# Week 8

## Introduction to Programming and Numerical Analysis

Ebener, Luca

UNIVERSITY OF COPENHAGEN

# Overview

- Functions
- Floating point numbers
- Classes
- NumPy
- Work on problem set 0 and DataCamp

# Functions

## Explanation

- Functions in Python are reusable blocks of code that perform a specific task
- They help organize code, promote code reuse, and improve readability

## Use functions...

- ...to modularize code
- ...for tasks that are performed repeatedly

## Tips

- Keep functions short and focused on a single task
- Use meaningful function names that describe their purpose
- Document functions with docstrings to explain their usage and behavior

# Function Examples

```python
def greet(name):
    """ Greets a person by name. """
    print("Hello, ", name)

add = lambda x, y: x + y # sums input arguments

def sum_values(*args):
    """ Calculates the sum of multiple values. """
    return sum(args)

def print_info(**kwargs):
    """ Prints key-value pairs. """
    for key, value in kwargs.items():
        print(key + ":", value)
```

# Floating point number approximation

## Approximation in computers

- Floating point numbers in computers are approximations of real numbers
- Computers typically use base 2 (binary) to represent floating point numbers

## Representation formula

- Floating point number: $(-1)^{\mathit{significant}} * (1 + \mathit{mantissa}) * 2^{\mathit{exponent}}$
- 's' is the sign bit, 'mantissa' is the fractional part, and 'exponent' is the power of 2

## Examples

- Exact representation for $0.5$ has significant-exponent-mantissa:
  $0 - 01111111110 - 000000000000000000000000000000000000000000000000$
- Approximated representation for $0.1$:
  $0 - 01111111011 - 100110011001100110011001100110011001100110011001$

# Working with floating point numbers

```
# Check if a floating point number is infinity
print(np.isinf(float('inf'))) # Output: True

# Check if two floating point numbers are close
print(np.isclose(0.1 + 0.2, 0.3)) # Output: True

# Rounding floating point numbers
print(round(0.12345 + 0.2222, 2)) # Output: 0.35

# Check if floating point number is not a number
print(np.isnan(0.1)) # Output: False
```

# Basics of classes and object-orientated programming (OOP)

**Introduction to classes**

- Classes allow to create new types of objects

**Key concepts**

- Encapsulation: Bundling data and methods together while controlling access
- Abstraction: Hiding implementation details and showing only essential features
- Inheritance: Creating new classes based on existing ones, promoting code reuse and establishing a hierarchical relationship
- Polymorphism: Treating objects of different classes through a common interface, allowing for flexibility and extensibility in code

# Example class (1)

```python
class Rectangle:
    def __init__(self, width, height):
        self.width = width
        self.height = height

    def __str__(self):
        return f"Width={self.width}, height={self.height}"

    def __getattr__(self, attr):
        if attr == 'area':
            return self.width * self.height
        else:
            raise AttributeError(f"'Rectangle' object has no attribute '{attr}'")
```

# Example class (2)

```python
# create instances of Rectangle class
rect = Rectangle(5,4)

# print out the rectangle
print(rect) # Output: Width=5, height=4

# Accessing dynamically calculated attribute
print(rect.area) # Output: 20
```

# Introduction to NumPy

**What is NumPy?**

- NumPy is a fundamental package for scientific computing in Python
- It provides powerful tools for working with arrays, matrices, and mathematical functions

**Why use NumPy?**

- Numerical operations and vectorized computations are faster than on Python lists
- NumPy integrates seamlessly with other scientific computing libraries in Python such as SciPy, pandas, and Matplotlib

**NumPy arrays**

- NumPy arrays are multidimensional data structures of elements of the same data type
- They are more memory efficient than Python lists

# Broadcasting with NumPy arrays

**Broadcasting...**

- ...allows arrays with different shapes to be combined in arithmetic operations
- ...simplifies the syntax for performing element-wise operations on arrays of different shapes

**How broadcasting works**

- If the arrays have different numbers of dimensions, the shape of the smaller array is padded with ones on its left side
- The size of each dimension of the arrays must be compatible, meaning it must be equal or one of them must be 1

# Example broadcasting

```
arr1 = np.array([1,2,3])
scalar = 2
result = arr1 + scalar # scalar is broadcasted to [2,2,2]
print(np.shape(arr1 + scalar) # Output: (3,)

arr2 = np.ones((3,1,3))
print(np.shape(arr2*arr1)) # Output: (3, 1, 3)

arr3 = arr1[:,np.newaxis]
print(np.shape(arr3)) # Output: (3, 1)
print(np.shape(arr2 * arr3)) # Output: (3, 3, 3)
```