

Lecture 2

Functions in Math and Programming

Jan Tekülve

jan.tekuelve@ini.rub.de

Computer Science and Mathematics
Preparatory Course

20.09.2019

Overview

1. Motivation

2. Functions in Math

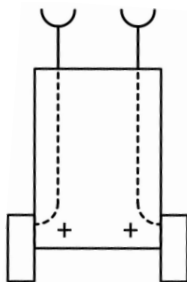
- Definition
- Parametrization
- Function Types
- Properties

3. Programming

- Functions
- Lists
- Writing Files

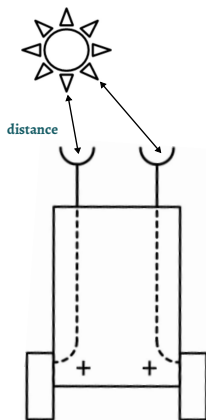
4. Tasks

Functions in Braitenberg Vehicles



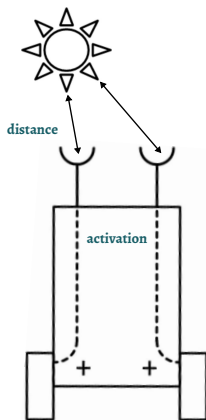
2a

Functions in Braitenberg Vehicles



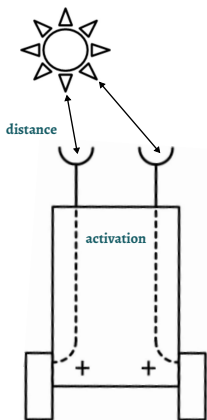
2a

Functions in Braitenberg Vehicles

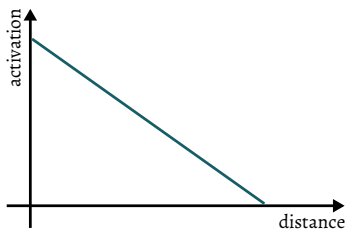


2a

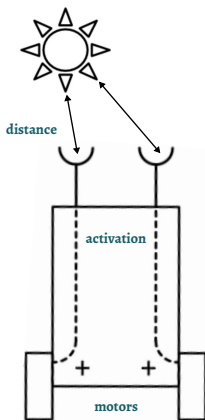
Functions in Braitenberg Vehicles



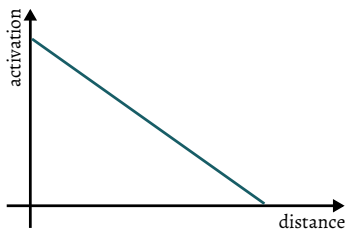
2a



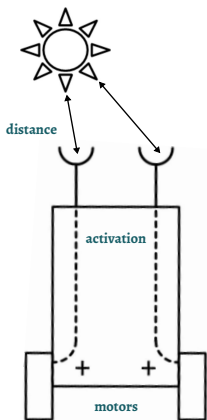
Functions in Braitenberg Vehicles



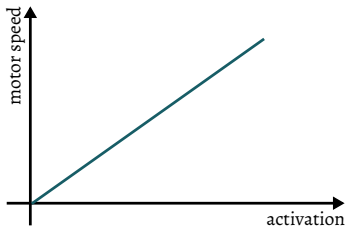
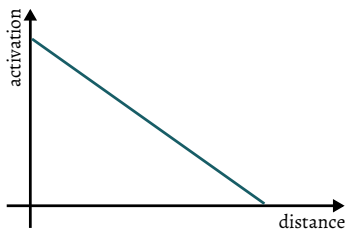
2a



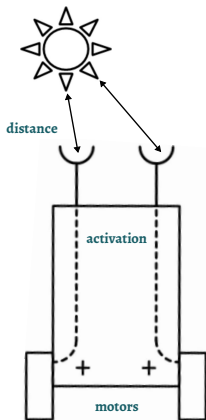
Functions in Braitenberg Vehicles



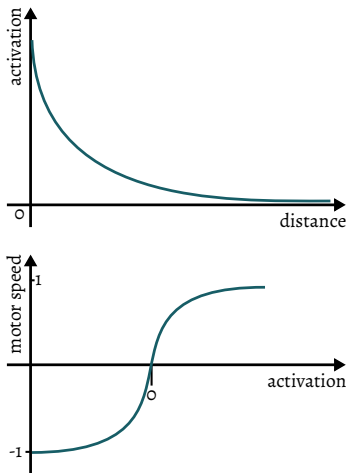
2a



Functions in Braitenberg Vehicles



2a



1. Motivation

2. Functions in Math

- Definition
- Parametrization
- Function Types
- Properties

3. Programming

- Functions
- Lists
- Writing Files

4. Tasks

Function Definition

Function

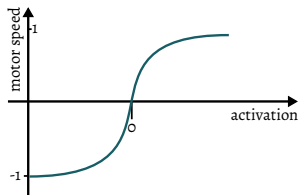
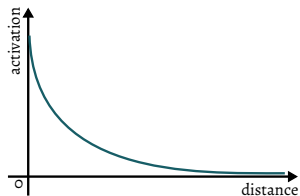
X and Y are two non-empty sets.

