The Art and Science of Transportation Research in the AI Era

# A Gentle Introduction to Python

M.Sc. Hiba Karam

**M. Sc. Hiba Karam**
karam@verkehr.tu-darmstadt.de

## Learning Goals

**#1** Be able to understand and write basic Python code

**#2** Get familiar with Python (Jupyter Notebook) interface

**#3** Develop your debugging skills

**#4** Know where to get data for practising

## Lecture Structure

**#1** A mini lecture on what Python is and why it is useful

**#2** Coding in Python 101

**#3** Data source

## The Zen of Python

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

Tim Peters - Python Humor

## #1 A Mini Lecture on Python

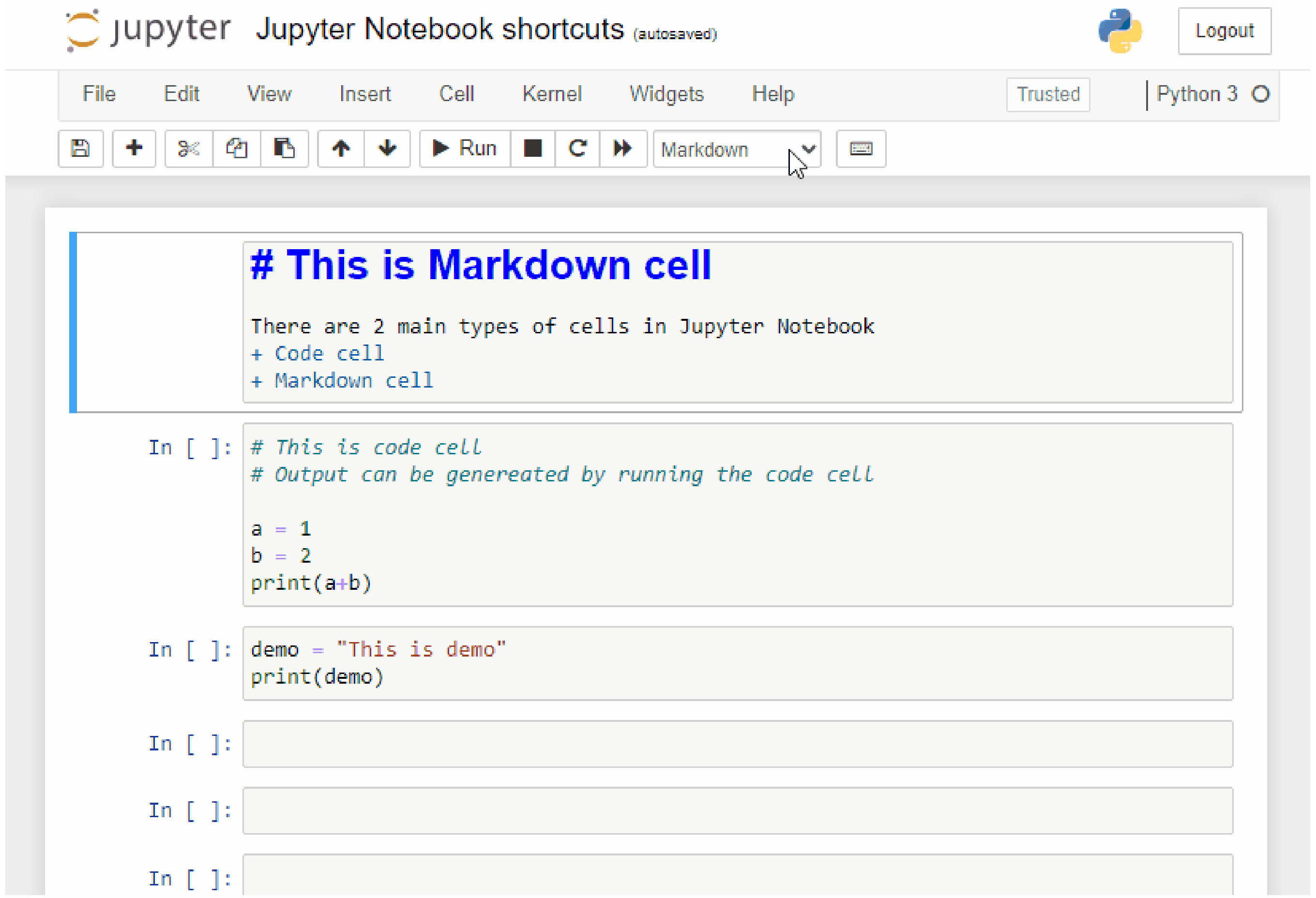
- **Python** is a programming language.
- A programming language is **an artificial language** that humans use to **communicate with computers**.
- We call it a language because **it has its own vocabulary and grammatical rules.**
- It is artificial because humans **explicitly engineered it for specific purposes**. It is different from the natural languages we speak, such as English and German.

# #1 A Mini Lecture on Python

## What is Jupyter Notebook?

**a web-based platform that allows us to create, run, and share code**. It is one type of graphical user interface, meaning that it displays our data, results, graphs, charts together with the code.

JupyterLab, Google Colab, RStudio, and others.



jupyter Jupyter Notebook shortcuts (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Markdown

```
# This is Markdown cell

There are 2 main types of cells in Jupyter Notebook
+ Code cell
+ Markdown cell
```

```
In [ ]: # This is code cell
        # Output can be genereated by running the code cell

        a = 1
        b = 2
        print(a+b)

In [ ]: demo = "This is demo"
        print(demo)

In [ ]:

In [ ]:

In [ ]:
```

Jupyter Notebook Interface

**Cell Type:**

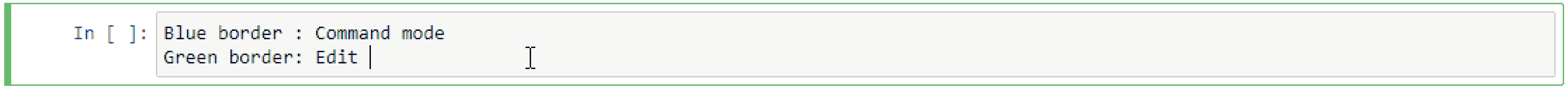
**Code:** contain Python code.

**Markdown:** is a way to format text. It is used to include explanatory text in your notebooks.

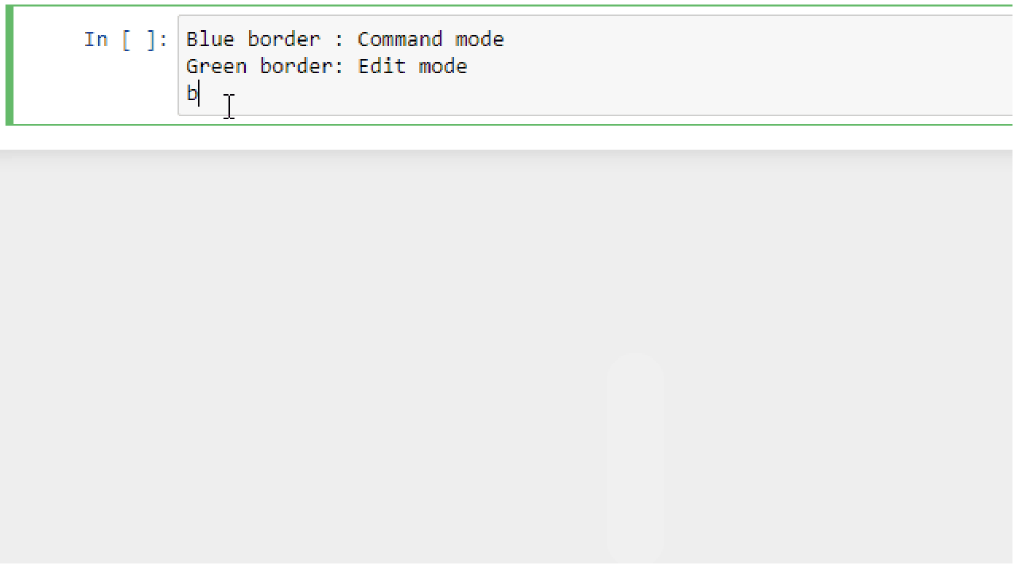
# #1 A Mini Lecture on Python

The Jupyter Notebook has two different keyboard input modes.

