Lecture 2
Functions in Math and Programming

Jan Tekülve

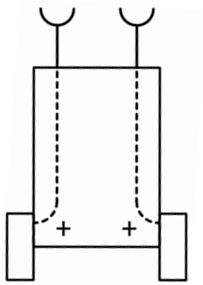
jan.tekuelve@ini.rub.de

Computer Science and Mathematics
Preparatory Course

24.09.2018

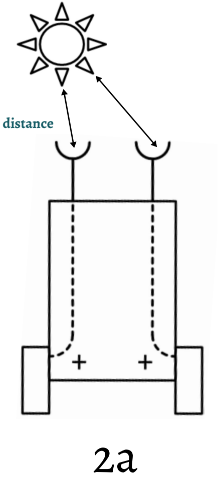
Overview

Functions in Braitenberg Vehicles

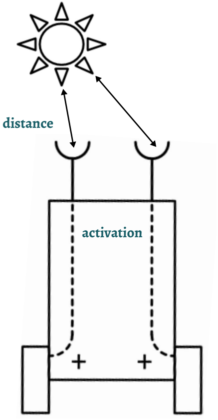


2a

Functions in Braitenberg Vehicles

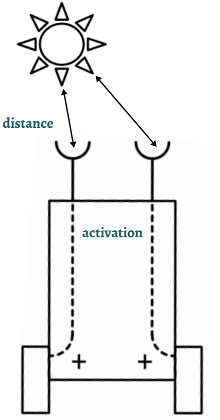


Functions in Braitenberg Vehicles

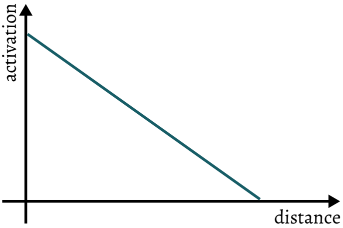


2a

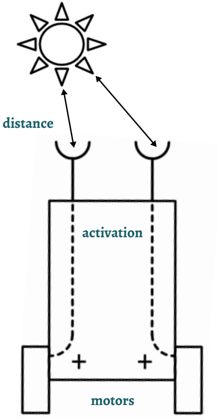
Functions in Braitenberg Vehicles



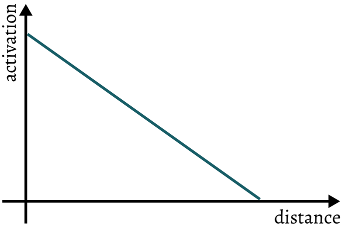
2a



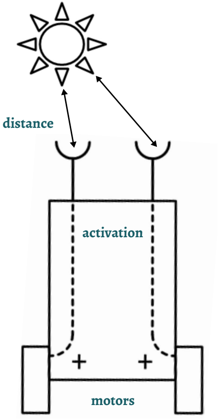
Functions in Braitenberg Vehicles



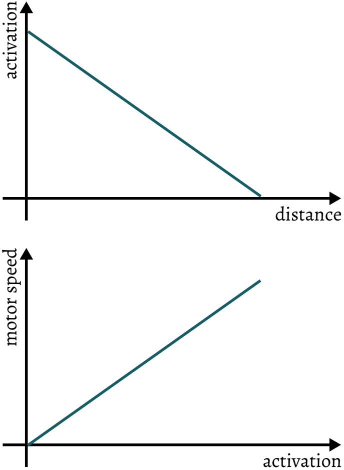
2a



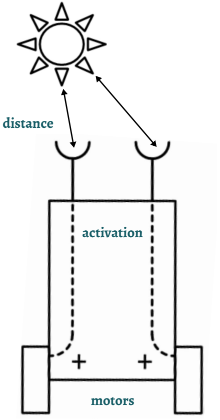
Functions in Braitenberg Vehicles



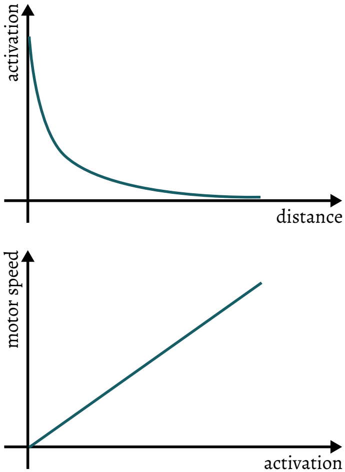
2a



Functions in Braitenberg Vehicles



2a



Function Definition

Function

X and Y are two non-empty sets.
A **function** $f : X \to Y$, assigns each element $x \in X$ exactly one element $y \in Y$.