A **function** $f : X \rightarrow Y$, assigns each element $x \in X$ exactly one element $y \in Y$.

$$x \rightarrow y = f(x)$$

- ▶ x is called the **function argument**
- ▶ y is called the **function value**
- ▶ X is called the **domain of a function**
- ▶ Y is called the **codomain of a function**
- ▶ The **image** W of $f(x)$ are all values in Y that can be assumed

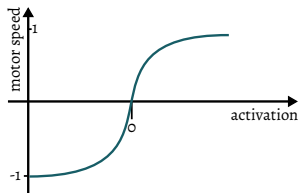
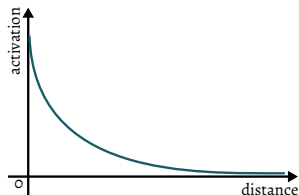
Braitenberg Examples



Braitenberg Examples

► Assumptions:

- Distance d may assume positive values
- Activation a may assume any value
- Motor Speed s may assume any value between -1 and 1



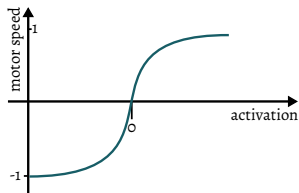
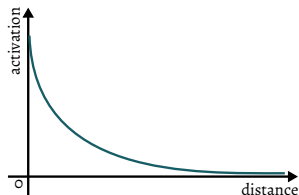
Braitenberg Examples

► Assumptions:

- Distance d may assume positive values
- Activation a may assume any value
- Motor Speed s may assume any value between -1 and 1

► $f : d \rightarrow a$

- $f(d) = e^{-d}$
- $X = \mathbb{R}^+$, $Y = \mathbb{R}$
- $W = \mathbb{R}^+$



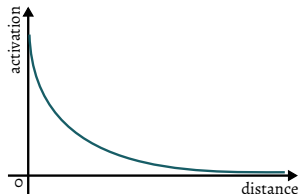
Braitenberg Examples

► Assumptions:

- Distance d may assume positive values
- Activation a may assume any value
- Motor Speed s may assume any value between -1 and 1

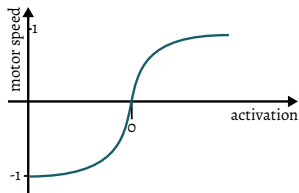
► $f : d \rightarrow a$

- $f(d) = e^{-d}$
- $X = \mathbb{R}^+, Y = \mathbb{R}$
- $W = \mathbb{R}^+$



► $g : a \rightarrow s$

- $g(a) = \tanh(a)$
- $X = \mathbb{R}, Y = \mathbb{R}$
- $W = [-1, 1]$



Plotting Functions

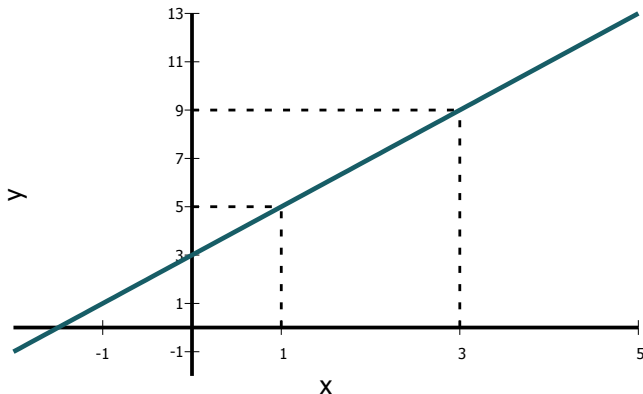
Tabular Interpretation of: $f(x) = 2x + 3$

x	0	1	2	3	4	5
y	3	5	7	9	11	13

Plotting Functions

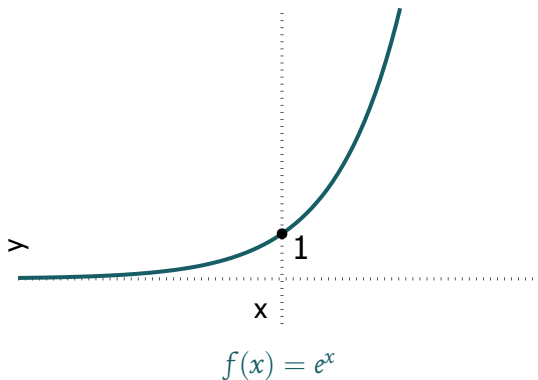
Tabular Interpretation of: $f(x) = 2x + 3$

