Numeric and Scalar Types

Robert Smallshire
@robsmallshire
rob@sixty-north.com

Presenter

Austin Bingham
@austin_bingham
austin@sixty-north.com

pluralsight
hardcore dev and IT training

python™

int

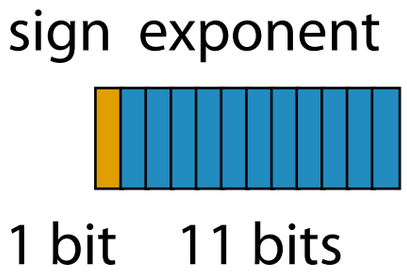
unlimited precision signed integer

float

IEEE-754 double precision (64-bit)

53 bits of binary precision

15 to 17 bits of decimal precision

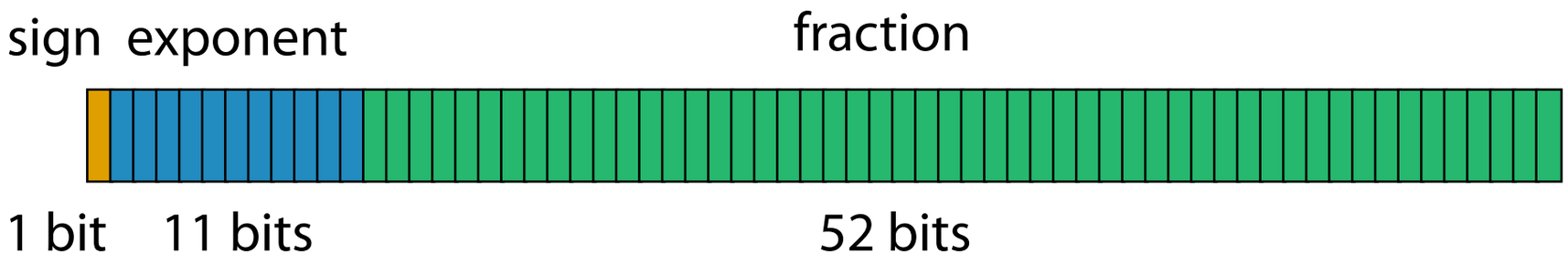


float

IEEE-754 double precision (64-bit)

53 bits of binary precision

15 to 17 bits of decimal precision



1

What Every Computer Scientist Should Know About Floating-Point Arithmetic

D

This appendix has the following organization:

David Goldberg
Journal ACM Computing Surveys (CSUR)
Volume 23 Issue 1, March 1991
Pages 5-48

The standard library module

`decimal`

containing the class

`Decimal`

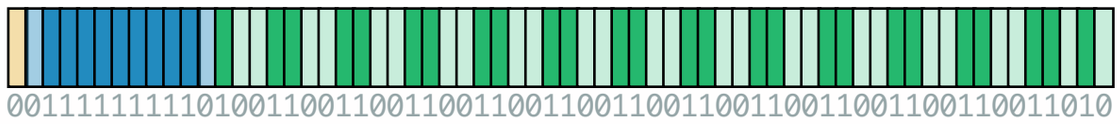
decimal floating point

configurable (although finite) precision

defaults to 28 digits of decimal precision

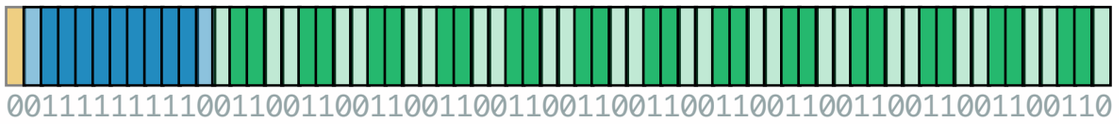
Decimal(0.8) - Decimal(0.7)