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

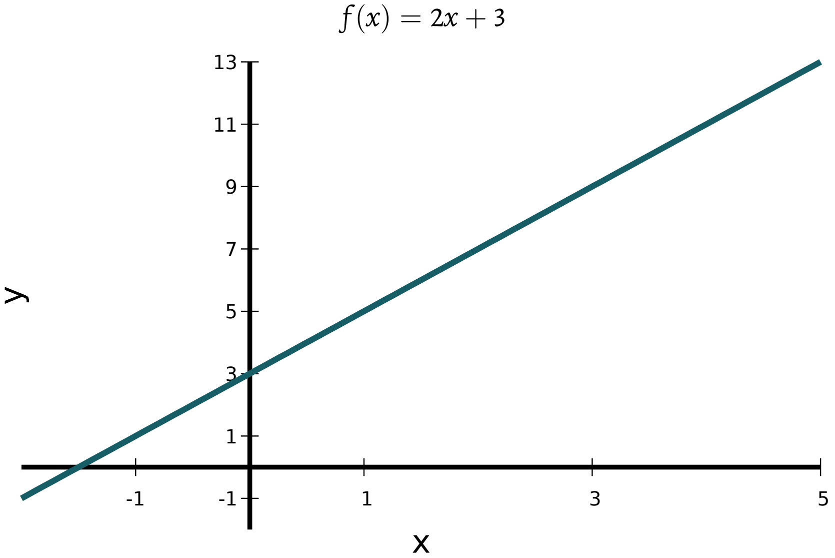
- x is called the **function argument**
- y is called the **function value**
- X is called the **domain of a function**
- Y is called the **codomain of a function**
- The **image** W of $f(x)$ are all values in Y that can be assumed

Function Examples

- Sample Functions and Domains
  - $f : \mathbb{R} \to \mathbb{R}, f(x) = x^2, X = \mathbb{R}, Y = \mathbb{R}, W = \mathbb{R}^+$
  - $f : \mathbb{R} \to \mathbb{R}^+, f(x) = x^2, X = \mathbb{R}, Y = \mathbb{R}^+, W = \mathbb{R}^+$
  - $f : \mathbb{R} \to \mathbb{R}, f(x) = 2x, X = \mathbb{R}, Y = \mathbb{R}, W = \mathbb{R}$
- Tabular Interpretation of: $f(x) = 2x + 3$

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

Plotting Functions

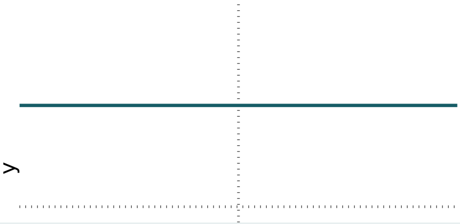


Injective and Surjective

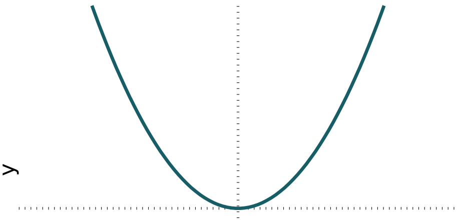
- An image f is **injective**, if two different elements $x_1 \neq x_2$ are always projected to two different elements $y_1 \neq y_2$
- An image f is **surjective**, if for each element $y \in Y$ one $x \in X$ exists, such that $y = f(x)$