**1. Edit mode** allows you to type code or text into a cell and is indicated by a green cell border.



```
In [ ]: Blue border : Command mode
        Green border: Edit
```



```
In [ ]: Blue border : Command mode
        Green border: Edit mode
        b
```

**2. Command mode** Allow you to perform actions like selecting multiple cells, copying and pasting cells, adding or deleting cells, etc. To perform the above actions, you need to first enter into command mode by **pressing ESC** or **clicking the left border**

## #1 A Mini Lecture on Python

**Keyboard shortcuts:**

Shift + Enter run the current cell and select below

While in command mode (press Esc)

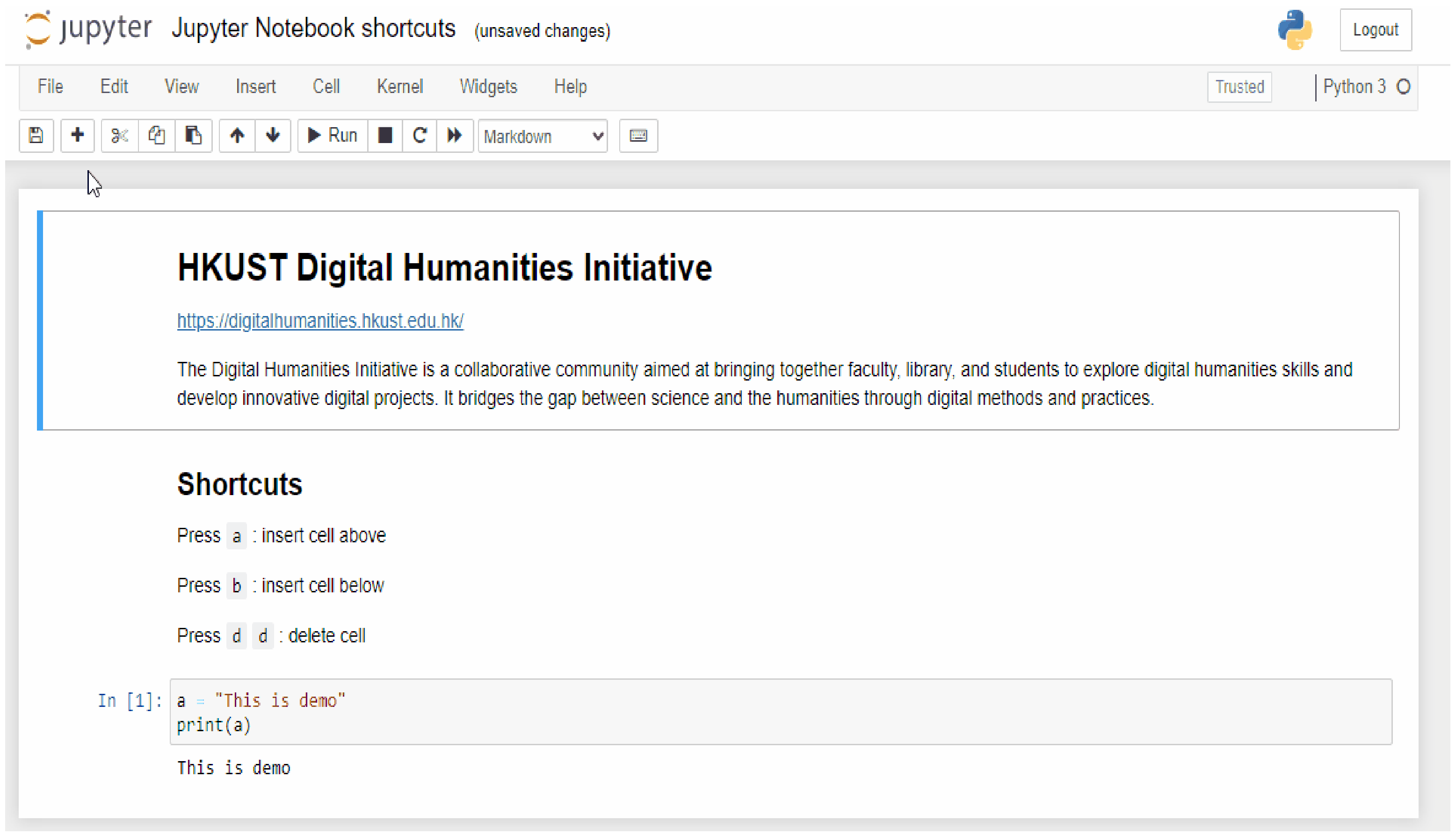
H show all shortcuts

A insert cell above

B insert cell below

X cut the current cell

D D delete the selected cell(s)


jupyter Jupyter Notebook shortcuts (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Markdown

HKUST Digital Humanities Initiative

https://digitalhumanities.hkust.edu.hk/

The Digital Humanities Initiative is a collaborative community aimed at bringing together faculty, library, and students to explore digital humanities skills and develop innovative digital projects. It bridges the gap between science and the humanities through digital methods and practices.

Shortcuts

Press a : insert cell above

Press b : insert cell below

Press d d : delete cell

```
In [1]: a = "This is demo"
        print(a)

This is demo
```

## #1 A Mini Lecture on Python

As you can see in the Zen of Python, its philosophy of design emphasizes on simplicity and readability.

As a result, it is relatively easy to understand and learn.

Automation

Data Science

Machine Learning

Application Development

**Python Usage**

## #2 Coding in Python 101

TECHNISCHE UNIVERSITÄT DARMSTADT

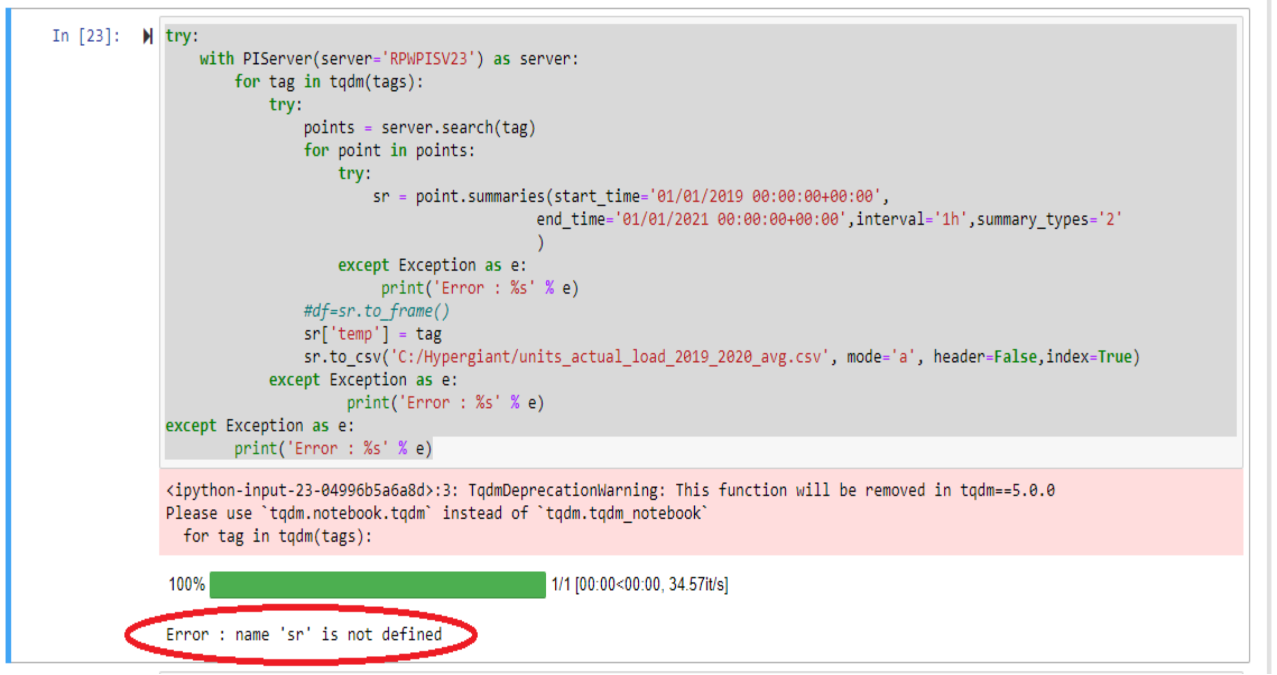
Institut für Verkehrsplanung und Verkehrstechnik TU Darmstadt

There is **no need to memorize commands,** but we need to know **where and how to find help**.

There will **always be bugs** (i.e., coding errors). If things don't work, don't panic. That's normal.

