Lecture 1

Introduction to Variables and Control Statements

Jan Tekülve

jan.tekuelve@ini.rub.de

Computer Science and Mathematics
Preparatory Course

12.10.2020

Course Formalities

Goals:

- Learning basic programming with Python
- Refreshing elementary mathematical concepts

Concept:

- Each lecture will usually be split into a theoretical explanation and a programming session
- On the last day (23.10.) there will be an ungraded “test”

Corona Formalities

https://rub.corona-erfassung.de/users/newuser

Course Number: 310024

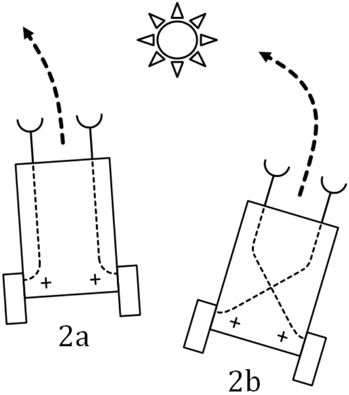
Overview

1. **Motivation**

2. **Programming**
   - First Steps
   - Variables
   - Control Statements
   - Utilities

3. **Tasks**

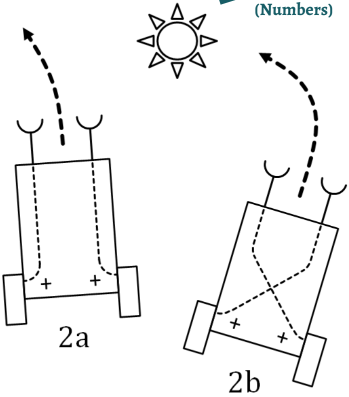
Motivation: Modeling a cognitive agent



Braitenberg Vehicles

[Braitenberg, 1986]

Motivation: Modeling a cognitive agent

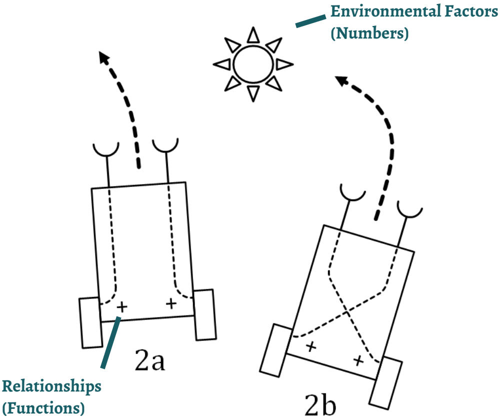


Environmental Factors (Numbers)

Braitenberg Vehicles

[Braitenberg, 1986]

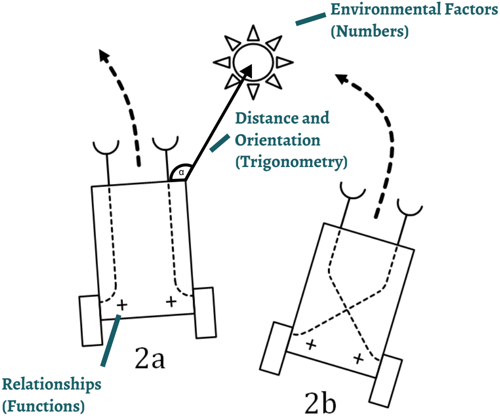
Motivation: Modeling a cognitive agent



Braitenberg Vehicles

[Braitenberg, 1986]

Motivation: Modeling a cognitive agent



Environmental Factors (Numbers)

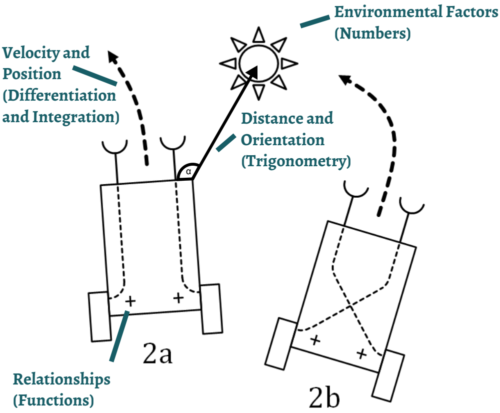
Distance and Orientation (Trigonometry)

