



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

```
1 def greet(name):  
2     """ Greets a person by name. """  
3     print("Hello, ", name)  
4  
5 add = lambda x, y: x + y # sums input arguments  
6  
7 def sum_values(*args):  
8     """ Calculates the sum of multiple values. """  
9     return sum(args)  
10  
11 def print_info(**kwargs):  
12     """ Prints key-value pairs. """  
13     for key, value in kwargs.items():  
14         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: $(-1)^{\text{significant}} * (1 + \text{mantissa}) * 2^{\text{exponent}}$
- '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 – 0111111110 – 00
- Approximated representation for 0.1:
0 – 01111111011 – 100110011001100110011001100110011001100110011001

Working with floating point numbers

```
1 # Check if a floating point number is infinity
2 print(np.isinf(float('inf'))) # Output: True
3
4 # Check if two floating point numbers are close
5 print(np.isclose(0.1 + 0.2, 0.3)) # Output: True
6
7 # Rounding floating point numbers
8 print(round(0.12345 + 0.2222, 2)) # Output: 0.35
9
10 # Check if floating point number is not a number
11 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 attributes and methods together within a single unit
- Abstraction: Controlling the visibility of a classes features
- Inheritance: Defining new classes based on existing classes
- Polymorphism: The ability to use a common interface for objects of different classes

Use classes to...

- ...organize and modularize code, promoting code reuse and maintainability
- ...represent complex data structures and relationships in applications

Example class (1)

```
1 class Rectangle:
2     def __init__(self, width, height):
3         self.width = width
4         self.height = height
5
6     def __str__(self):
7         return f"Width={self.width}, height={self.height}"
8
9     def __getattr__(self, attr):
10        if attr == 'area':
11            return self.width * self.height
12        else:
13            raise AttributeError(f"'Rectangle' object has no attribute '{attr}'")
```


Example class (2)

```
1 # create instances of Rectangle class
2 rect = Rectangle(5,4)
3
4 # print out the rectangle
5 print(rect) # Output: Width=5, height=4
6
7 # Accessing dynamically calculated attribute
8 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
- NumPy's core data structure is the ndarray (N-dimensional array)

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

- NumPy automatically broadcasts arrays to have compatible shapes by adding singleton dimensions or replicating elements along existing dimensions
- Broadcasting follows a set of rules to determine how arrays are combined
 - 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

```
1 arr1 = np.array([1,2,3])
2 scalar = 2
3 result = arr1 + scalar # scalar is broadcasted to [2,2,2]
4 print(np.shape(arr1 + scalar) # Output: (3,)
5
6 arr2 = np.ones((3,1,3))
7 print(np.shape(arr2*arr1)) # Output: (3, 1, 3)
8
9 arr3 = arr1[:,np.newaxis]
10 print(np.shape(arr3)) # Output: (3, 1)
11 print(np.shape(arr2 * arr3)) # Output: (3, 3, 3)
```