Lecture' )
```

```
Hello  
Welcome  
to  
my  
Lecture
```


Methods versus Functions

- In contrast to Functions, there are also **Methods**
- Methods come with Objects, such as **Lists**
- And they have a different **Syntax**
- Have a look at **append**, which appends an element to the **end** of a list
- Note that **the empty list []** is also a list
- There are other list methods like insert, pop, remove, reverse you may try for yourself

```
list = [1,2,3,4,5,6]  
list.append(7)  
print(list)
```

```
[1, 2, 3, 4, 5, 6, 7]
```

```
lst = []  
lst.append(1)  
lst.append(10)  
lst.append(15)  
print(lst)
```

```
[1, 10, 15]
```

List Methods

Most Important List Methods
to **remember**:

- **append**, which appends an **Element** to the end of a List
- **extend**, who appends a **List** to the end of a List
- **remove**, which removes a given element from a List
- **pop**, which **removes and returns** the element of a List at a given index
- Let's try (and remember, because this is relevant for the midterm exam)

```
lst = [1,6,3]
lst.append(5)
print(lst)
```

```
[1, 6, 3, 5]
```

```
1 lst.extend([7,8,9])
2 print(lst)
```

```
[1, 6, 3, 5, 7, 8, 9]
```

```
lst.pop(0)
```

```
1
```

```
print(lst)
```

```
[6, 3, 5, 7, 8, 9]
```

```
print(lst)
```

```
[6, 3, 5, 7, 8, 9]
```

```
lst.remove(5)
print(lst)
```

```
[6, 3, 7, 8, 9]
```

List Access

- You can access single elements of a list with index in square brackets
- Note that indexing starts 0
- You get the last element by -1, the next last by -2 ...

```
lst = [63, 35, 20, 12, 22, 11, 80]  
lst[0]
```

35

```
lst[1]
```

35

```
lst[-1]
```

80

```
lst[-2]
```

11

Now let's try to define
our first own Function

max



© Can Stock Photo - csp22329014

The max-Function

- Given a list of Integers, you want to find the maximum
- Best you introduce an auxiliary variable **temp**, and store the **best value so far**
- Start with the **first** element
- **Iterate** through the list

```
► def max(list):  
    temp = list[1]  
    for value in list:  
        if temp < value:  
            temp = value  
    return(temp)
```