Relationships (Functions)

Braitenberg Vehicles

[Braitenberg, 1986]

Motivation: Modeling a cognitive agent



Environmental Factors (Numbers)

Velocity and Position (Differentiation and Integration)

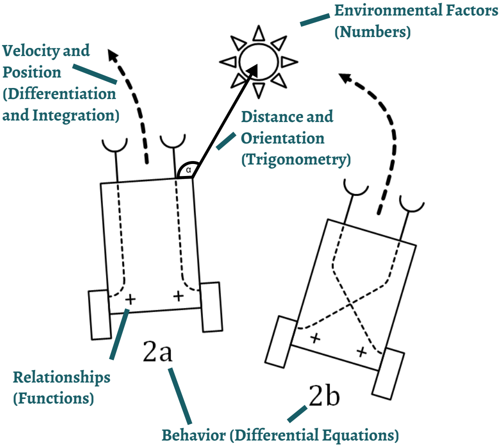
Distance and Orientation (Trigonometry)

Relationships (Functions)

Braitenberg Vehicles

[Braitenberg, 1986]

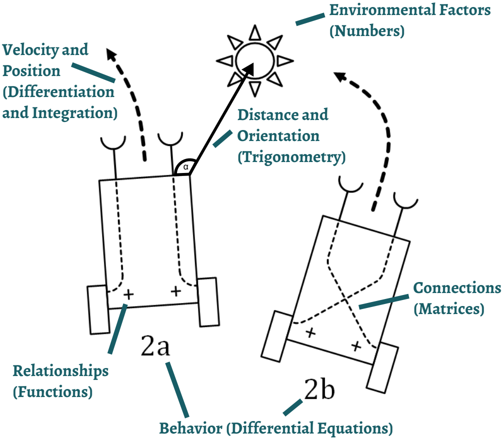
Motivation: Modeling a cognitive agent


Environmental Factors (Numbers)
Velocity and Position (Differentiation and Integration)
Distance and Orientation (Trigonometry)
α
+
+
+
+
2a
2b
Relationships (Functions)
Behavior (Differential Equations)

Braitenberg Vehicles

[Braitenberg, 1986]

Motivation: Modeling a cognitive agent



Environmental Factors (Numbers)

Velocity and Position (Differentiation and Integration)

Distance and Orientation (Trigonometry)

Connections (Matrices)

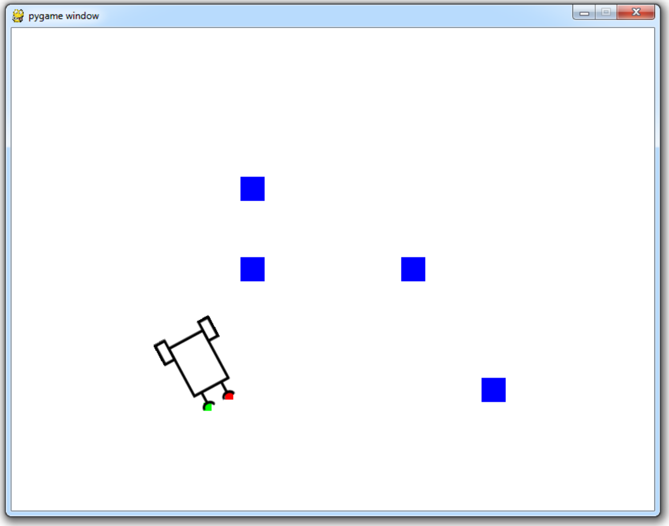
Relationships (Functions)

Behavior (Differential Equations)

Braitenberg Vehicles

[Braitenberg, 1986]

Programming Goal



Course Structure

#	Date	Title	Topics
1	12.10.	Variables and Control Statements	*Data Types, Control Statements*
2	13.10.	Functions in Math and Programming	*Function Types and Properties, Plotting Functions, Lists*
3	14.10.	Full-Time Programming Session	*Deepen Programming Skills*
4	15.10.	Coordinate Systems	*Vectors, Trigonometry, The Pygame Module*
5	16.10.	Differentiation	*Derivative Definition, Calculating Derivatives, Numerical Differentiation, File-Input/Output*