Non-injective,non-sujective Functions

$f(x) = 2$



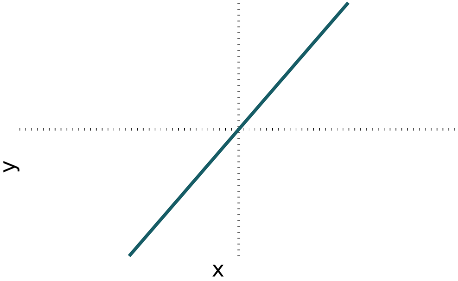
$f(x) = x^2$



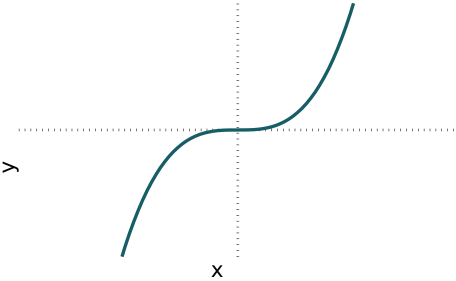
Bijective Functions

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

$f(x) = 4x$



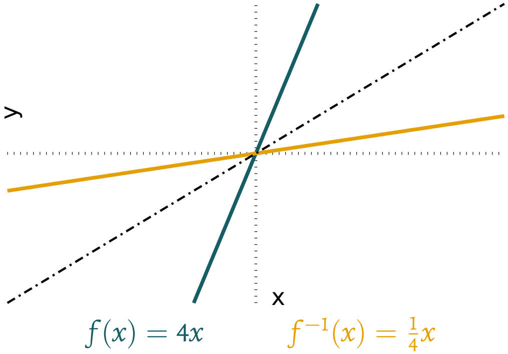
$f(x) = x^3$



Inverse Function

Definition

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



Monotonicity

Definition

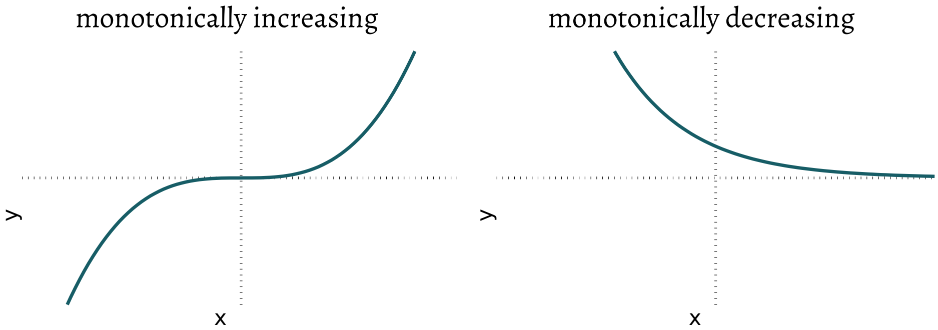
- A function $f : \mathbb{R} \to \mathbb{R}$ is called **monotonically increasing**, if for all x_1, x_2 order is preserved by applying f:

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

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

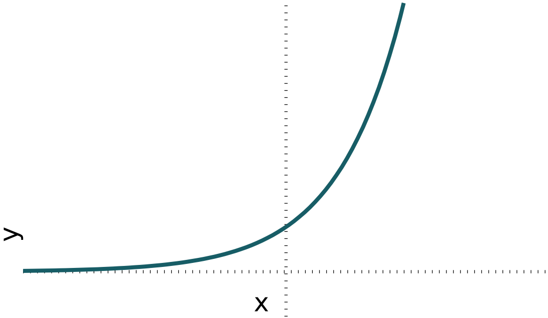
$$x_1 \leq x_2 \Rightarrow f(x_1) \geq f(x_2)$$

Monoticity Examples



Function Translation

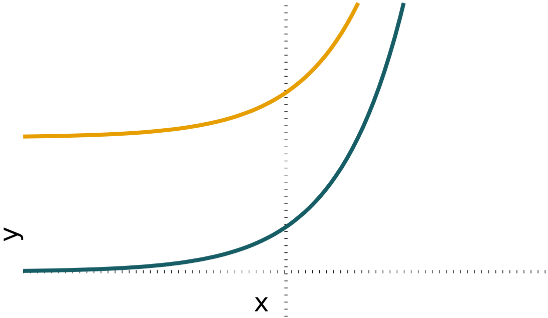
- Translation in y-direction: $\hat{f}(x) = f(x) + b$
- Translation in x-direction: $\hat{f}(x) = f(x - a)$



