Mathematics and Computer Science for Modeling
Unit 1: Introduction to Programming in Python

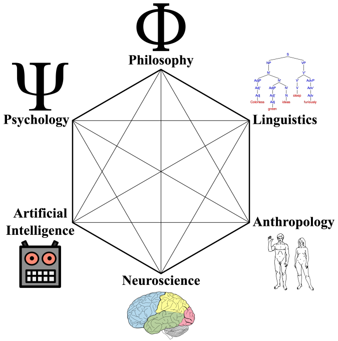
Daniel Sabinasz
based on materials by Jan Tekülve and Daniel Sabinasz

Institut für Neuroinformatik, Ruhr-Universität Bochum

26.09.2022

Why this course?

- Anyone with a Bachelor's degree in any of the cognitive sciences can start this Master's degree
- You will then be exposed to lectures from all of the cognitive science disciplines



Why this course?

- Not all of you will have the same level of background knowledge for all of the lectures
- The preparatory courses are here to help you bridge that gap
- Goal here: Bring you on a similar level regarding mathematics and computer science skills
- … which will hopefully make it easier for you to get through the Master programme
- The course is not mandatory, but highly recommended

Course concept

- The course is split into lecture parts and exercise parts

Exam

- At the end of the course, there will be a written exam (07.10. at 3 pm)
- The exam is graded, but this is only for your feedback and won't enter into your average grade

About Me

- My name is Daniel Sabinasz
- B.Sc. computer science and M.Sc. cognitive science
- PhD candidate at the Institute for Neural Computation
- Working on mathematical modeling of the neural processes that underlie language understanding
- Email me with any questions you might have: daniel.sabinasz@ini.rub.de

Course Structure

Unit	Title	Topics
1	Intro to Programming in Python	*Variables, if Statements, Loops, Functions, Lists*
-	Full-Time Programming Session	*Deepen Programming Skills*
2	Functions in Math	*Function Types and Properties, Plotting Functions, Lists*
3	Linear Algebra	*Vectors, Trigonometry, Matrices*
4	Calculus	*Derivative Definition, Calculating Derivatives*

Course Structure

Unit	Title	Topics
5	Integration	*Geometrical Definition, Calculating Integrals, Numerical Integration*
6	Differential Equations	*Properties of Differential Equations, Euler Approximation, Braitenberg Vehicle*
-	Programming Session & Recap	*Repetition, Questions, Test Topics*
-	07.10.22: Test	

Lecture Slides/Material

Use the following URL to access the lecture slides:

https://www.ini.rub.de/teaching/courses/preparatory_course_mathematics_and_computer_science_for_modeling_summer_term_2022

1. Motivation

2. **Programming**
    - Python
    - First Steps
    - Variables
    - if Statements
    - Loops

3. Functions

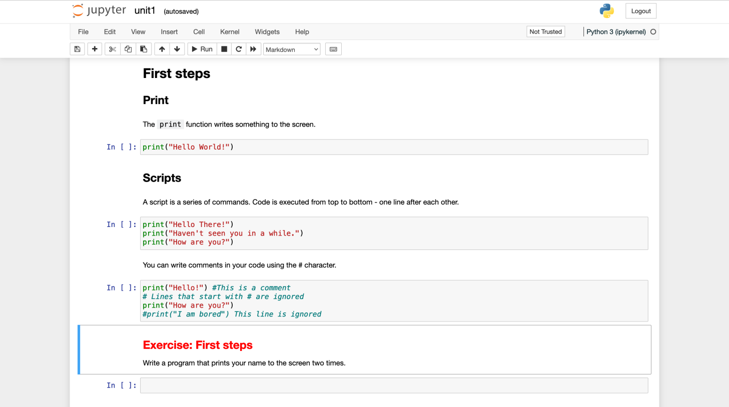
4. Lists
    - for Loops

Getting Started

- Install Anaconda: https://www.anaconda.com/distribution/
- Download the document “Jupyter notebook” for Unit 1 (filename “unit1.ipynb”) from the course website
- Start the program “Anaconda-Navigator”. Find the application “Jupyter Notebook” and click on “launch”.
- (Alternative: Start the program “Anaconda Prompt”. Wait for a prompt to appear and then enter “jupyter notebook” into that prompt)
- Navigate to the directory where you saved the “unit1.ipynb” file and click on that file