Course Structure

#	Date	Title	Topics
6	19.10.	Integration	*Geometrical Definition, Calculating Integrals, Numerical Integration*
7	20.10.	Differential Equations	*Properties of Differential Equations, Euler Approximation, Braitenberg Vehicle*
8	21.10.	Programming Session & Recap	*Repetition, Questions, Test Topics*
9	22.10.	"Make a wish Lecture"	*Individual Wishes, e.g. Object-Oriented Programming, Matrix Calculation*
10	23.10.	"Test"	*Self-evaluation*

1. Motivation

2. **Programming**
   - First Steps
   - Variables
   - Control Statements
   - Utilities

3. Tasks

The Python Programming Language

Why Python?

- It is simple but high level
- It is interpreted “on the fly”
- It is the state of the art scripting language

Helpful Resources

- The Anaconda Distribution contains all necessary software: https://www.anaconda.com/distribution/
- You can find helpful documentation here: https://docs.python.org/3/

Setting Up

- Open the *Spyder* IDE (Integrated Development Environment)
- Create your first python script file
  - Close the default temporary file
  - Go to *File* → *Save as ...*
  - *(Recommended)* Create a new folder for your python projects
  - Choose the name *helloworld.py*
- You are set up to write your first Python script!

Hello World

- Write the following line into the file:

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

- Press the green *Play* button in the toolbar to execute the script
- Observe the output in the console on the right

Hello World

- Write the following line into the file:

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

- Press the green *Play* button in the toolbar to execute the script
- Observe the output in the console on the right
- The *print()* function writes its argument to the console

Script: 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?")
```

Script: 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?")
```

- 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
```

Variables

- Variables are the elementary building block of every program

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

Variables

- Variables are the elementary building block of every program

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

- Variables are assigned via ‘=’

```
var1 = "Hello" #variable names may be chosen arbitrarily
long_variable_name5 = "Hi"
#letters, numbers and underscores may make up a name
```

Variables

- Variables are the elementary building block of every program

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

- Variables are assigned via ‘=’

```
var1 = "Hello" #variable names may be chosen arbitrarily
long_variable_name5 = "Hi"
#letters, numbers and underscores may make up a name
```

- Assigned variables are available for code following the assignment

```
print(greeting) #prints "Hello, Hello!"
greeting = "Hey!" #variables may be overwritten
print(greeting) #prints "Hey!"
```

Data Types and Operations

- Variables store information of various type:

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

Data Types and Operations

- Variables store information of various type:

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

- Operations may be performed using variables

```
print(num1+num2) #prints 8.0
```

Data Types and Operations

- Variables store information of various type:

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

- Operations may be performed using variables

```
print(num1+num2) #prints 8.0
```

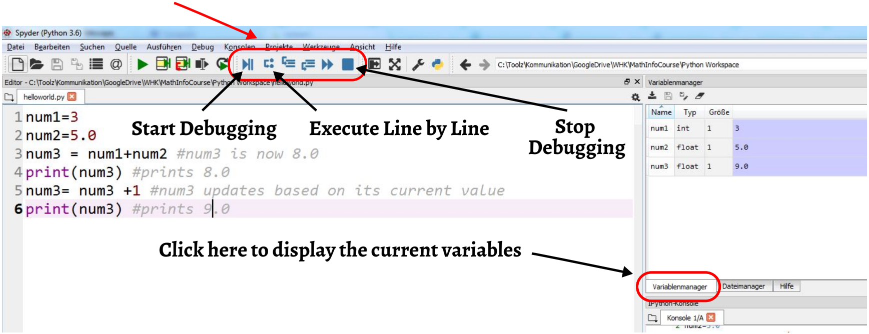
- Results may again be stored in variables

```
num3 = num1+num2 #num3 is now 8.0
print(num3) #prints 8.0
num3 = num3+1 #num3 updates based on its current value
print(num3) #prints 9.0
```

Excursion: The Spyder Debugger

- A debugger allows a look under the 'hood' of a program

These are the Debug Controls



