Introduction to Scientific Computing

with Python

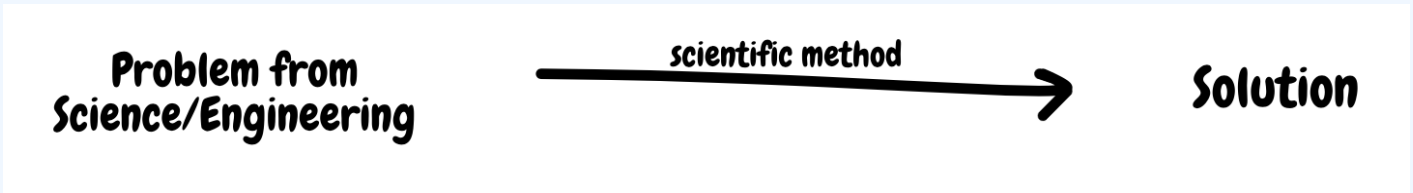
Utku Kaya

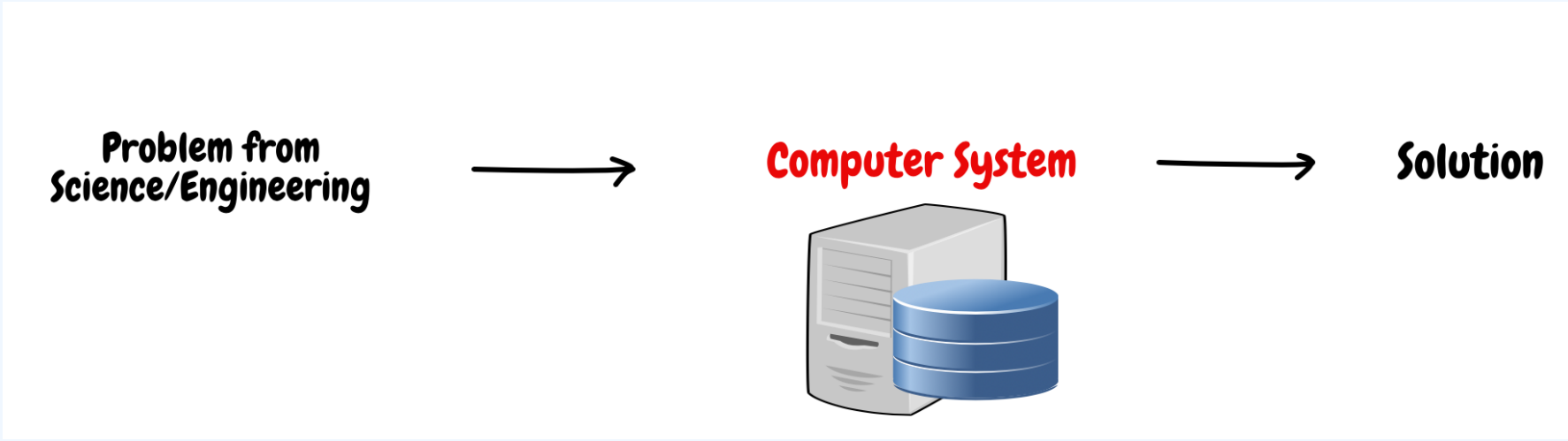
Department of Mathematics
University of Kiel, Germany

24-28 02 2020

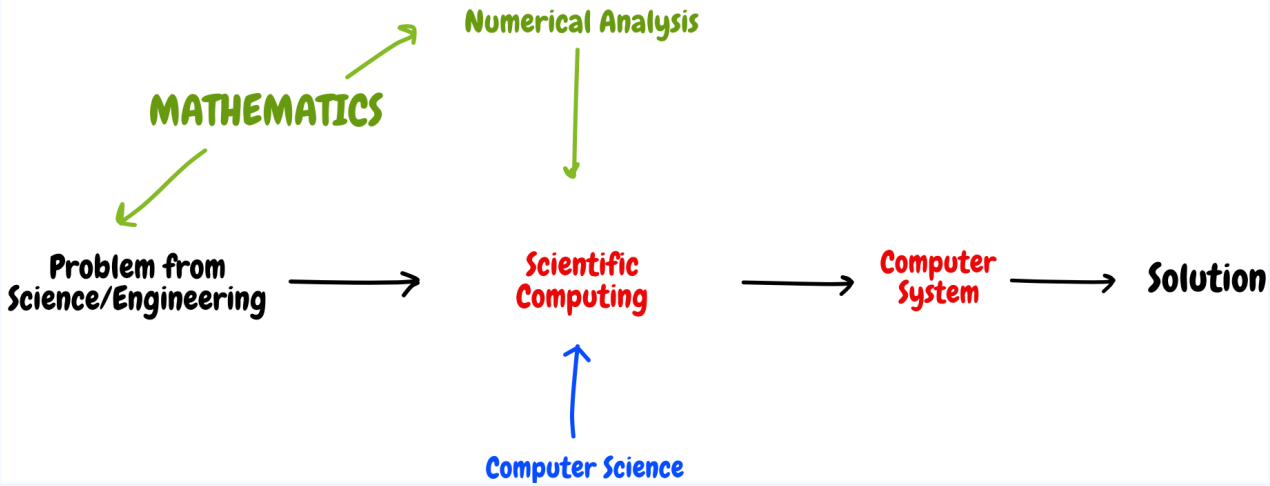
Getting started

What is scientific computing?