Getting Started

- You are now presented with a so-called Jupyter Notebook, a document that allows you to execute existing Python code and write your own Python code while being guided by narrative text


jupyter unit1 (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3 (ipykernel)
Run
Markdown

First steps

Print

The `print` function writes something to the screen.

```
In [ ]: print("Hello World!")
```

Scripts

A script is a series of commands. Code is executed from top to bottom - one line after each other.

```
In [ ]: print("Hello There!")
        print("Haven't seen you in a while.")
        print("How are you?")
```

You can write comments in your code using the # character.

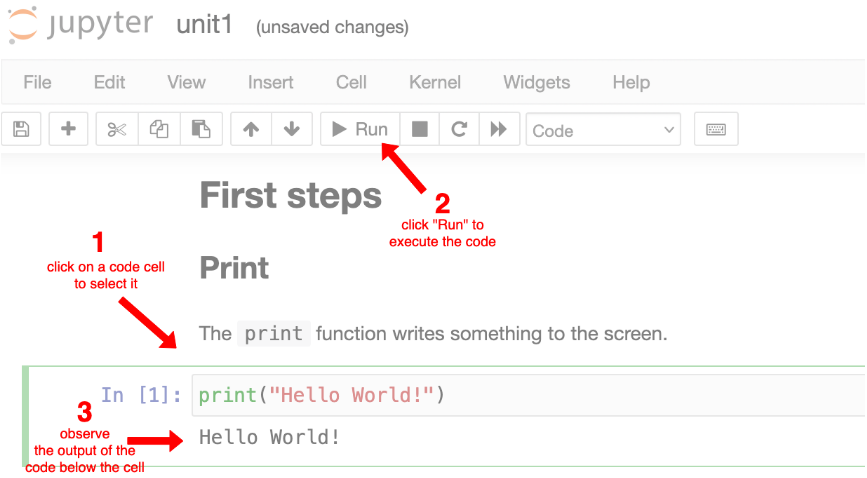
```
In [ ]: print("Hello!") #This is a comment
        # Lines that start with # are ignored
        print("How are you?")
        #print("I am bored") This line is ignored
```

Exercise: First steps

Write a program that prints your name to the screen two times.

```
In [ ]:
```

Getting Started



First steps

Print

The `print` function writes something to the screen.

```
In [1]: print("Hello World!")
```

```
Hello World!
```

1 click on a code cell to select it

2 click "Run" to execute the code

3 observe the output of the code below the cell

Print

- The print() function writes a text string to the output of the program:

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

Output:

```
Hello World!
```

Scripts

- A script is a series of commands
- Code is executed from top to bottom - one line after each other

```
print("Hello There!")
print("Haven't seen you in a while.")
print("How are you?")
```

Output:

```
Hello There!
Haven't seen you in a while.
How are you?
```

Scripts

- You can write comments in your code using the # character

```
print("Hello!") #This is a comment
# Lines that start with # are ignored
print("How are you?")
#print("I am bored") This line is ignored
```

Output:

```
Hello!
How are you?
```

Exercise: First steps

1. Write a program that prints your name to the screen two times.
2. Add comments to the program.

Variables

- Variables are containers for storing data values.
- A value can be assigned to a variable using the ‘=‘ sign.

```
greeting = "Hello, Hello!"
print(greeting)
```

Output:

```
Hello, Hello!
```

Variables

- A variable name may consist of letters, numbers and underscores.

```
var1 = "Hello"
long_variable_name5 = "Hi"
```

Variables

- A variable name may consist of letters, numbers and underscores.

```
var1 = "Hello"
long_variable_name5 = "Hi"
```