```
num1=3
num2=5.0
num3 = num1+num2 #num3 is now 8.0
print(num3) #prints 8.0
num3= num3 +1 #num3 updates based on its current value
print(num3) #prints 9.0
```

Name	Typ	Größe	
num1	int	1	3
num2	float	1	5.0
num3	float	1	9.0

Start Debugging

Execute Line by Line

Stop Debugging

Click here to display the current variables

Useful Operations on Data Types

- Operations on Numbers

```
2+2 #4
50-5*6 #20
(50-5*6)/4 #5.0
8/5 #1.6
17/3  #5.666666666666667
17//3  #5 Integer Division
17%3  #2  Rest of the Division
```

- Operations on Strings

```
'Wo' + 'rd' #'Word' or "Word"
'Isn't' # This results in an error!
'Isn\'t' #'Isn't' Use \ to escape characters
```

Control Statements

- if-Statement

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

Control Statements

- if-Statement

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

- else-statement

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

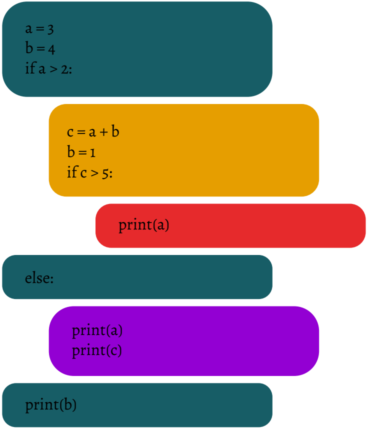
Control Statements

- else if-statement

```
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!")
```

Variable Scope

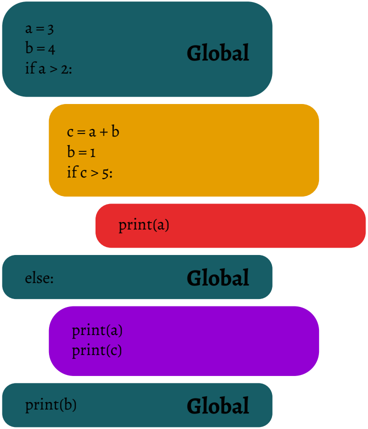
- Python code is organized in blocks by indentation (4 spaces)



```
a = 3
b = 4
if a > 2:
    c = a + b
    b = 1
    if c > 5:
        print(a)
else:
    print(a)
    print(c)
print(b)
```

Variable Scope

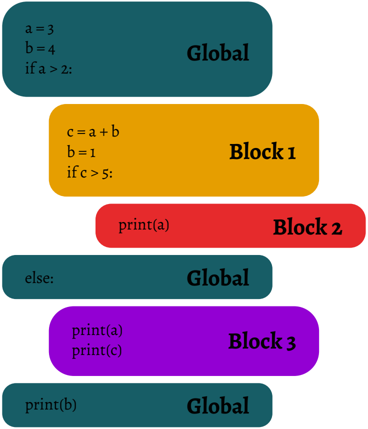
- Python code is organized in blocks by indentation (4 spaces)
- Variables defined in the global scope are available at all positions in the code below its definition



```
a = 3
b = 4
if a > 2:
    c = a + b
    b = 1
    if c > 5:
        print(a)
else:
    print(a)
    print(c)
print(b)
```

Variable Scope

- Python code is organized in blocks by indentation (4 spaces)
- Variables defined in the global scope are available at all positions in the code below its definition
- Variables defined in a block are available in the block and all blocks inside it



a = 3
b = 4
if a > 2:
Global

c = a + b
b = 1
if c > 5:
Block 1

print(a)
Block 2

else:
Global

print(a)
print(c)
Block 3

print(b)
Global

Variable Scope

- Example

```
a = 3 # Global Scope
b = 4
if a > 2 :
    c = a + b # Block 1
    b = 1
    if c > 5:
       print(a) # Block 2
else : # Global
   print(a) # Block 3
   print(c) # If a <= 2 this will result in an error
print(b) # '1' or '4' if a <= 2
```

While Loops

- Print the numbers from 1 to 10

```
a = 0
while a < 10 :
    a = a +1 # Increase a by 1
    print(a)
```

While Loops

- Print the numbers from 1 to 10

```
a = 0
while a < 10 :
    a = a +1 # Increase a by 1
    print(a)
```

- Be careful with the exit condition

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

You can kill the running program by pressing the red terminate button

Boolean Statements

- Examples