$f(x) = e^x$

Function Translation

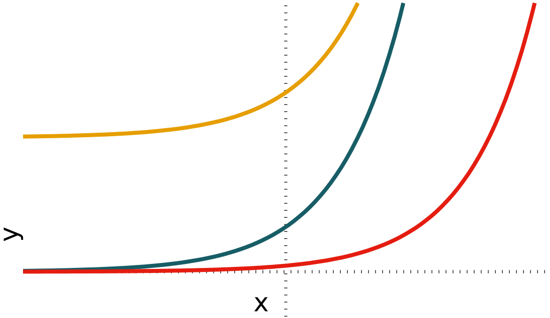
- Translation in y-direction: $\hat{f}(x) = f(x) + b$
- Translation in x-direction: $\hat{f}(x) = f(x - a)$



$f(x) = e^x$ $g(x) = e^x + 3$

Function Translation

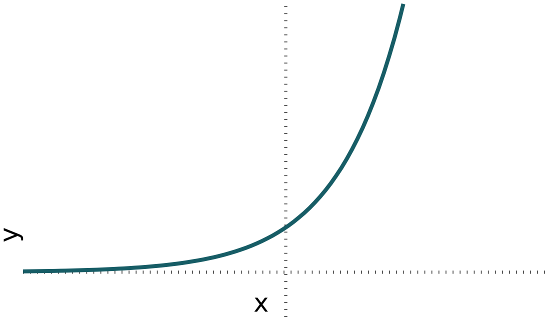
- Translation in y-direction: $\hat{f}(x) = f(x) + b$
- Translation in x-direction: $\hat{f}(x) = f(x - a)$



$f(x) = e^x$ $\quad$ $g(x) = e^x + 3$ $\quad$ $h(x) = e^{x-2}$

Function Stretching and Compression

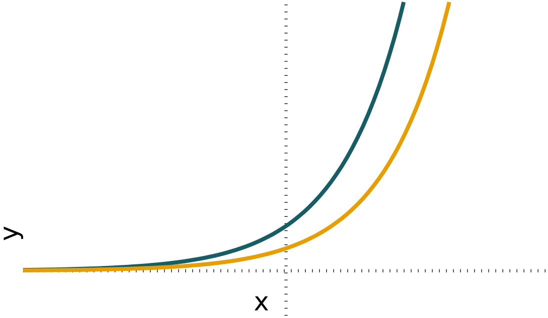
- Stretching/Compression in ***y*-direction**: $\hat{f}(x) = \boldsymbol{d}f(x), d > 0$
- Stretching/Compression in ***x*-direction**: $\hat{f}(x) = f(\boldsymbol{c}x), c > 0$



$f(x) = e^x$

Function Stretching and Compression

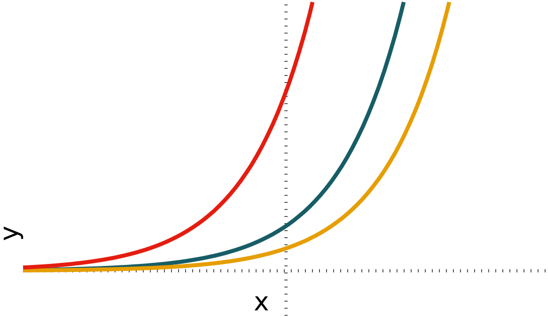
- Stretching/Compression in ***y*-direction**: $\hat{f}(x) = \boldsymbol{d}f(x), d > 0$
- Stretching/Compression in ***x*-direction**: $\hat{f}(x) = f(\boldsymbol{c}x), c > 0$