**General ways of debugging:**

- Google the error message (ChatGPT)
- Search / ask on Stack Overflow
- Read the documentation
- Divide your code and get results for each part to locate the problem
- Use the help() function in Jupyter Notebook



```
In [23]:  try:
              with PIServer(server='RPWPISV23') as server:
                  for tag in tqdm(tags):
                      try:
                          points = server.search(tag)
                          for point in points:
                              try:
                                  sr = point.summaries(start_time='01/01/2019 00:00:00+00:00',
                                                       end_time='01/01/2021 00:00:00+00:00',interval='1h',summary_types='2'
                                                       )
                              except Exception as e:
                                    print('Error : %s' % e)
                          #df=sr.to_frame()
                          sr['temp'] = tag
                          sr.to_csv('C:/Hypergiant/units_actual_load_2019_2020_avg.csv', mode='a', header=False,index=True)
                      except Exception as e:
                            print('Error : %s' % e)
          except Exception as e:
                  print('Error : %s' % e)
```

```
<ipython-input-23-04996b5a6a8d>:3: TqdmDeprecationWarning: This function will be removed in tqdm==5.0.0
Please use `tqdm.notebook.tqdm` instead of `tqdm.tqdm_notebook`
  for tag in tqdm(tags):

100% 1/1 [00:00<00:00, 34.57it/s]

Error : name 'sr' is not defined
```

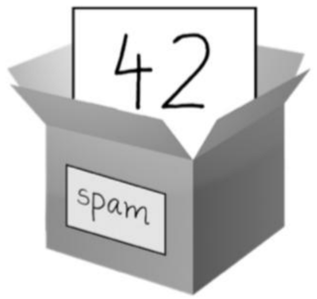
# #2.1 Variables

Symbolic name or **identifier** that represents **a value stored** in Python's memory.

Think of it as a **container or a label that holds data**, **which can be used and manipulated throughout Python**. The value stored in a variable can change, hence the name "variable." We can assign a value to a variable using the equals sign =.

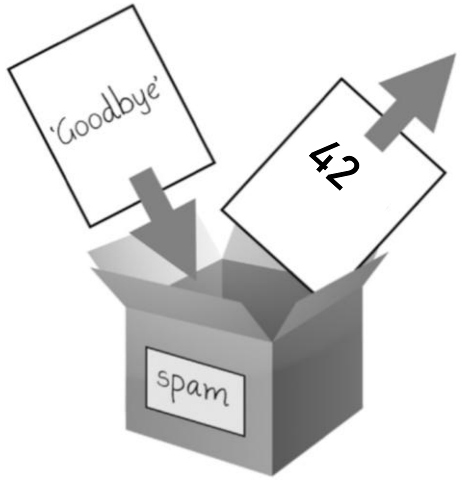
**spam = 42**

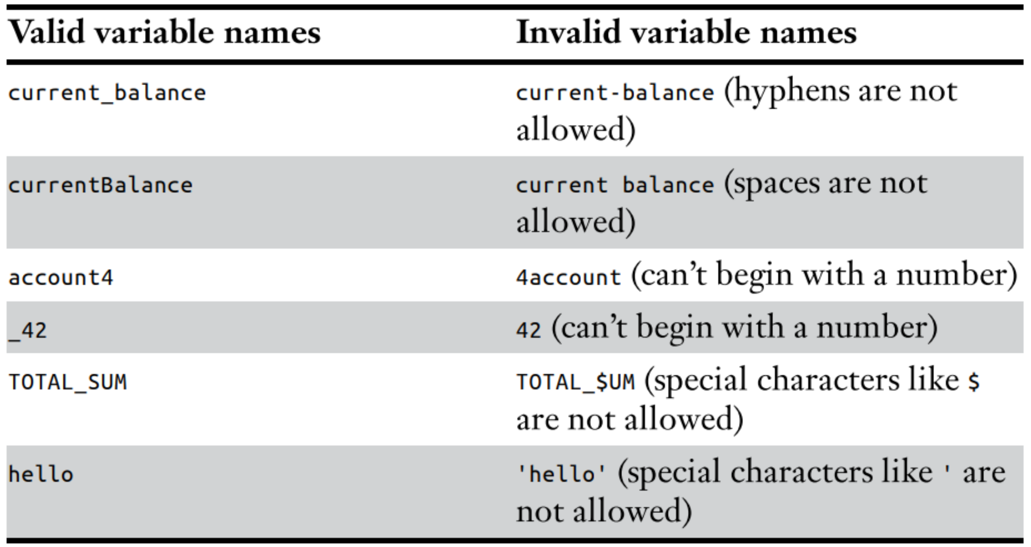
is like telling the program, "The variable spam now has the integer value 42 in it."



**spam = 'Goodbye'**

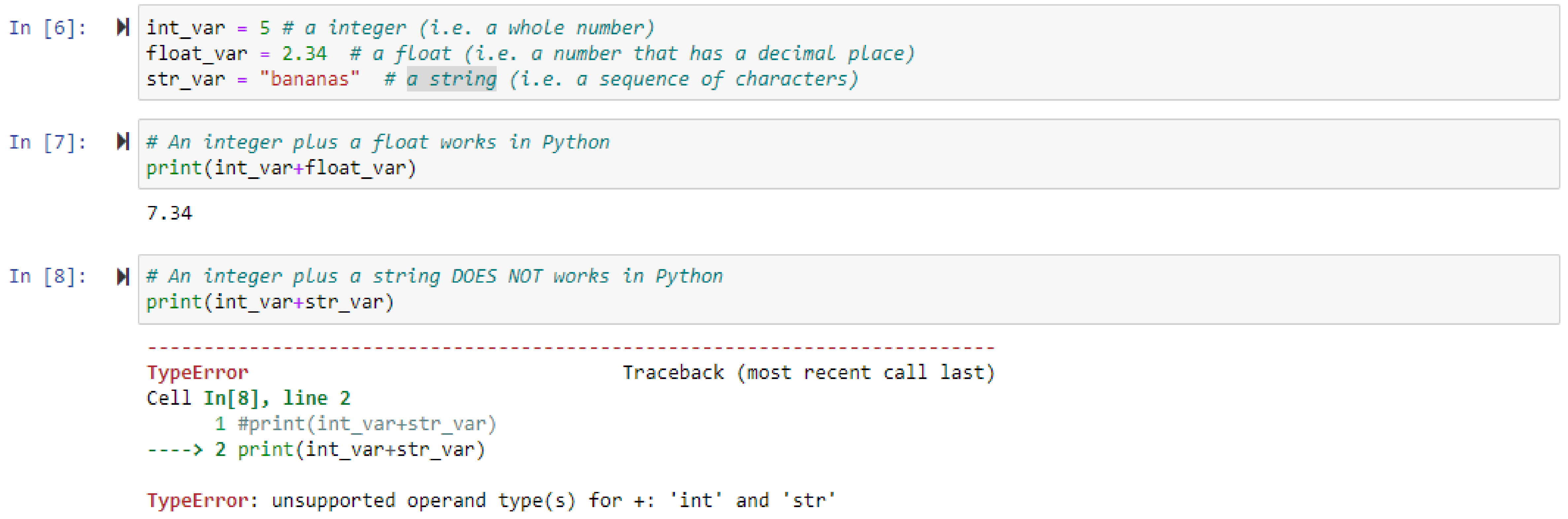
When a new value is assigned to a variable, the old one is forgotten.





| Valid variable names | Invalid variable names |
|---|---|
| `current_balance` | `current-balance` (hyphens are not allowed) |
| `currentBalance` | `current balance` (spaces are not allowed) |
| `account4` | `4account` (can't begin with a number) |
| `_42` | `42` (can't begin with a number) |
| `TOTAL_SUM` | `TOTAL_$UM` (special characters like `$` are not allowed) |
| `hello` | `'hello'` (special characters like `'` are not allowed) |