```
► list = [5,7,1,3,12,10]  
m = max(list)  
print(m)
```

12

max_sort: Sort a List with Descending values

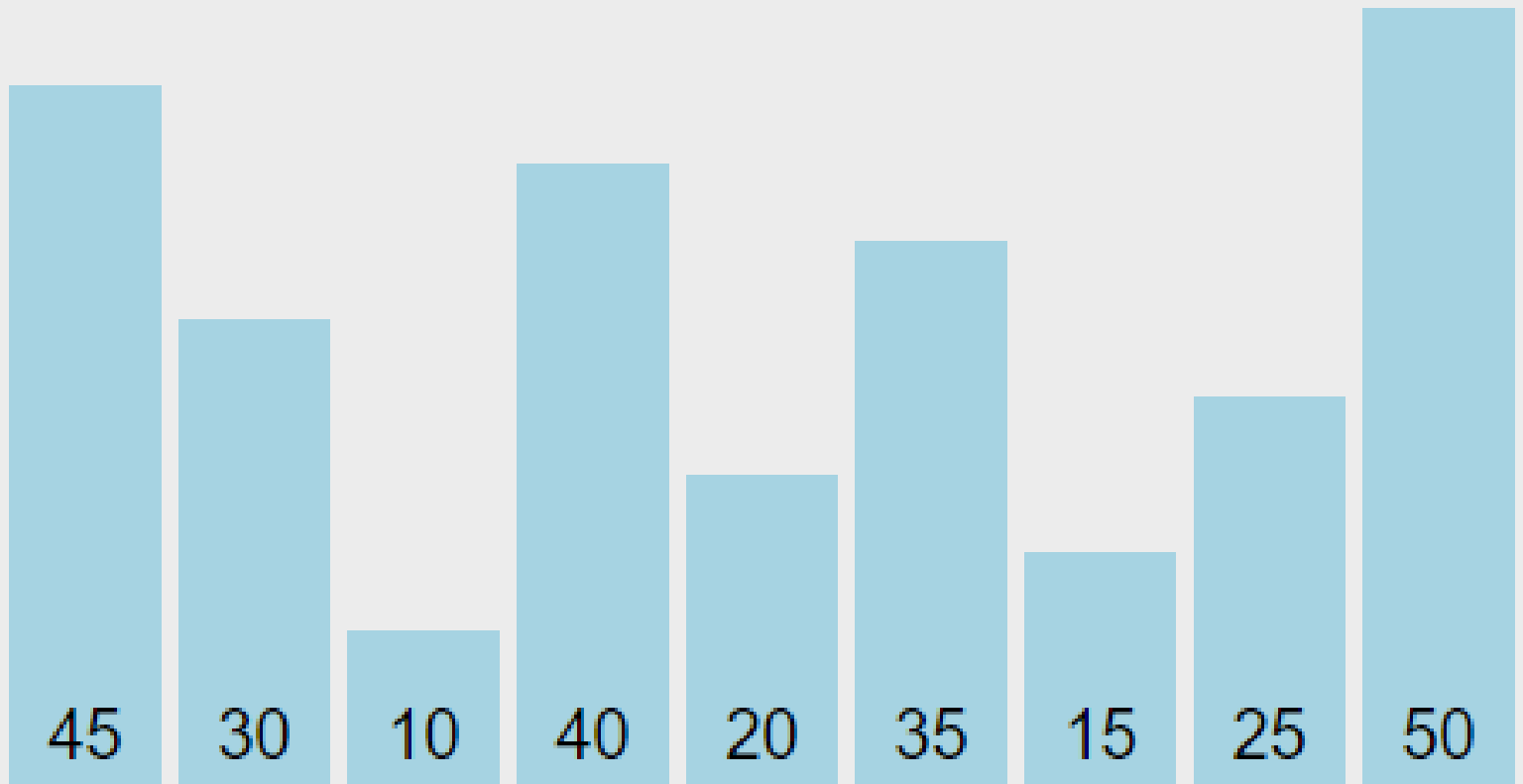
- The idea is easy:
- Find the max-element of a list
- Remove it and make it the first element of a new list
- Iterate with the remaining list
- Apply the list-methods **append** and **remove**
- Use an auxiliary variable **temp**
- And note that the empty list `[]` is a nice list as well
- See how the maximum is removed from the old list and appended to the new list named **temp**

```
def max_sort(lst):  
    temp = []  
    n = len(lst)  
    for i in range(n):  
        m = max_elem(lst)  
        temp.append(m)  
        lst.remove(m)  
    return temp  
  
max_sort([5,3,7,1,9,15,2,11])  
[15, 11, 9, 7, 5, 3, 2, 1]
```


Bubble Sort



Bubble Sort



Idea of Bubblesort

- The Idea is that the so far largest element **bubbles** to the top (the end of the list)
- To that end you just need to **swap** (exchange) **neighbouring** elements if the first is larger then the second
- After the first run, the largest element is on the end of the list. The rest is not yet sorted. That's why we need **two** for loops "bubbling" the larger elements step by step
- Don't be afraid of **nested Loops**
- Note that the real work is done in the **inner** Loop

```
def bubble_sort(lst):  
    n = len(lst)  
    for i in range(n):  
        for j in range(0, n-i-1):  
            if lst[j] > lst[j + 1]:  
                lst[j], lst[j + 1] = lst[j + 1], lst[j]  
    return lst  
  
bubble_sort([63, 35, 20, 12, 22, 11, 80])
```

Classroom Exercise

- Print 6 numbers in the same line with “ +++ ” in between
- Write all the functions mentioned in this lectures and make a nice wrapping for the output such as “6 is the factorial of 3”
- Maybe: “Hi Liu, you asked me for the factorial of 3. I can tell you. It is 6”
- Star Exercise:
- Write a function for adding fractions, using GCD (greatest common divisor) and least common multiple (LCM). Make sure that the fraction are reduced (gcd of numerator and denominator not greater than 1)

Home Exercise

- Make a sheet of Paper, Din A 4, two pages, handwritten, with the most important notes from this lecture. You can use it during midterm exam. That is, beside your laptop, the only thing you can use.
- Give a copy of that sheet to your group counselor before the midterm exam (there is no control of your notes, it is just for us to understand how well you prepared)
- There is no more homework this week, take your time to prepare the exam