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

Function Stretching and Compression

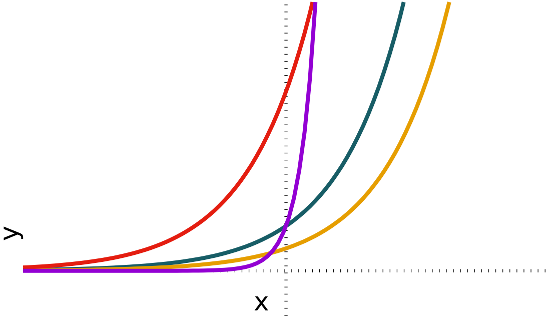
- Stretching/Compression in ***y*-direction**: $\hat{f}(x) = \boldsymbol{d}f(x), d > 0$
- Stretching/Compression in ***x*-direction**: $\hat{f}(x) = f(\boldsymbol{c}x), c > 0$



$f(x) = e^x$ $g(x) = \frac{1}{2}e^x$ $h(x) = 4e^x$

Function Stretching and Compression

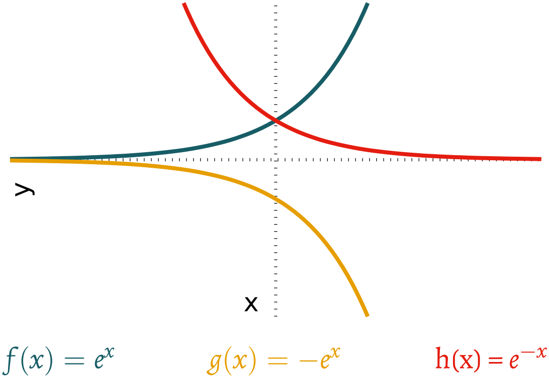
- Stretching/Compression in ***y*-direction**: $\hat{f}(x) = \boldsymbol{d}f(x), d > 0$
- Stretching/Compression in ***x*-direction**: $\hat{f}(x) = f(\boldsymbol{c}x), c > 0$



$f(x) = e^x$ $g(x) = \frac{1}{2}e^x$ $h(x) = 4e^x$ $j(x) = e^{4x}$

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**

$$y = mx + b$$

- **Power Functions**

$$y = ax^n$$

Function Types

- **Linear Functions**

$$y = mx + b$$

- **Power Functions**

$$y = ax^n$$

- **Polynomial Functions**

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

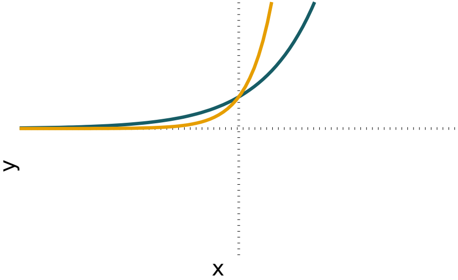
$$y = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \ldots a_n x^n$$