## #2.1 Variables

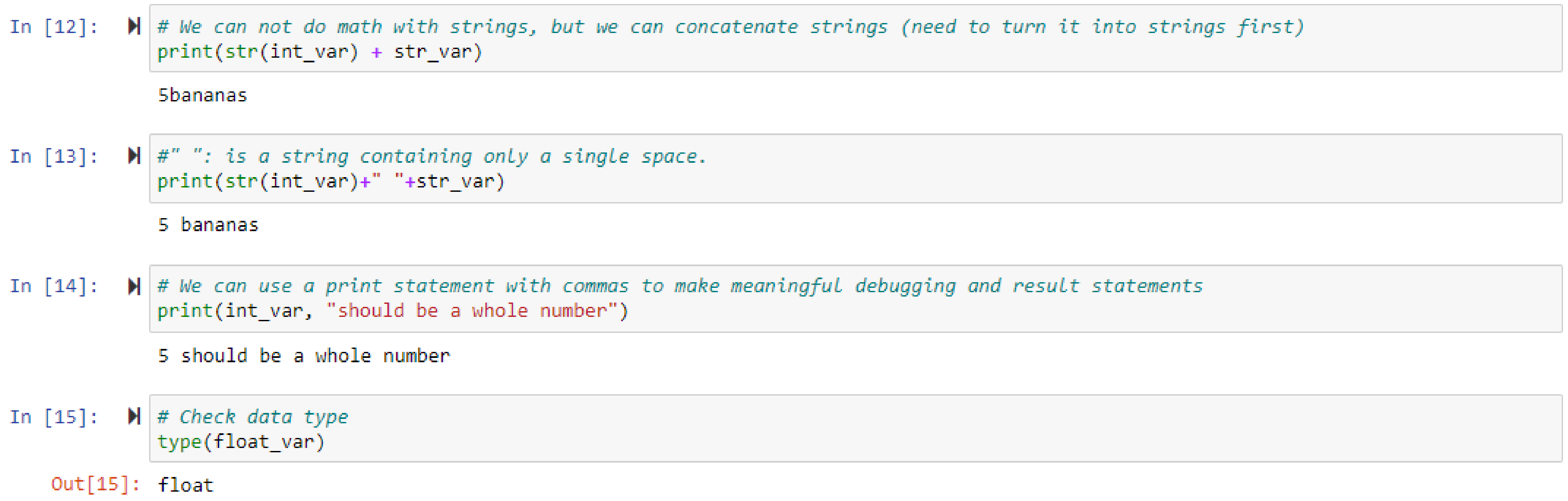


```
In [6]:  int_var = 5 # a integer (i.e. a whole number)
         float_var = 2.34  # a float (i.e. a number that has a decimal place)
         str_var = "bananas"  # a string (i.e. a sequence of characters)
```

```
In [7]:  # An integer plus a float works in Python
         print(int_var+float_var)

         7.34
```

```
In [8]:  # An integer plus a string DOES NOT works in Python
         print(int_var+str_var)

         ---------------------------------------------------------------------------
         TypeError                                 Traceback (most recent call last)
         Cell In[8], line 2
               1 #print(int_var+str_var)
         ----> 2 print(int_var+str_var)

         TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

# #2.1 Variables



```
In [12]:  # We can not do math with strings, but we can concatenate strings (need to turn it into strings first)
          print(str(int_var) + str_var)

          5bananas

In [13]:  #" ": is a string containing only a single space.
          print(str(int_var)+" "+str_var)

          5 bananas

In [14]:  # We can use a print statement with commas to make meaningful debugging and result statements
          print(int_var, "should be a whole number")

          5 should be a whole number

In [15]:  # Check data type
          type(float_var)

Out[15]:  float
```

## #2.1 Variables

1. Print the statement Hello, World!

2. Assign a variable name called greetings to the above statement and then print the variable.

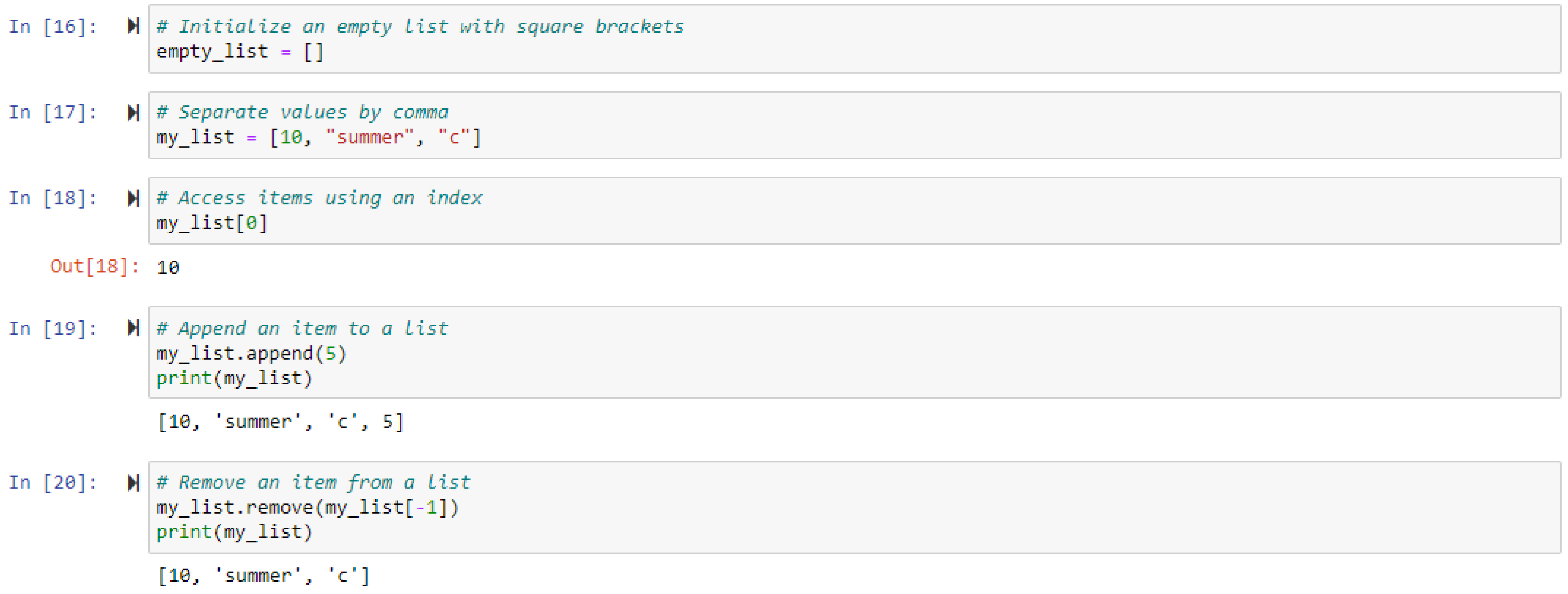
## #2.1 Variables

```
print("Hello, World!")
```

```
greetings = "Hello, World!"
print(greetings)
```

## #2.2 Lists

- A list is the most common **data structure** in Python. It **stores a series of items in a particular order**.
- The first index starts with zero, the second index is one, and so forth.
- Items in a list need not be of the same type. More on list methods can be found here.



```
In [16]:  # Initialize an empty list with square brackets
          empty_list = []

In [17]:  # Separate values by comma
          my_list = [10, "summer", "c"]

In [18]:  # Access items using an index
          my_list[0]

Out[18]:  10

In [19]:  # Append an item to a list
          my_list.append(5)
          print(my_list)

          [10, 'summer', 'c', 5]

In [20]:  # Remove an item from a list
          my_list.remove(my_list[-1])
          print(my_list)