Problem from
Science/Engineering
scientific method
Solution


Problem from
Science/Engineering
Computer System
Solution


Problem from Science/Engineering
Scientific Computing
Computer System
Solution



Scientific computing covers the collection of tools, techniques and theories required to solve on a computer mathematical models of problems in science and engineering. [1]

---

[1] Golub, Ortega, 2014. Scientific Computing: An Introduction with Parallel Computing.

Getting started

What is Python?

Python is an interpreted, high-level, general-purpose programming language.

Released in 1991, Guido van Rossum (Netherlands).

The Zen of Python, by Tim Peters:

- Beautiful is better than ugly.
- Explicit is better than implicit.
- Simple is better than complex.
- Complex is better than complicated.
- Readability counts.
- . . .

Course Overview

Course Overview

Today: Introduction, syntax, variables, data types, operators, built-in mathematical functions,

25.02: Functions, conditional statements, recursion, classes/objects, iterators, reading & writing files,

26.02: NumPy introduction, numerical differentiation and integration,

27.02: Solving systems of linear equations,

28.02: Example: a finite difference code for Poisson equation in 2D.

First lines of code

```
>>> print("Hello, Peru!")
```

The >>> symbol is called a prompt.

```
>>> person = input('Enter your name: ')
>>> print('Hello ', person, '!')
>>> print('Hello ' + person + '!')
>>> print('Hello, {}!'.format(person))
>>> print('Hello ', person, '!', sep='')
```

First lines of code

Always put a space before and after every binary operator!

```
>>> num = 2
```

`num` is assigned `2`.

```
>>> 2 = num
```

Assignment is not symmetric!

First lines of code

```
>>> num = 2
>>> print(num)
>>> print("num␣=",num)
>>> print( type(num))
>>> print("The␣number␣", num, "␣is␣of␣type", type(num))
>>> num = 3.0
>>> print(isinstance(num, int))
>>> num = 3 + 5j
>>> print(isinstance(num, complex))
>>> num = float(-1)
>>> print(num)
```

Note: a float type number can have precision up to 15 decimal places.

Reassigning to Variables

```
>>> num = 2
>>> num1 = 3 * num
>>> num1
>>> num = 4
>>> num1
```

Memory Addresses

Python remembers and reuses some objects.

```
>>> id(3)
>>> help(id)
>>> x = 3
>>> id(x)
>>> id(3.1)
>>> id(300)
```

Strings

```
>>> txt = "Some␣text"
>>> len(txt)
>>> len(txt) == 12
>>> txt = """This is a string
... in two lines"""
>>> print(txt)
>>> print(type(txt))
>>> first_2_chars = txt[:2]
>>> print(first_2_chars)
>>> last_2_chars = txt[-2:]
>>> print(last_2_chars)
```

Multiple assignments

Following lines are equivalent.

```
>>> x, y = 10, 20
>>> x, y = (10, 20)
>>> (x, y) = 10, 20
>>> (x, y) = (10, 20)
```

Multiple assignments work for strings, too.

```
>>> x, y = 'je'
>>> x
>>> y
>>> x = y = 'je'
>>> print(x, y)
>>> print(x * 3)
```

Arithmetic Operators

```
>>> x, y = -3, 4
>>> x + y      # Addition
>>> x - y      # Subtraction
>>> x * y      # Multiplication
>>> x / y      # Division
>>> x % y      # Modulus
>>> x ** y     # Exponentiation
>>> x // y     # Floor division
>>> -3 ** 2
>>> -(3 ** 2)
>>> (-3) ** 2
>>> print('{0} + {1} = {2}'.format(x, y, x + y))
```

It holds `x = (x // y) * y + (x % y)`.

Numeric Precision

```
>>> 2 / 3 + 1
>>> 5 / 3
>>> .1 + .1 + .1 == .3
```

Bruce M. Bush, *Programming with the Perils:*

There are no easy answers. It is the nature of binary floating-point to behave the way I have described. In order to take advantage of the power of computer floating-point, you need to know its limitations and work within them.

Assignment Operators

```
>>> x = 13
>>> x += 5
>>> x -= 3
>>> x *= 2
>>> x /= 2
>>> x %= 3
>>> x //= 3
>>> x **= 3
>>> x &= 3
>>> x = 3
```

Assignment Operators

```
>>> x = 2
>>> x *= 3 + 4
>>> x
```

Multiple Lines

```
>>> (2
... + 3)
>>> 2 + \
... 3
```

Comparison Operators

```
>>> x, y = 13, 9
>>> x == y
>>> x != y
>>> x > y
>>> x >= y
>>> x < y
>>> x <= y
```

Logical Operators and Identity Operators

```
>>> x, y = 13, 9
>>> x < 5
>>> y < 10
>>> x < 5 and y < 10
>>> x < 5 or y < 10
>>> not(x < 5 and y < 10)
>>> x is y
>>> x is 13
>>> x is not 4
```

Membership Operators

```
>>> 'abc' in 'abcd'
>>> 'abc' in 'Abcd'
>>> 'abc' not in 'Abcd'
```

Built-in Functions

```
>>> abs(-9)
>>> abs(3 - 5)
>>> pow(2, 3)
>>> pow(2, abs(-3))
>>> pow(2, abs(round(-3.6)))
>>> help(abs)
>>> round(3.141592653, 2)
```

Built-in math functions

Tell Python that you want to use functions in module math:

```
>>> import math
>>> help(math)
>>> math.fabs(-2.1)
>>> x = 2*math.pi
>>> math.sin(x)
>>> math.sqrt(9)
```

Built-in math functions

If you need only several functions from `math` module:

```
>>> from math import pi, sqrt
>>> help(sqrt)
>>> sqrt(9)
```

Thanks for your attention!