x	0	1	2	3	4	5
y	3	5	7	9	11	13



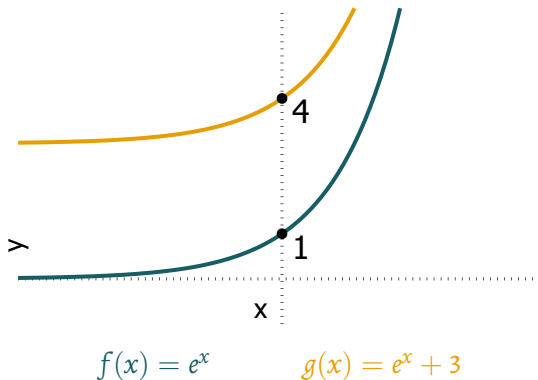
Function Translation

- ▶ Translation in y -direction: $\hat{f}(x) = f(x) + b$
- ▶ Translation in x -direction: $\hat{f}(x) = f(x - a)$



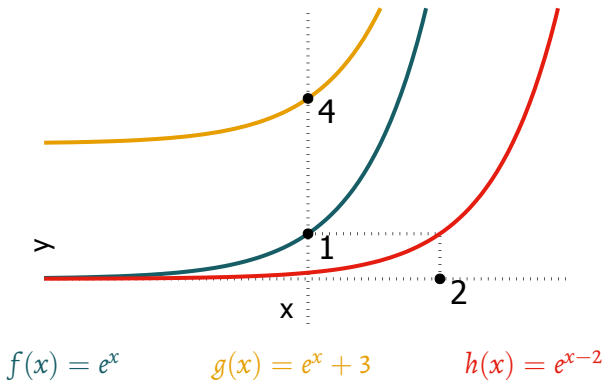
Function Translation

- ▶ Translation in y -direction: $\hat{f}(x) = f(x) + b$
- ▶ Translation in x -direction: $\hat{f}(x) = f(x - a)$



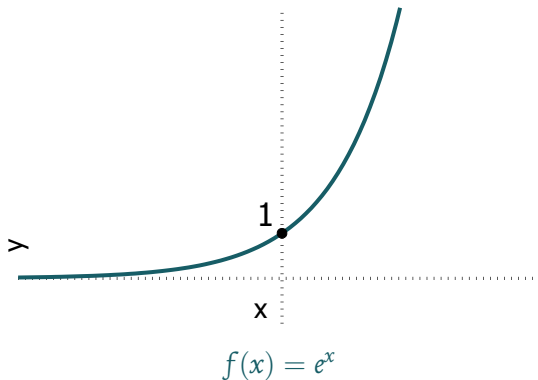
Function Translation

- Translation in y -direction: $\hat{f}(x) = f(x) + b$
- Translation in x -direction: $\hat{f}(x) = f(x - a)$



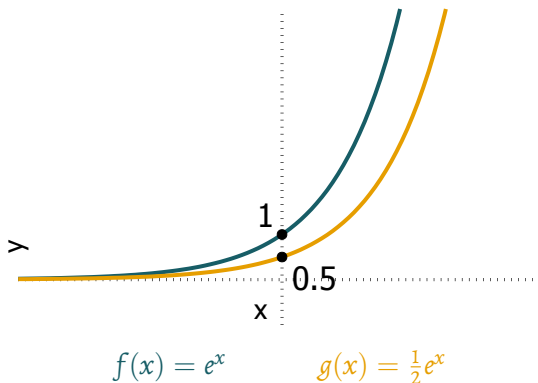
Function Stretching and Compression

- ▶ Stretching/Compression in **y-direction**: $\hat{f}(x) = df(x), d > 0$
- ▶ Stretching/Compression in **x-direction**: $\hat{f}(x) = f(cx), c > 0$



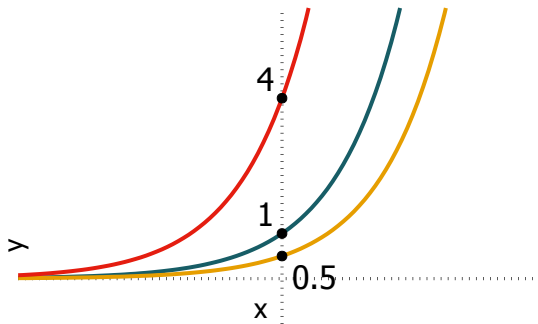
Function Stretching and Compression

- ▶ Stretching/Compression in **y-direction**: $\hat{f}(x) = d f(x), d > 0$
- ▶ Stretching/Compression in **x-direction**: $\hat{f}(x) = f(cx), c > 0$