          [10, 'summer', 'c']
```

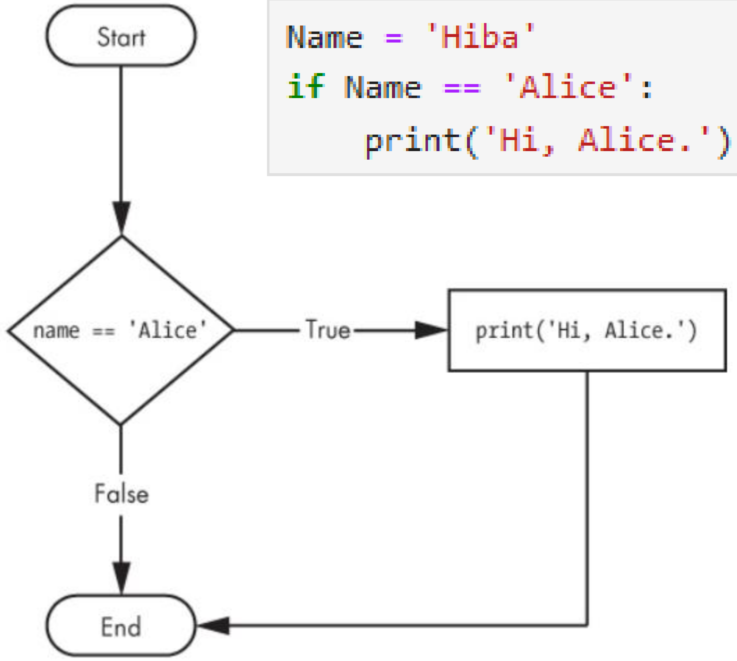
❓ **Question**: Guess what does -1 mean in the above code my_list[-1]

# #2.3 If Statements

If statement allow us to **examine logical conditions** and act based on different conditions.

If this condition is true, execute the code in the clause.

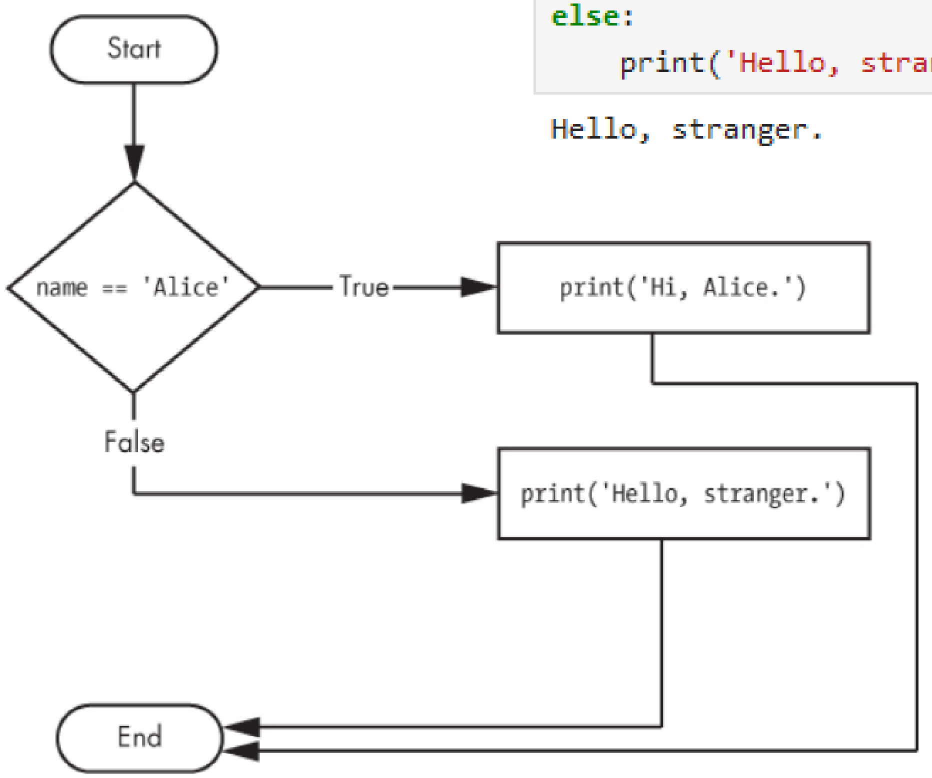
```python
Name = 'Hiba'
if Name == 'Alice':
    print('Hi, Alice.')
```



Flowchart of If statement

```python
Name = 'Hiba'
if Name == 'Alice':
    print('Hi, Alice.')
else:
    print('Hello, stranger.')
```

```
Hello, stranger.
```



Flowchart of If else statement

TECHNISCHE UNIVERSITÄT DARMSTADT

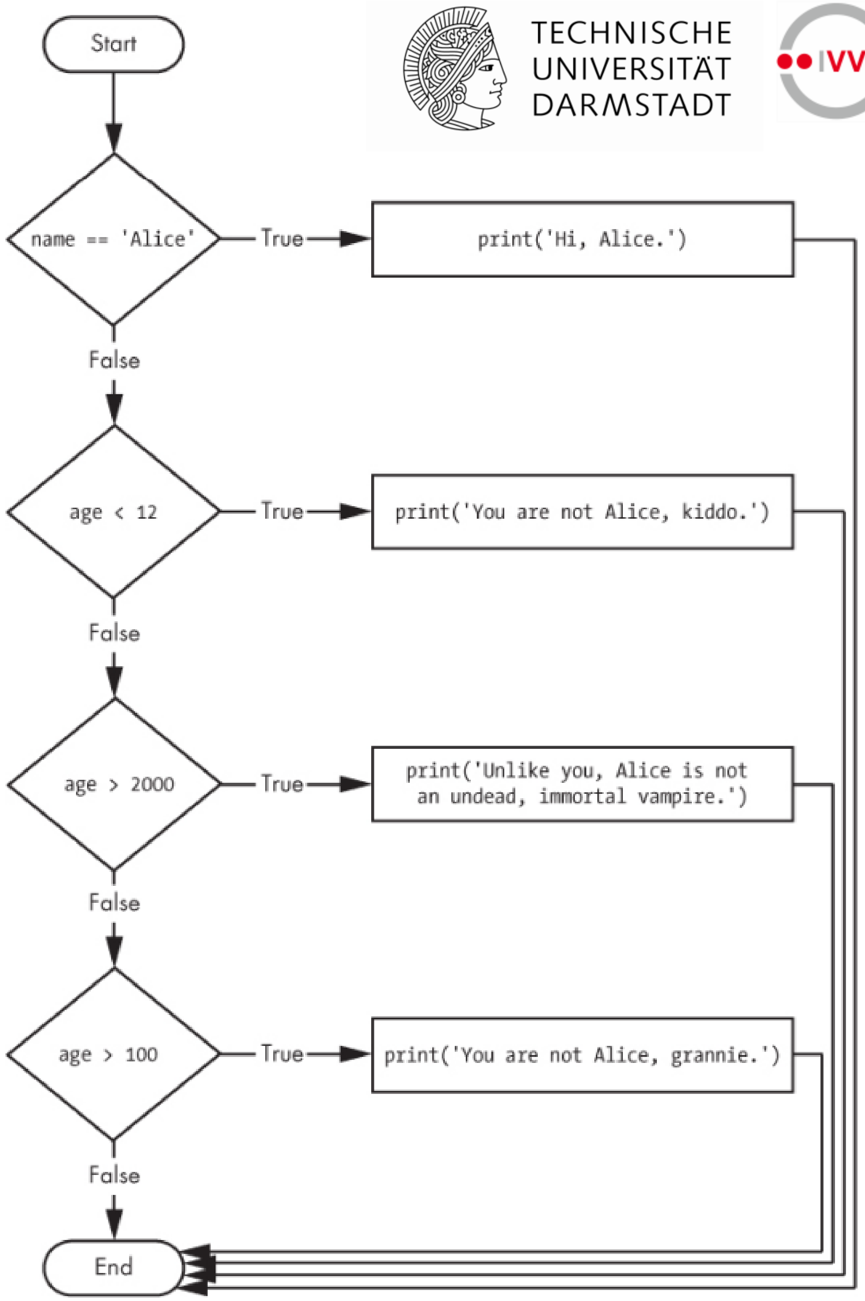
Institut für Verkehrsplanung und Verkehrstechnik TU Darmstadt

# #2.3 If Statements

While only one of the if or else clauses will execute, you may have a case where you want **one of many possible clauses to execute**.
Here we use elif.

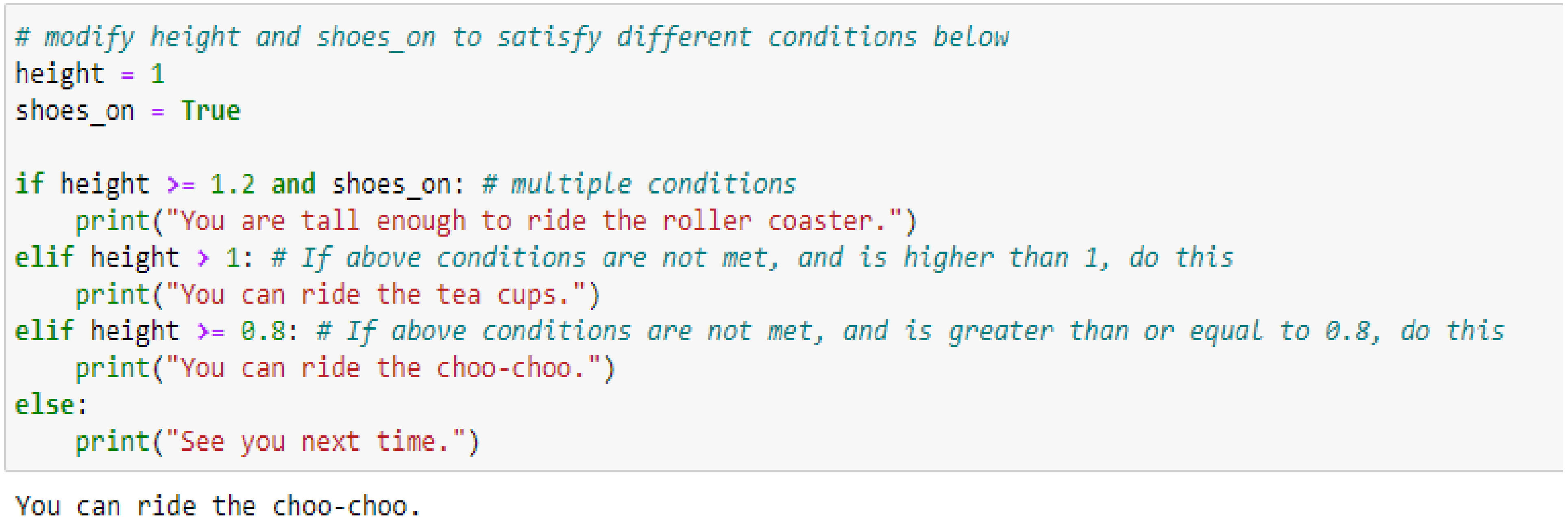
```
name = 'Carol'
age = 3000

if name == 'Alice':
    print('Hi, Alice.')
elif age < 12:
    print('You are not Alice, kiddo.')
elif age > 2000:
    print('Unlike you, Alice is not an undead, immortal vampire.')
elif age > 100:
    print('You are not Alice, grannie.')
```