```
3 > 2 #True, greater than
3 < 3 #False, less than
3 <= 3 # True, equal or less than
4 == 5 # False, == checks equality
4 != 5 # True, != is the opposite of ==
"ello" in "Hello" # True, only works for sequence types
"hel" not in "Hello" # True, "in" is case sensitive
```

Boolean Statements

- Examples

```
3 > 2 #True, greater than
3 < 3 #False, less than
3 <= 3 # True, equal or less than
4 == 5 # False, == checks equality
4 != 5 # True, != is the opposite of ==
"ello" in "Hello" # True, only works for sequence types
"hel" not in "Hello" # True, "in" is case sensitive
```

- Boolean Variables

```
test = 7
isGreaterThanOne = test > 1
if isGreaterThanOne:
    print("The number is Greater than 1!")
```

User Input

- Use input to prompt the user

```
person = input('Enter your name: ')
print('Hello ' + person)
```

User Input

- Use input to prompt the user

```
person = input('Enter your name: ')
print('Hello ' + person)
```

- Invalid Data Types

```
inputValue = input('Please enter a number: ')
result = 5 + inputValue # This results in an error!
```

User Input

- Use input to prompt the user

```
person = input('Enter your name: ')
print('Hello ' + person)
```

- Invalid Data Types

```
inputValue = input('Please enter a number: ')
result = 5 + inputValue # This results in an error!
```

- Variables might need to be *type casted*

```
result = 5 + float(inputValue)
#This works if an actual number was typed
```

Type Casting

- Implicit Typecast

```
a = 1.0 #float
b = 2 #int
c = a + b #3.0 float
```

Type Casting

- Implicit Typecast

```
a = 1.0 #float
b = 2 #int
c = a + b #3.0 float
```

- Explicit Typecasts

```
d = float(b) #2.0
e = 3.7
f = int(3.7) #3 Any floating point is cut off
g = str(e) #String '3.7'
h = int(g) # This results in an error!
i = float(g) # 3.7
print('Variable i is: ' +str(i)) #Print expects strings
```

Useful built-in Functions

- Rounding and Absolute Value

```
a = 3.898987897897
b = round(a,3) #3.899
c = abs(-3.2) #|-3.2| = 3.2
t = type(c) #t is <class 'float'>
test = t is float # True
```

- The math module

```
import math #Import makes a module available
squareTwo = math.sqrt(2) #$\sqrt{2}$
power = math.pow(3,4) # $3^4$
exponential = math.exp(4) #$e^4$
piNumber = math.pi #3.14159265359
```

Lecture Slides/Material

Use the following URL to access the lecture slides:

https://www.ini.rub.de/teaching/courses/c_science_math_2020

Tasks

1. Write a script that takes a percentage and prints out the corresponding verbal grade.

- Define a variable *perc* and assign it a number between 1 and 100.
- Use If and Else to print out the correct grade depending on the value of *perc*.

%	Grade	%	Grade
86-100	A	40-55	D
71-85	B	25 -39	E
56-70	C	1 - 24	F

2. Write a script that asks the user for two different inputs and prints their sum

   - Define a variable *num*1 and assign it a value using the *input*() function
   - Repeat the above step for a second variable *num*2
   - Add *num*1 and *num*2 together in a third variable *sum* and print it (Do not forget to typecast *num*1 and *num*2)

Tasks Continued

3*. Write a script that asks the user for number input until the sum of the inputs is greater than 20.

- Start with a variable *sum* that is initialized with the value 0.
- Create a while-loop that ends when *sum* is greater than 20.
- Inside the while-loop ask the user for input and add the input to *sum*. (Do not forget to typecast the input)

4*. Write a script that finds the maximum number out of 3 numbers.

- Example:
  You choose the three numbers to be 13, 16 and 5.
  The program should print: “The highest number is 16”.
- Define three variables each containing a different number.
- Use If and Else statements to find the highest of the three numbers.
- Print the number to the console.
- The script should work for any three numbers.

References

Braitenberg, V. (1986).
Vehicles: Experiments in synthetic psychology.
MIT press.