Function Stretching and Compression

- ▶ Stretching/Compression in **y-direction**: $\hat{f}(x) = df(x), d > 0$
- ▶ Stretching/Compression in **x-direction**: $\hat{f}(x) = f(cx), c > 0$



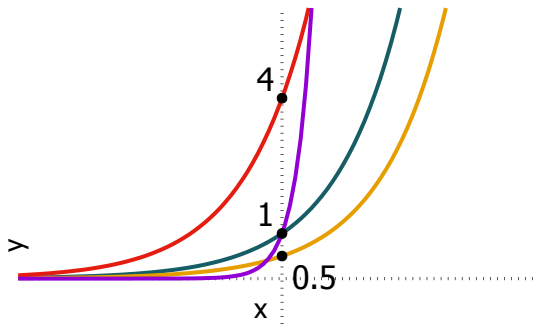
$$f(x) = e^x$$

$$g(x) = \frac{1}{2}e^x$$

$$h(x) = 4e^x$$

Function Stretching and Compression

- ▶ Stretching/Compression in **y-direction**: $\hat{f}(x) = df(x), d > 0$
- ▶ Stretching/Compression in **x-direction**: $\hat{f}(x) = f(cx), c > 0$



$$f(x) = e^x$$

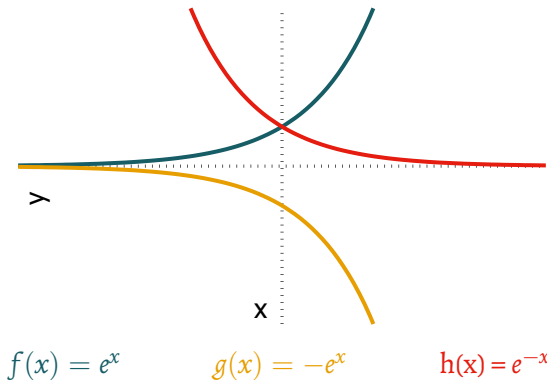
$$g(x) = \frac{1}{2}e^x$$

$$h(x) = 4e^x$$

$$j(x) = e^{4x}$$

Function Reflection

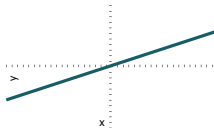
- ▶ Reflection across the **y-axis**: $\hat{f}(x) = f(-x)$
- ▶ Reflection across the **x-axis**: $\hat{f}(x) = -f(x)$