```
Unlike you, Alice is not an undead, immortal vampire.
```


Start
name == 'Alice'
True
print('Hi, Alice.')
False
age < 12
True
print('You are not Alice, kiddo.')
False
age > 2000
True
print('Unlike you, Alice is not an undead, immortal vampire.')
False
age > 100
True
print('You are not Alice, grannie.')
False
End

Flowchart for multiple elif statements

## #2.3 If Statements



```
# modify height and shoes_on to satisfy different conditions below
height = 1
shoes_on = True

if height >= 1.2 and shoes_on: # multiple conditions
    print("You are tall enough to ride the roller coaster.")
elif height > 1: # If above conditions are not met, and is higher than 1, do this
    print("You can ride the tea cups.")
elif height >= 0.8: # If above conditions are not met, and is greater than or equal to 0.8, do this
    print("You can ride the choo-choo.")
else:
    print("See you next time.")
```

```
You can ride the choo-choo.
```

## #2.3 If Statements

2. Write a short code that print "You are an adult." if the age is 18 or older, and "You are not an adult." if the age is under 18.

## #2.3 If Statements

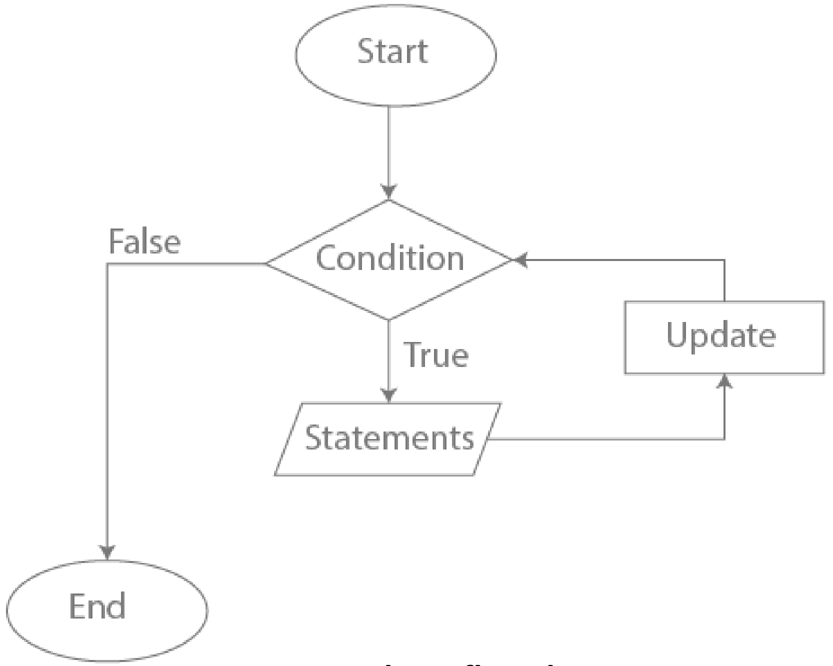
```python
if age >= 18:
    print("You are an adult.")
else:
    print("You are not an adult.")
```

# #2.4 Loops

- There are two types of loops in Python: for loop and while loop.
- A loop is an instruction that **executes a statement until a specific condition is reached**. The number of times the loop repeats itself is known as iteration.
- A **for loop** iterates over a given sequence. It **repeats** a block of code the **number of times** described in the "for" statement.

```python
traffic_lights = ["Red", "Green", "Yellow"]
for light in traffic_lights:
    print("The light is now " + light)
```

```
The light is now Red
The light is now Green
The light is now Yellow
```



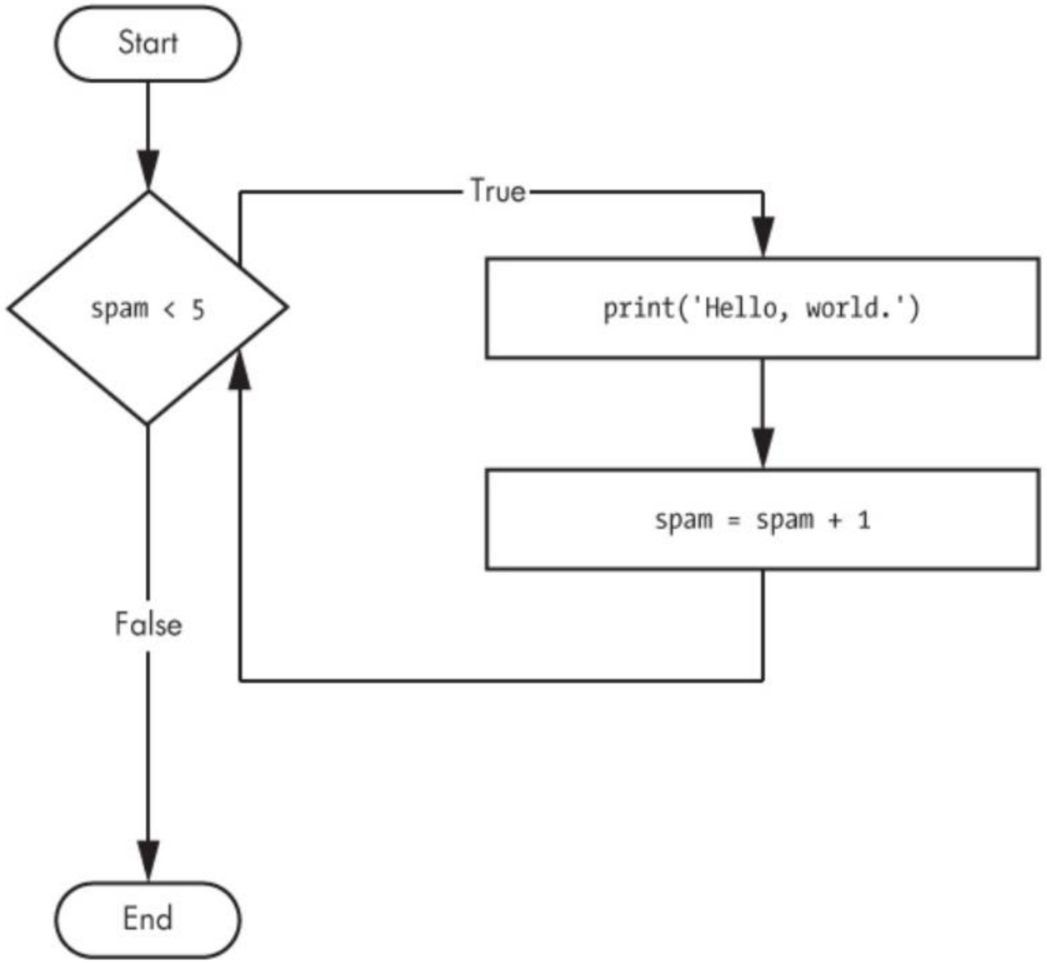
**For loop flowchart**

## #2.4 Loops

- A **while loop** repeats a block of code as long as a certain condition is met.
- The while loop keeps looping **while its condition is True** (which is the reason for its name).