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

Exponentials Functions

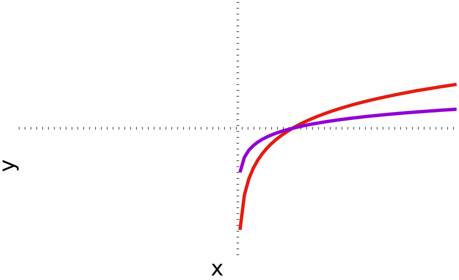
Exponential Functions

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



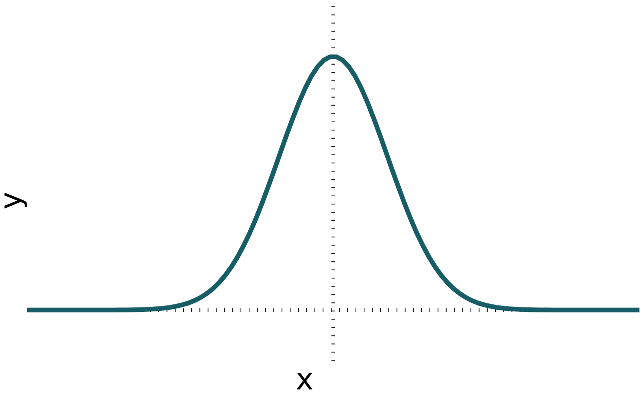
Logarithmic Functions

$h(x) = ln(x)$ $j(x) = log_{10}(x)$



The Gaussian Function

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

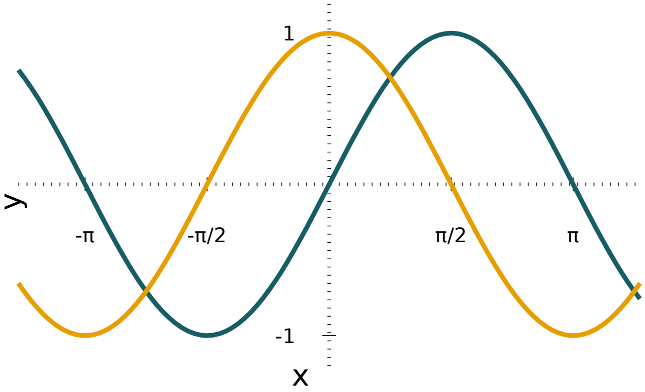


The Gaussian Function

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

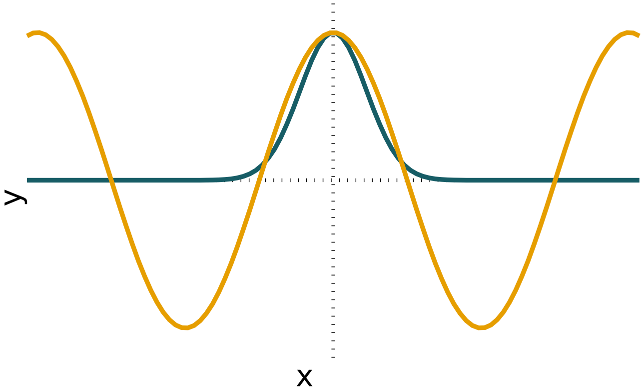
Trigonometric Functions

$f(x) = sin(x)$ $g(x) = cos(x)$



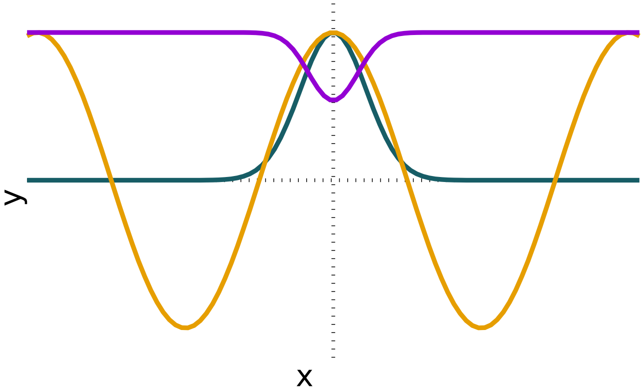
Chaining Functions

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



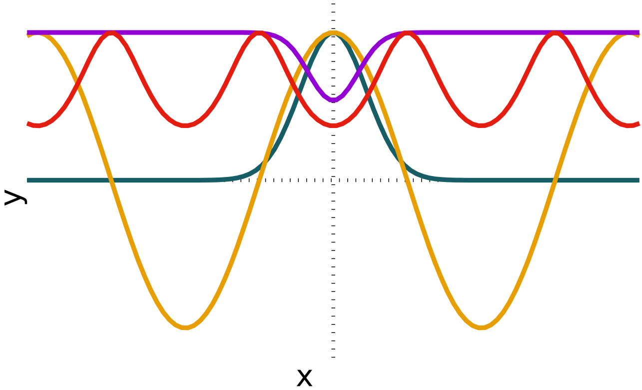
Chaining Functions

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



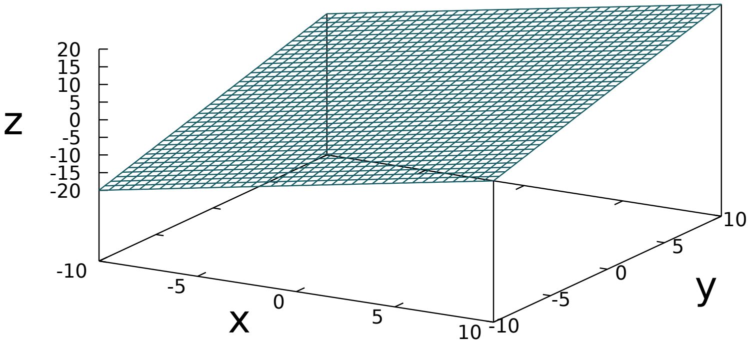
Chaining Functions

$f(x) = e^{-x^2}$ $g(x) = cos(x)$ $h(x) = g(f(x))$ $j(x) = f(g(x))$



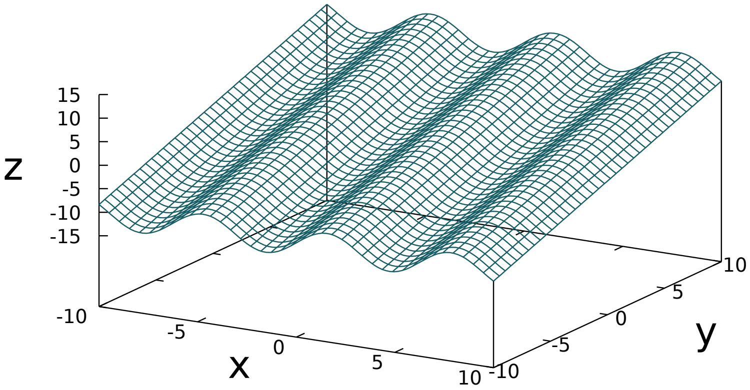
Multiple Function Arguments

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



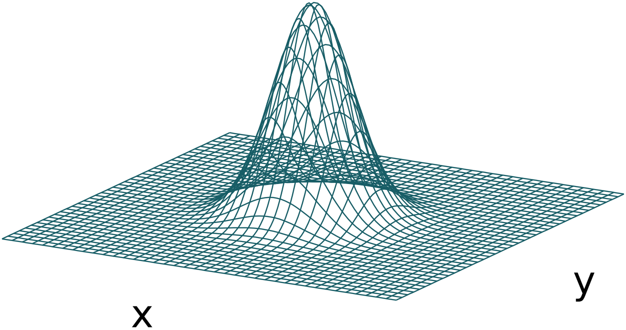
Multiple Function Arguments

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



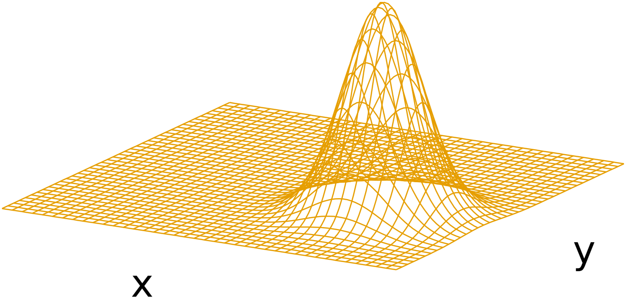
Multiple Function Arguments

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



Multiple Function Arguments

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



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 x*x
```

- 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 subract(minuend,subtrahend):
  return minuend-subtrahend
```

- Advanced Function calling:

```
result = subract(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 subract(minuend,subtrahend,console=False):
  if console:
    print(str(minuend-subtrahend))
  return minuend-subtrahend
```

- Optional arguments may be ignored while calling:

```
result = subract(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:

```
fileObj.close()
```

Tasks

1. Write a function that calculates $f(x) = -x$ for a single input value. Apply the function to an input of your choice and print the result on the console.
2. Change the above function that it calculates $f(x)$ for a single x-value, where $f(x)$ is a polynomial of degree 4 with variable coefficients a_0 to a_4.
3. Calculate $f(x)$ for all integer values from -10 to 10 with coefficients of your choice and store the result in a list.
4. Write the list to a file with the following format:

$$\begin{array}{ccc} -10 & , & f(-10) \\ -9 & , & f(-9) \\ & \vdots & \\ 9 & , & f(9) \\ 10 & , & f(10) \end{array}$$