Function Types

► Linear Functions

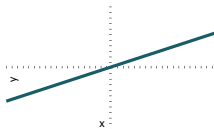
$$y = mx + b$$



Function Types

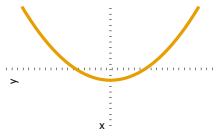
► Linear Functions

$$y = mx + b$$



► Power Functions

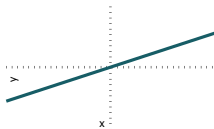
$$y = ax^n$$



Function Types

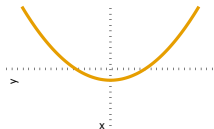
► Linear Functions

$$y = mx + b$$



► Power Functions

$$y = ax^n$$

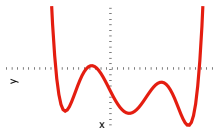


► Polynomial Functions

$$y = \sum_{i=0}^n a_i x^i$$

$$y = a_0 + a_1x + a_2x^2 + a_3x^3 + \dots a_nx^n$$

describe a polynomial of degree n , where $a_n \neq 0$

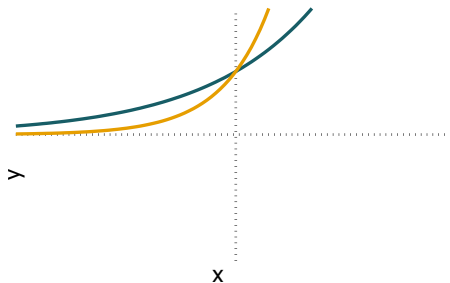


Exponentials Functions

Exponential Functions

$$f(x) = e^x$$

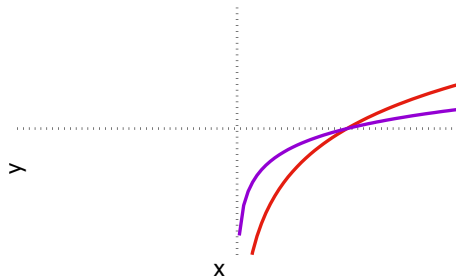
$$g(x) = 10^x$$



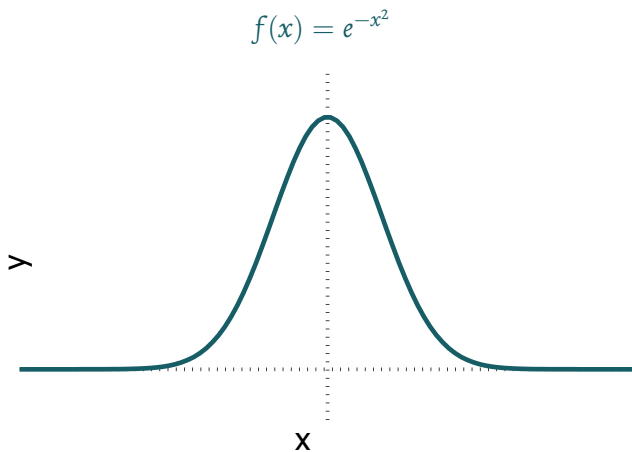
Logarithmic Functions

$$h(x) = \ln(x)$$

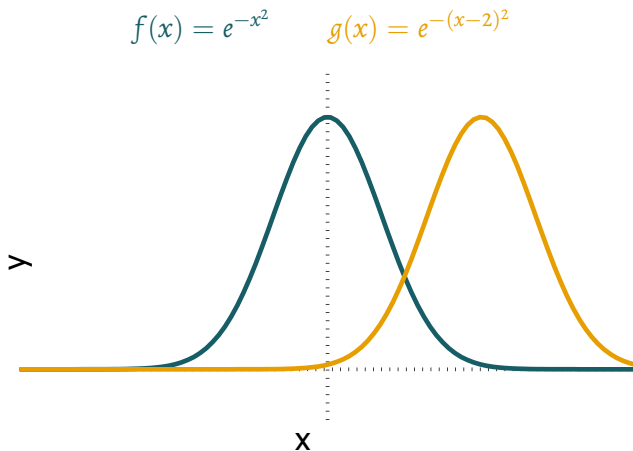
$$j(x) = \log_{10}(x)$$



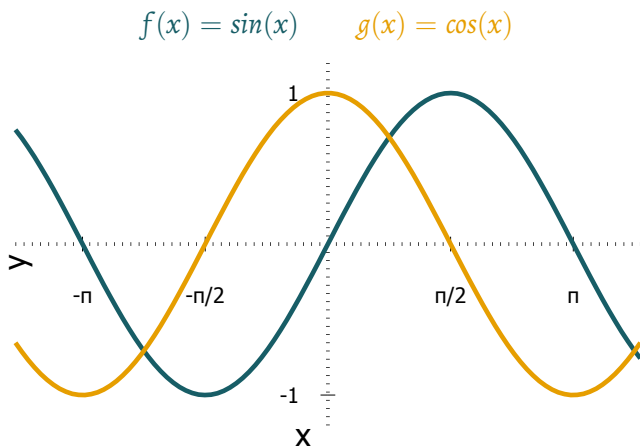
The Gaussian Function



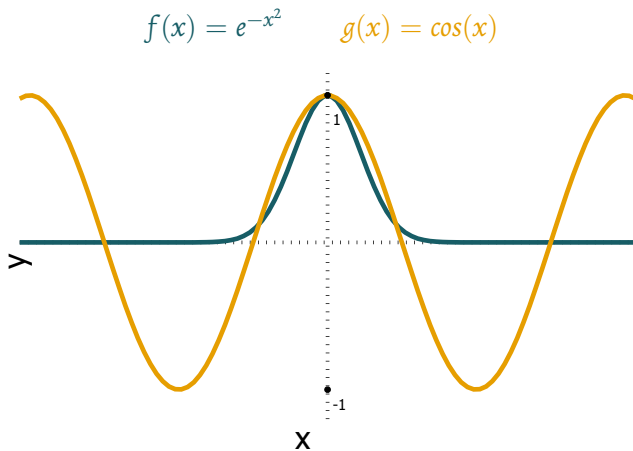
The Gaussian Function



