Statistical Methods in Particle Physics

Selected topic 3:
Practical tips

Heidelberg University, WS 2020/21

Klaus Reygers (lectures)
Rainer Stamen, Martin Völkl (tutorials)

Numerics (1)

A double-precision floiting point number is stored in a total of 64 bits (1 sign bit, 11-bit exponent, 52-bit mantissa). This corresponds to about 15 decimal digits of precision. Any calculation resulting in a higher precision is subject to rounding errors. An upper bound of the relative error due to rounding can be obtain in python like this:

```
import sys
sys.float_info.epsilon
```

```
2.220446049250313e-16
```

Comparing floating point numbers

```
0.3**2 == 0.09
```

```
True
```

```
0.2**2 == 0.04
```

```
False
```

```
import numpy as np
np.isclose((0.2)**2, 0.04)
```

```
True
```

See notebook practical_tips.ipynb on the lecture web page

Numerics (2)

Range of floating point numbers

```
print(sys.float_info.min, sys.float_info.max)
```

```
2.2250738585072014e-308 1.7976931348623157e+308
```

In numpy, results that are larger than the maximum range a set to `inf` or `-inf`, respectively.

```
a = np.exp(1000)
print(a)
```

```
inf
```

```
<ipython-input-29-7891d9d166db>:1: RuntimeWarning: overflow encountered in exp
  a = np.exp(1000)
```

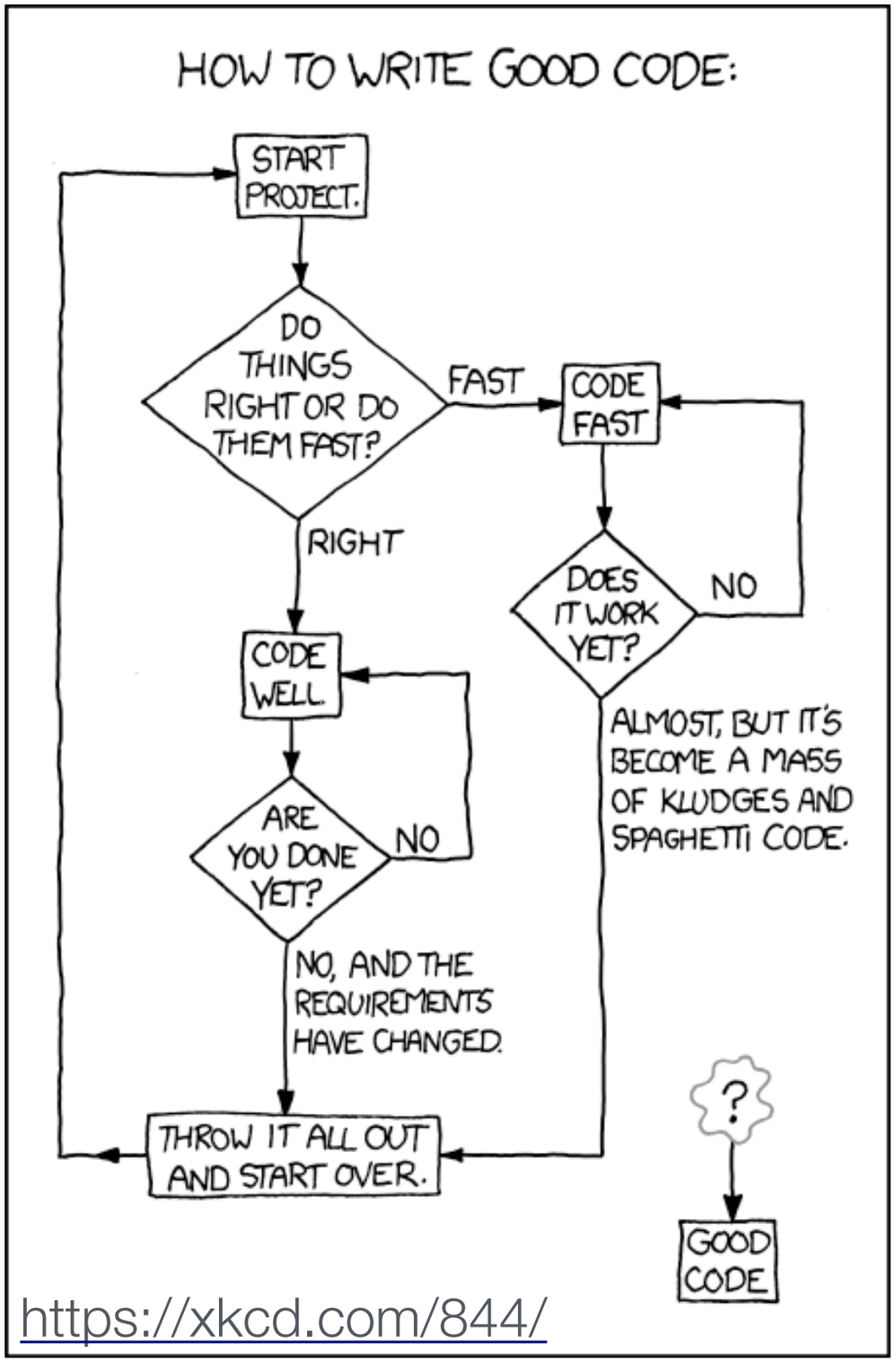
Undefined operations result in the floating point value `nan`

```
print(np.sqrt(-1), np.log(-1), np.arcsin(1.1))
```

```
nan nan nan
```

```
<ipython-input-9-e46aa987836e>:1: RuntimeWarning: invalid value encountered in sqrt
  print(np.sqrt(-1), np.log(-1), np.arcsin(1.1))
<ipython-input-9-e46aa987836e>:1: RuntimeWarning: invalid value encountered in log
  print(np.sqrt(-1), np.log(-1), np.arcsin(1.1))
<ipython-input-9-e46aa987836e>:1: RuntimeWarning: invalid value encountered in arcsin
  print(np.sqrt(-1), np.log(-1), np.arcsin(1.1))
```

Programming and “clean code”



https://xkcd.com/844/

A key issue in large software projects is to manage complexity. The total cost of having a mess can be large in terms of wasted time and money. From Robert C. Martin books Clean Code:

"As the mess builds, the productivity of the team continues to decrease, asymptotically approaching zero".

A few simple rule might help. The following examples were adapted from

- Jamie Bullock, Clean Code: 5 Essential Takeaways
- Esteban Solorzano, Clean Code in Python

Programming and “clean code”

Keep it short

- Function bodies should be short — hardly ever longer than 20 lines and mostly less than 10 lines
- Functions should take as few arguments as possible

Make Code Self-Documenting

"Clear and expressive code with few comments is far superior to cluttered and complex code with lots of comments. — Robert C. Martin

Not so clear:

```
// Check to see if the employee is eligible for full benefits
if ((employee.flags & HOURLY_FLAG) &&
    (employee.age > 65))
```

Better:

```
if (employee.isEligibleForFullBenefits())
```

Programming and “clean code”

Use meaningful and intention-revealing names

- Example: `int elapsedTimeInDays` is better than `int days`
- Function names should say what they do

Unit tests

- "Unit tests are typically automated tests written and run by software developers to ensure that a section of an application (known as the "unit") meets its design and behaves as intended.” (wikipedia)
- Makes sure that changes do not break your code
- Check out the unittest module in python: `import unittest`

Programming and “clean code”

From the Zen of Python

- Beautiful is better than ugly
- Simple is better than complex
- Readability counts

(try `import this`)