```python
spam = 0
while spam < 5:
    print('Hello, world.')
    spam = spam + 1
```

```
Hello, world.
Hello, world.
Hello, world.
Hello, world.
Hello, world.
```

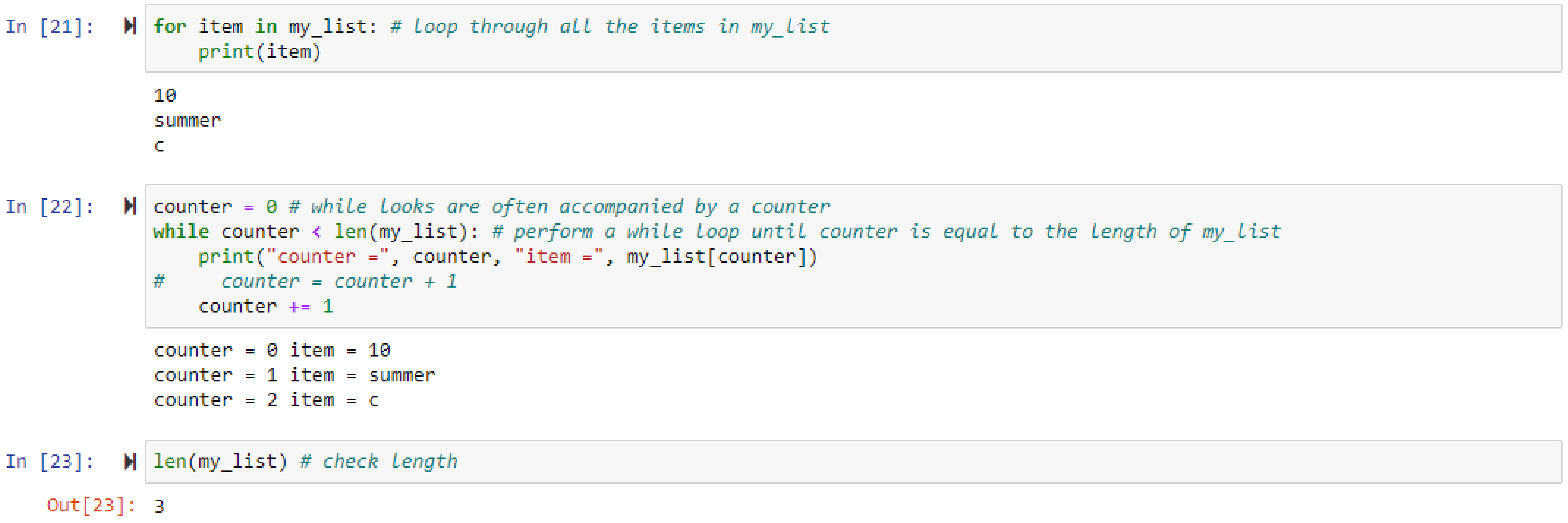


**While loop flowchart**

TECHNISCHE UNIVERSITÄT DARMSTADT

Institut für Verkehrsplanung und Verkehrstechnik TU Darmstadt

## #2.4 Loops



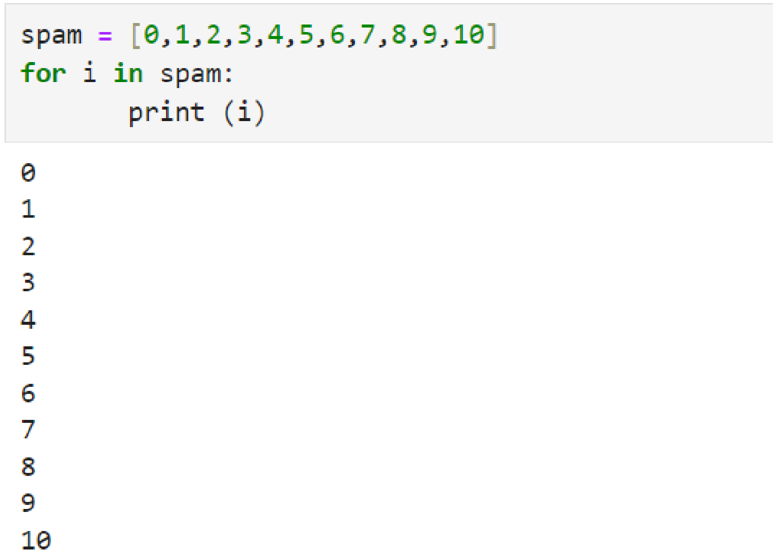
```
In [21]:  for item in my_list: # loop through all the items in my_list
              print(item)

          10
          summer
          c

In [22]:  counter = 0 # while looks are often accompanied by a counter
          while counter < len(my_list): # perform a while loop until counter is equal to the length of my_list
              print("counter =", counter, "item =", my_list[counter])
          #     counter = counter + 1
              counter += 1

          counter = 0 item = 10
          counter = 1 item = summer
          counter = 2 item = c

In [23]:  len(my_list) # check length

Out[23]:  3
```

**Note:** the counter ensures that the loop will eventually terminate once the counter exceeds the length of my_list.

## #2.4 Loops

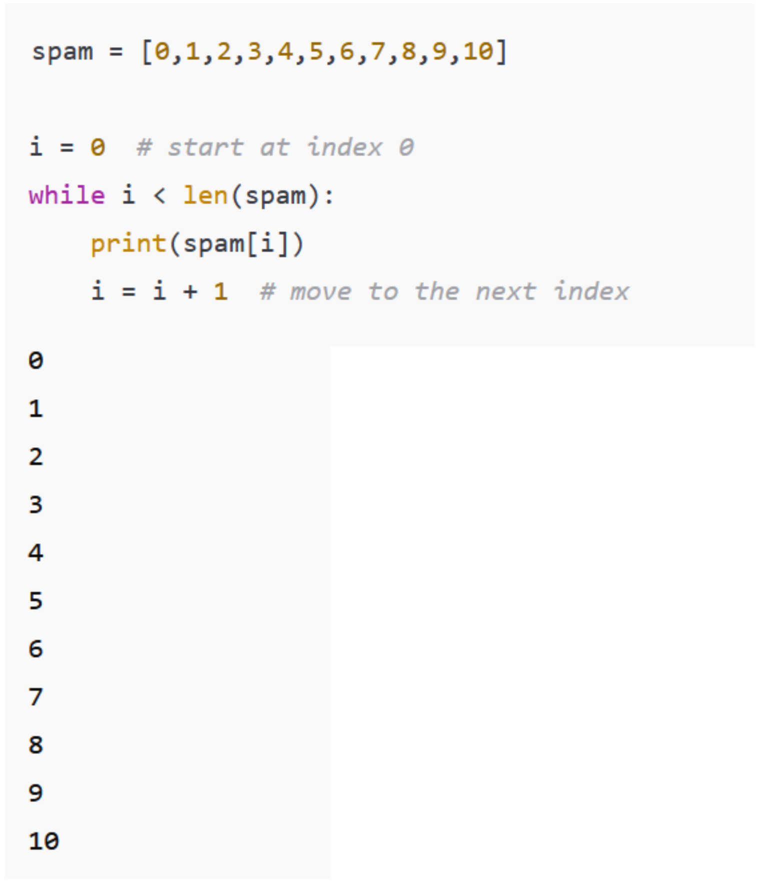
3. Write a short code that makes a list of numbers from **0 to 10** and then prints them using a for loop and while loop.

## #2.4 Loops



```
spam = [0,1,2,3,4,5,6,7,8,9,10]
for i in spam:
        print (i)
```

```
0
1
2
3
4
5
6
7
8
9
10
```



```
spam = [0,1,2,3,4,5,6,7,8,9,10]

i = 0  # start at index 0
while i < len(spam):
    print(spam[i])
    i = i + 1  # move to the next index
```

```
0
1
2
3
4
5
6
7
8
9
10
```

## #2.5 Functions

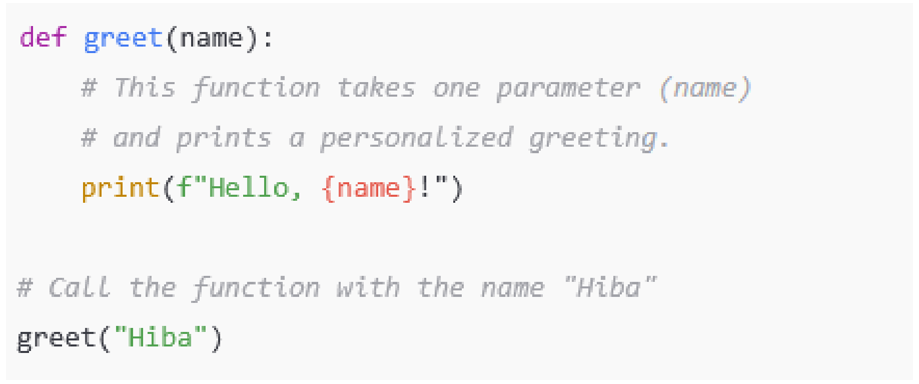
A function is a **reusable block of code** that performs a specific task. Functions allow you to organize your code and **avoid repetition**.

A variable defined inside a function is not universal.