- Variables may be overwritten

```
greeting = "Hello, Hello!"
print(greeting)
greeting = "Hey!"
print(greeting)
```

Output:

```
Hello, Hello!
Hey!
```

Data Types and Operations

- Variables may store information of various types:

```
farewell = "Bye, Bye!" #String Type
num1 = 5 # Integer Type
num2 = 3.0 # Float Type
```

Data Types and Operations

- Variables may store information of various types:

```
farewell = "Bye, Bye!" #String Type
num1 = 5 # Integer Type
num2 = 3.0 # Float Type
```

- Operations may be performed using variables

```
print(num1+num2)
```

Output:

```
8.0
```

Data Types and Operations

- Results may again be stored in variables

```
num1 = 5
num2 = 3.0
num3 = num1+num2 #num3 is now 8.0
print(num3)
num3 = num3+1 #num3 updates based on its current value
print(num3)
```

Output:

```
8.0
9.0
```

Arithmetic Operators

```
2+2 # Addition
5-3 # Subtraction
5*6 # Multiplication
8/5 # Division
50-5*6 # Combining operators
(50-5*6)/4
17%3 # Remainder of the Division
```

Exercise: Variables

1. Create a variable to which you assign an integer. Create another variable to which you assign a float. Add up the two variables and store the result in a third variable. Print the value of that third variable.
2. Do the following once with variables and once in a single line without variables: Add 6 to 10, multiply the result by 5 and subtract 10 from this.

if statements

- An `if` statement executes a code block only if a condition is fulfilled.

```
x = 3.5
if x > 0: # Indentation organizes code blocks
    print("x is positive!") # Indent with 4 spaces
print("Program is finished!")
```

if statements

- An if statement executes a code block only if a condition is fulfilled.

```
x = 3.5
if x > 0: # Indentation organizes code blocks
    print("x is positive!") # Indent with 4 spaces
print("Program is finished!")
```

- Output:

```
x is positive!
Program is finished!
```

if statements

- An else statement executes a code block if the condition is not fulfilled.

```
x = 3.5
if x > 0: # Indentation organizes code blocks
    print("x is positive!") # Indent with 4 spaces
else :
    print("x is not positive!")
print("Program is finished!")
```

if statements

- An else statement executes a code block if the condition is not fulfilled.

```
x = 3.5
if x > 0: # Indentation organizes code blocks
    print("x is positive!") # Indent with 4 spaces
else :
    print("x is not positive!")
print("Program is finished!")
```

- Output:

```
x is positive!
Program is finished!
```

if Statements

- An elif statement allows chaining if statements

```
x = 3.5
if x > 0 : #Indentation organizes blocks
    print("x is positive!") #Indent with 4 spaces
elif x < 0 :
    print("x is negative!")
else:
    print("x is zero!")
print("Program is finished!")
```

if Statements

- An `elif` statement allows chaining if statements

```
x = 3.5
if x > 0 : #Indentation organizes blocks
    print("x is positive!") #Indent with 4 spaces
elif x < 0 :
    print("x is negative!")
else:
    print("x is zero!")
print("Program is finished!")
```

- Output:

```
x is positive!
Program is finished!
```

Comparison Operators

- Comparison operators are used to compare two values

```
3 > 2 # Greater than
3 < 3 # Less than
4 == 5 # Equal to
3 <= 3 # Less than or equal to
3 >= 3 # Greater than or equal to
4 != 5 # Not equal to
```

Exercise: Variables

1. Change x in the code cell on if-elif-else above so that it prints “x is negative!”
2. Write a script that determines whether a number is even.
   - Define a variable ‘num‘ and assign it a number of your choice.
   - Use If and Else to print out either "The number is even" or "The number is odd" depending on the value of ‘num‘.
3. Write a script that takes a frequency in Hz and prints out the corresponding EEG frequency band. (see the Jupyter notebook for more details)

While Loops

- A `while` statement executes a block of code while a condition is fulfilled.
- Print the numbers from 1 to 10

```
a = 1
while a <= 10 :
    print(a)
    a = a + 1 # Increase a by 1
```