Trigonometric Functions



Chaining Functions

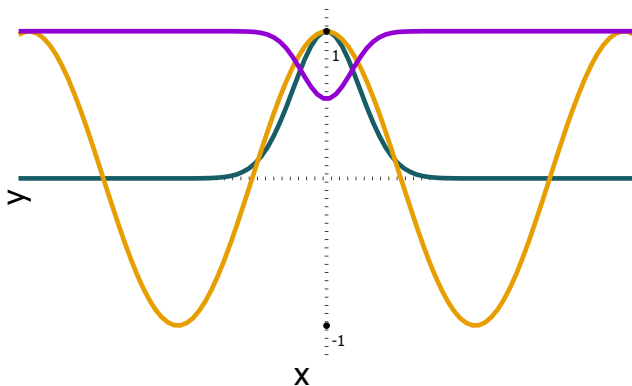


Chaining Functions

$$f(x) = e^{-x^2}$$

$$g(x) = \cos(x)$$

$$h(x) = g(f(x))$$



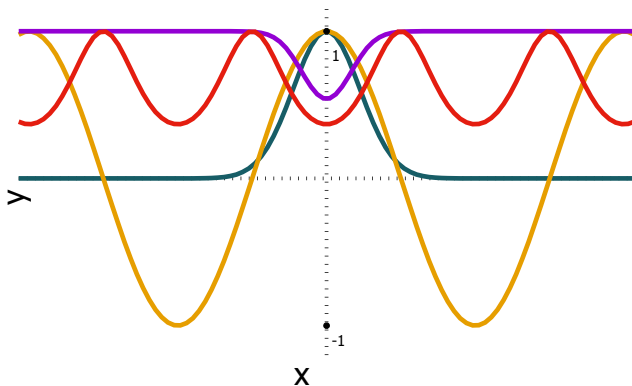
Chaining Functions

$$f(x) = e^{-x^2}$$

$$g(x) = \cos(x)$$

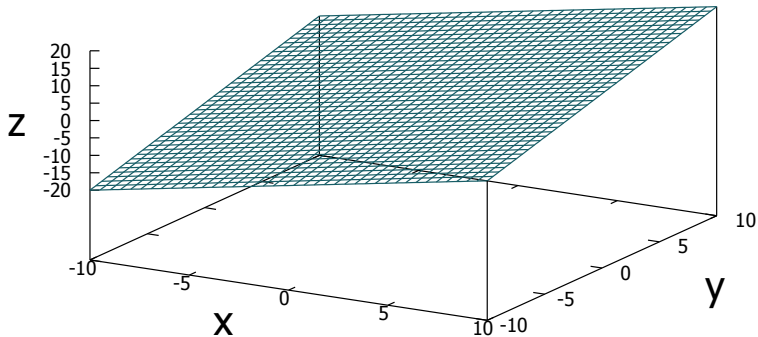
$$h(x) = g(f(x))$$

$$j(x) = f(g(x))$$



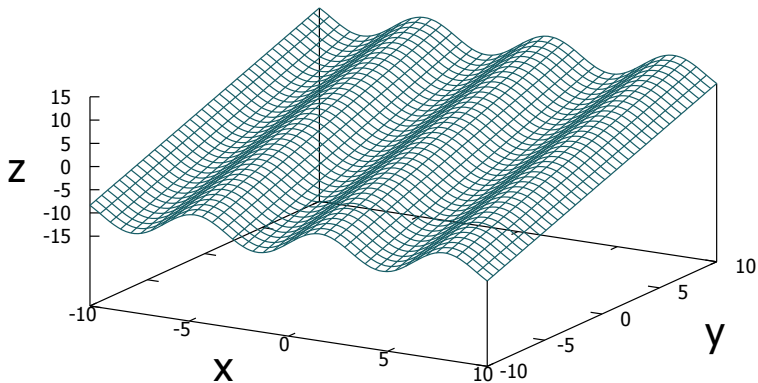
Multiple Function Arguments

$$f(x, y) = x + y$$



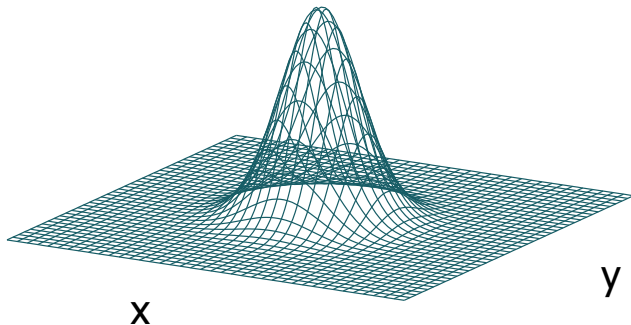
Multiple Function Arguments

$$f(x,y) = \sin(x) + y$$



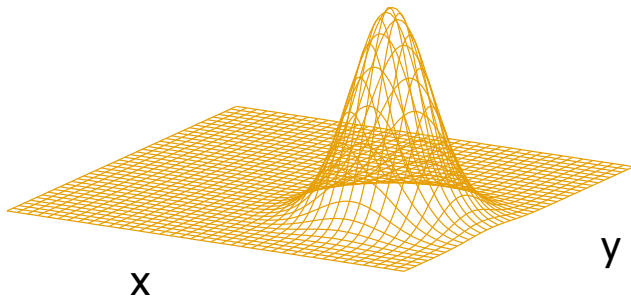
Multiple Function Arguments

$$f(x, y) = e^{-(x^2+y^2)}$$



Multiple Function Arguments

$$f(x, y) = e^{-((x-2)^2 + (y+1)^2)}$$

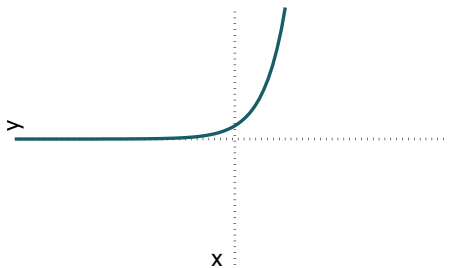


Injective and Surjective

- ▶ An image f is **injective**, if two different elements $x_1 \neq x_2$ are always projected to two different elements $y_1 \neq y_2$
- ▶ An image f is **surjective**, if for each element $y \in Y$ one $x \in X$ exists, such that $y = f(x)$