```python
def function_name(parameters):
    """Optional docstring to describe the function."""
    # Code block
    return some_value  # Optional
```

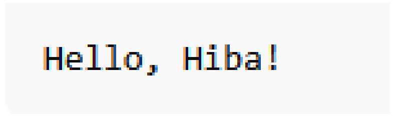
- **def:** This keyword is used to define a function.
- **function_name:** This is the name you give to the function. It should be descriptive of what the function does.
- **Parameters** (also called Inputs or arguments) are what we pass to a function.
- **a colon** is used to indicate the start of the function body.
- **Docstring (Optional)**: A string literal used to describe the function's purpose, which is often used for documentation.
- **Code Block:** This is the body of the function that performs the task.
- **return:** This keyword is used to return a value from the function. It's optional; if you don't use return, the function will return None by default.

## #2.5 Functions

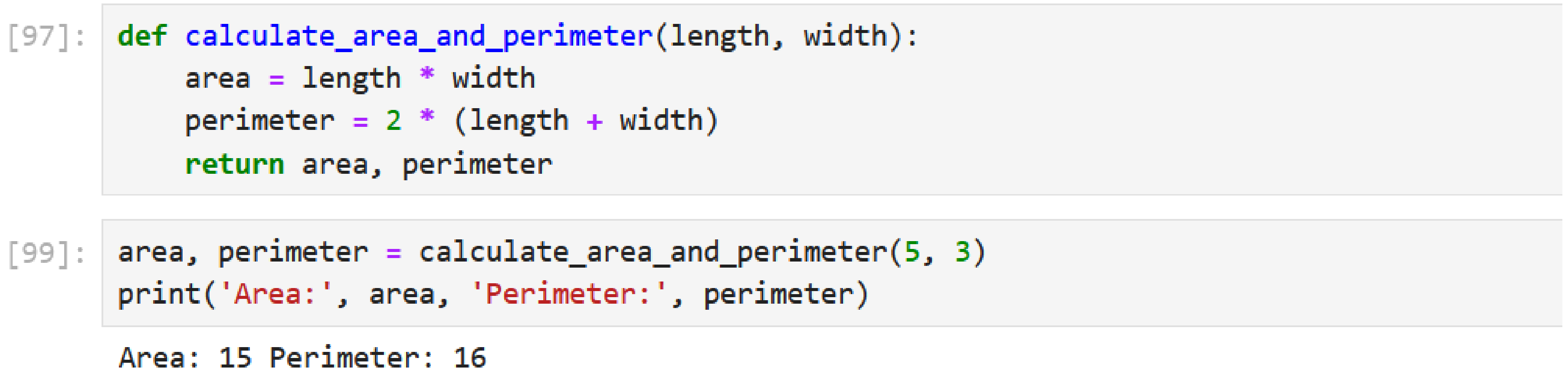


```python
def greet(name):
    # This function takes one parameter (name)
    # and prints a personalized greeting.
    print(f"Hello, {name}!")

# Call the function with the name "Hiba"
greet("Hiba")
```

### 💻 Output



```
Hello, Hiba!
```



```python
[97]: def calculate_area_and_perimeter(length, width):
          area = length * width
          perimeter = 2 * (length + width)
          return area, perimeter
```

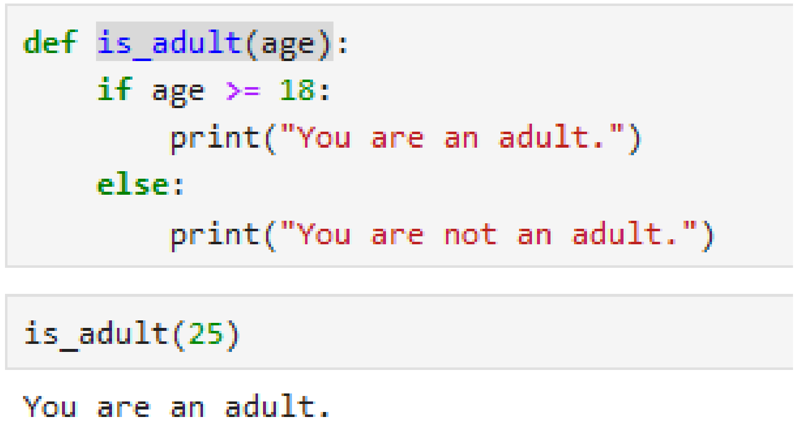
```python
[99]: area, perimeter = calculate_area_and_perimeter(5, 3)
      print('Area:', area, 'Perimeter:', perimeter)
```

```
Area: 15 Perimeter: 16
```

## #2.5 Functions

4. Make the code in Q2 a function (2. Write a short code that print "You are an adult." if the age is 18 or older, and "You are not an adult." if the age is under 18.) and try calling your function using different age numbers.

## #2.5 Functions

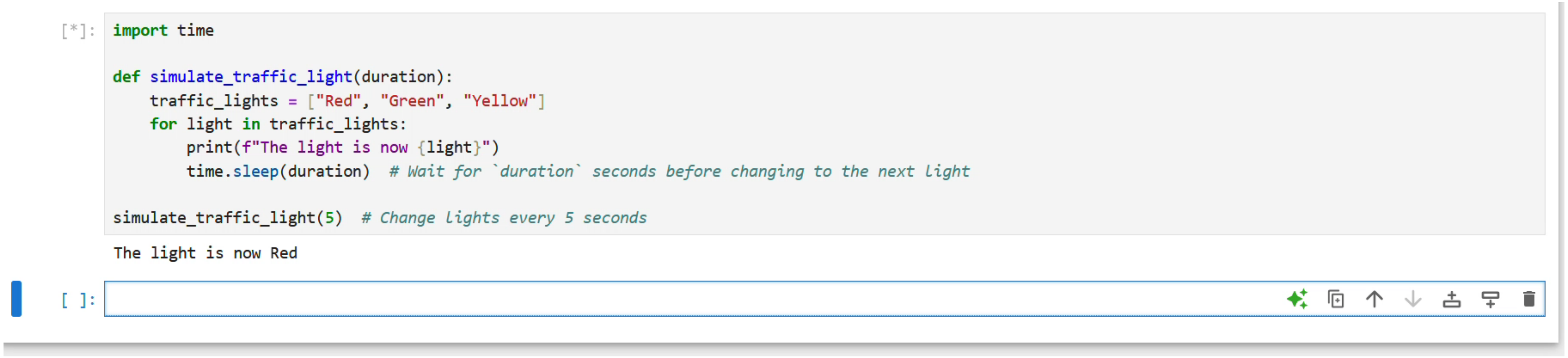


```
def is_adult(age):
    if age >= 18:
        print("You are an adult.")
    else:
        print("You are not an adult.")
```

```
is_adult(25)
```

```
You are an adult.
```

## #2.6 An Example



```
[*]: import time

     def simulate_traffic_light(duration):
         traffic_lights = ["Red", "Green", "Yellow"]
         for light in traffic_lights:
             print(f"The light is now {light}")
             time.sleep(duration)  # Wait for `duration` seconds before changing to the next light

     simulate_traffic_light(5)  # Change lights every 5 seconds

     The light is now Red

[ ]:
```

Basic code to simulate traffic control systems
Model simple traffic scenarios

## #2.7 Ready to Use Python Libraries

A cool thing about Python is that there are **tons of ready-to-use libraries**. We don't need to write too much code from scratch to solve a problem. All we need to do is to find a good library and learn how to use it.

| | | | |
|---|---|---|---|
| **Pandas** | Data analysis and manipulation | **NumPy** | Mathematical functions |
| **Matplotlib** | Data visualisations | **SeaBorn** | Data visualisations |
| **Tensorflow** | Machine Learning | **Keras** | Deep Learning |
| **SciPy** | Scientific computing | **PyTorch** | Machine Learning |
| **Scrapy** | Web crawling | **SQLModel** | Interact with SQL databases |

## #3 Data Source

**Primary vs. Secondary Data**:

- **Primary Data**: Data collected first-hand for specific research purposes (e.g., surveys, traffic counts, GPS data).
- **Secondary Data**: Existing data collected for other purposes but useful for your research (e.g., government databases, research papers, open data portals).

**Secondary Data Sources**:

- **Government Agencies**: e.g., Department of Transportation, Bureau of Transportation Statistics
- **Public and Private Databases**: e.g., Google Maps APIs, Uber Movement, World Bank, Data.world, Kaggle
- **Research Institutions**: e.g., University transportation research centers, published research papers
- **Open Data Portals**: e.g., Data.gov, GovData, European Data Portal, local city data portals (darmstadt open data)
- **Crowdsourced Data**: e.g., OpenStreetMap

- **THANK YOU**
- **DANKE**