While Loops

- A `while` statement executes a block of code while a condition is fulfilled.
- Print the numbers from 1 to 10

```
a = 1
while a <= 10 :
    print(a)
    a = a + 1 # Increase a by 1
```

- Output:

```
1
2
3
...
10
```

While Loops

- Be careful with the exit condition

```
a = 1
while a <= 10 :
    print(a) # Prints 0 until the end of time
```

Exercise: While Loops

1. Write a program that prints the numbers from 20 to 1 in descending order.

1. **Motivation**

2. **Programming**
   - Python
   - First Steps
   - Variables
   - if Statements
   - Loops

3. **Functions**

4. **Lists**
   - for Loops

Functions in Python

- A function in programming is a block of code that performs a particular task:

```python
def greeting(): # 'greeting' is the function name
  print("Hello!")
```

- A function is only executed when it is called:

```python
greeting()
```

Functions in Python

- A function in programming is a block of code that performs a particular task:

```
def greeting(): # 'greeting' is the function name
  print("Hello!")
```

- A function is only executed when it is called:

```
greeting()
```

- Output:

```
Hello!
```

Arguments

- A function may receive an argument, which can be passed to the function and be used inside the function:

```
def greeting(name):
    print("Hello " + name)

greeting("Alice")
myname = "Bob"
greeting(myname)
```

Arguments

- A function may receive an argument, which can be passed to the function and be used inside the function:

```
def greeting(name):
    print("Hello " + name)

greeting("Alice")
myname = "Bob"
greeting(myname)
```

- Output:

```
Hello Alice
Hello Bob
```

Returning Values

- Functions may return values to the program

```
def square(value):
  return value*value
```

- The return value may be assigned like any variable

```
x = 3
y = square(x)
print(y)
```

Returning Values

- Functions may return values to the program

```
def square(value):
  return value*value
```

- The return value may be assigned like any variable

```
x = 3
y = square(x)
print(y)
```

- Output:

```
9
```

Returning Values

- Functions may be chained

```
square(square(2))
```

Returning Values

- Functions may be chained

```
square(square(2))
```

- Output:

```
16
```

Multiple Arguments

- Functions may have multiple arguments

```
def subtract(minuend,subtrahend):
  return minuend-subtrahend

print(subtract(8, 5))
```

Multiple Arguments

- Functions may have multiple arguments

```
def subtract(minuend,subtrahend):
  return minuend-subtrahend

print(subtract(8, 5))
```

- Output:

```
3
```

Exercise: Functions

1. Write a function “add(a, b)” that adds two numbers
2. Write a function “add3(a, b, c)” that adds three numbers by calling the add function two times.
3. Write a function “max(a, b)” that returns the maximum of a and b

The List Datatype

- Lists allow to store a collection of values

```
names = ["Alice","Bob","Carl","Dora"]
numbers = [1,2,3,5,8]
```

- List elements can be accessed via their index

```
print(names[2]) # Carl
single_name = names[2] # single_name = 'Carl'
first_element = numbers[0] #first_element = 1
names[1] = "Bert" #names ['Alice','Bert','Carl','Dora']
```

Operations on Lists

```
names = ["Alice","Bert","Carl","Dora"]
numbers = [1,2,3,5,8]
```

- Example Operations

```
len(names) # Get the length (4)
names.append("Eric") # Append a value [...,"Dora","Eric"]
names + numbers # Concatenate lists ["Alice", ..., 1, 2, ..
numbers[1:4] # Get a subset of the list (2,3,5)
```

for Loops

- A `for` loop can be used to iterate over a list

```
for name in names:
    print(name)
```

Exercise: Lists

1. Create a list that contains 5 different float values
2. Append a 6th float value to the same list
3. Create a second list with 5 other float values and concatenate the two lists.
4. Get a subset of that list containing the 4th to 7th elements
5. Create a for loop that prints the elements of the list one by one. Do the same with a while loop.
6. Create a new list which is (automatically) filled with the squared values of the original list.