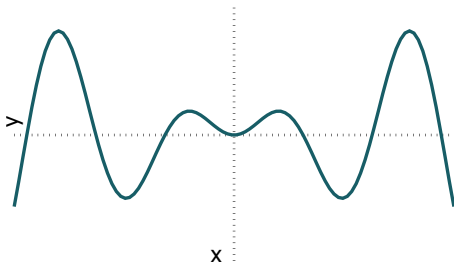
Injective, but not surjective

$$f(x) = e^x$$



Surjective, but not Injective

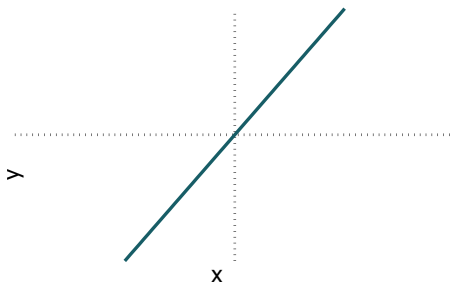
$$f(x) = x \sin(x)$$



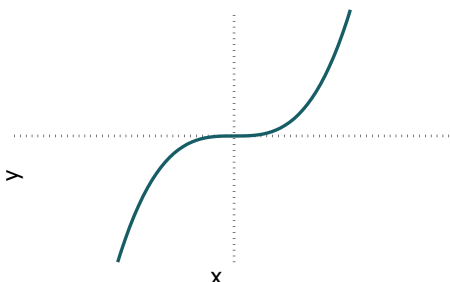
Bijjective Functions

- An image f is **bijjective**, if it is injective and surjective

$$f(x) = 4x$$



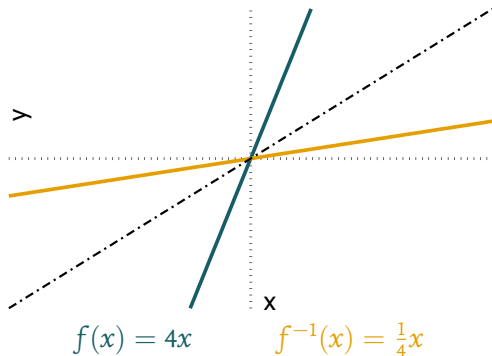
$$f(x) = x^3$$



Inverse Function

Definition

If a **function** $f : X \rightarrow Y$ is bijective, then $f^{-1} : Y \rightarrow X$ describes the **inverse function** of f



Monotonicity

Definition

- ▶ A function $f : \mathbb{R} \rightarrow \mathbb{R}$ is called **monotonically increasing**, if for all x_1, x_2 order is preserved by applying f :

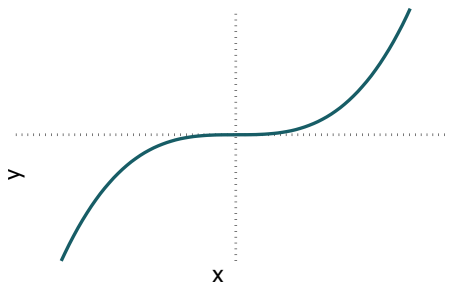
$$x_1 \leq x_2 \Rightarrow f(x_1) \leq f(x_2)$$

- ▶ A function $f : \mathbb{R} \rightarrow \mathbb{R}$ is called **monotonically decreasing**, if for all x_1, x_2 order is reversed by applying f :

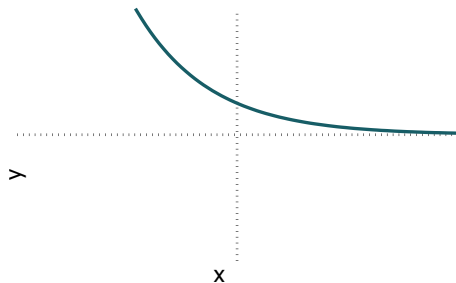
$$x_1 \leq x_2 \Rightarrow f(x_1) \geq f(x_2)$$

Monotonicity Examples

monotonically increasing



monotonically decreasing



1. Motivation

2. Functions in Math

- Definition
- Parametrization
- Function Types
- Properties

3. Programming

- Functions
- Lists
- Writing Files

4. Tasks

Functions in Python

- ▶ Functions in programming describe a parameterizable routine
- ▶ Functions are defined like this:

```
def greeting(person): #greeting is the function name
    print("Hello " +person) #person is the argument
#print is also a function
```

- ▶ Functions are called like this:

```
greeting("Bob") #"Hello Bob"
name = "Alice"
greeting(name) #"Hello Alice"
```

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) #y=9, x=3 arguments stay the same  
square(square(2)) #8 Functions may be chained
```

- ▶ ! Data types are not explicitly specified

```
x = square("Bob") #This results in an error!
```

Multiple Function Arguments

- ▶ Functions may have multiple arguments

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

- ▶ Advanced Function calling:

```
result = subtract(9,2) #result=7  
result = subtract(minuend=9,subtrahend=2) #result=7  
result = subtract(subtrahend=2,minuend=9) #result=7  
result = subtract(9,subtrahend=2) # error!
```

Optional Function Arguments

- Some Arguments may be declared optional

```
def subtract(minuend, subtrahend, console=False):  
    if console:  
        print(str(minuend-subtrahend))  
    return minuend-subtrahend
```

- Optional arguments may be ignored while calling:

```
result = subtract(9,2) #result=7  
result = subtract(9,2,True) #result=7 and prints  
result = subtract(9,2,False) #is equal to subtract(9,2)
```

The List Datatype

- ▶ Lists allow to manage a collection of variables

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

- ▶ Accessing and modifying elements in a lists

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

Operations on Lists

► Example Operations

```
numbers = [1,2,3,5,8]
names = ["Alice","Bob","Carl"]
count = len(names) #count=3
names.append("Daisy") #['Alice','Bob','Carl','Daisy']
numbers2 = [13,21,34]
numbers3 = numbers + numbers2 #[1,2,3,5,8,13,21,34]
subset = numbers3[2:5] #[3,5,8]
#characters from position 2 (included) to 5 (excluded)
```

The For Loop

- ▶ The For Loop runs for a fixed number of times

```
for x in range(0, 3): This runs a loop 3 times
    print(x) # prints 0,1,2
```

- ▶ x is automatically incremented with each iteration
- ▶ A while-loop would look like this

```
counter = 0
while counter < 3:
    print(counter)
    counter = counter + 1
```

Lists and Loops

- ▶ The for-loop is especially helpful for lists

```
numbers = [3,7,9]
for number in numbers: #This runs for each list element
    square = number * number
    print(square) #9,49,81
```

- ▶ A while-loop would look like this

```
counter = 0
while counter < len(numbers):
    square = numbers[counter] * numbers[counter]
    counter = counter + 1
    print(square)
```

Writing Files

► Opening a file

```
#This creates the file if it does not exist  
fileObject = open("fileOutput.txt", "w")  
#Option 'w' will overwrite existing files  
#Use the option 'a' to append to a file instead
```

► Writing to the file

```
#Add \n to end a line and \t to create a tab  
fileObject.write("Hello you!\n")
```

► Close the file after usage:

```
fileObject.close()
```

Tasks

1. Write a function that calculates $f(x) = 4x + 3$ for any number x
 - ▶ Define the function with a single parameter and a return value
 - ▶ Call the function with three different inputs and print the output each time
2. Define a second function $g(x, a_0, a_1, a_2, a_3)$ that calculates a polynomial of degree 3 with variable coefficients a_0 to a_3 .
 - ▶ To calculate the power function you need to *import math* at the beginning of the script
 - ▶ Calculating x^3 is then done via *math.pow(x, 3)*
3. Calculate $g(x)$ for all integer values from 0 to 20 with coefficients of your choice and store the result in a list.
 - ▶ Define an empty list variable
 - ▶ Run a for loop for an appropriate number of times
 - ▶ During each loop calculate $g(x)$ and append the result to your list

Tasks continued

4. Write a script that writes down the list to a file:

- ▶ Start by opening the file
- ▶ First write "Coefficients:\n" to the file to create the first line
- ▶ Write your coefficients in the second line separated by commas
- ▶ Write "Values:" to the next line
- ▶ Run a loop through your list and in each loop write down x and the function value $g(x)$ stored in the list

File Content Sketch:

Coefficients:

$a_3,$ $a_2,$ $a_1,$ a_0

Values:

0, $g(0)$

1, $g(1)$

$\vdots$

19, $g(19)$

20, $g(20)$