0.8



0.8000000000000004440892098500626161694526672363281 25

0.7

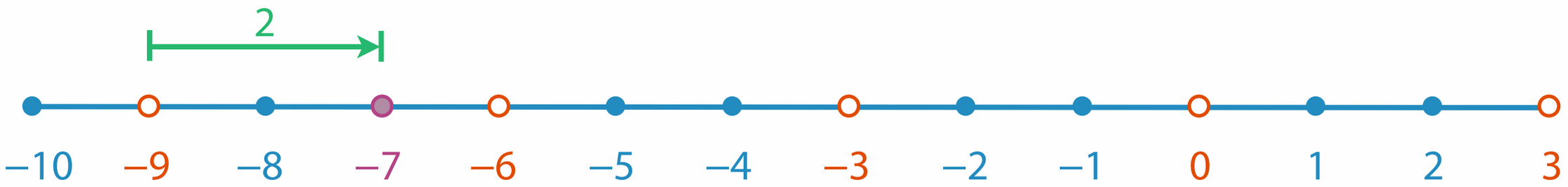


0.6999999999999999555910790149937383830547332763671875

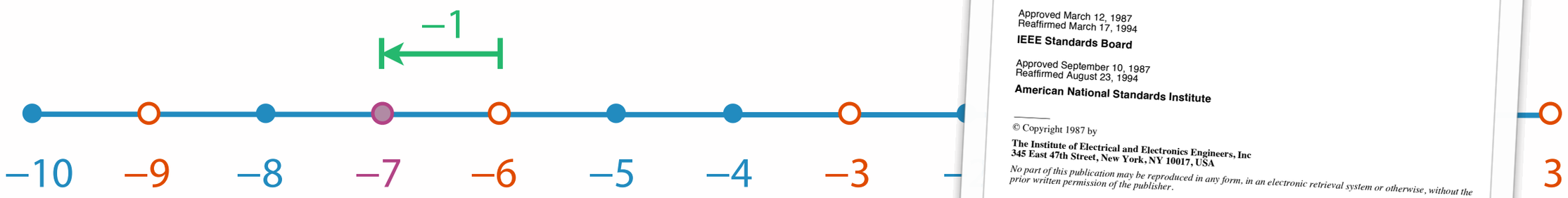
0.1000000000000000888178419700

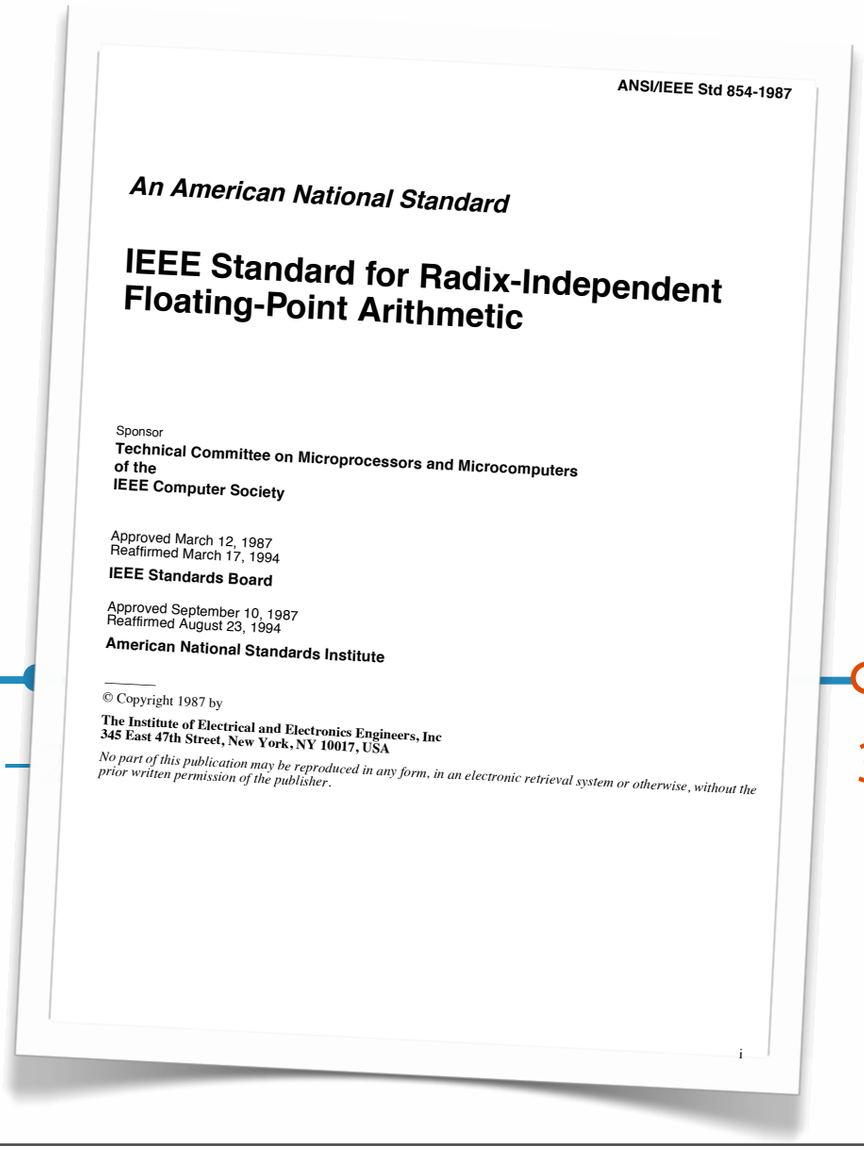
always **quote** literal fractional values

```
>>> (-7) % 3
2
```




```
>>> Decimal(-7) % Decimal(3)
Decimal('-1')
```





ANSI/IEEE Std 854-1987

An American National Standard

IEEE Standard for Radix-Independent Floating-Point Arithmetic

Sponsor
Technical Committee on Microprocessors and Microcomputers of the IEEE Computer Society

Approved March 12, 1987
Reaffirmed March 17, 1994
IEEE Standards Board

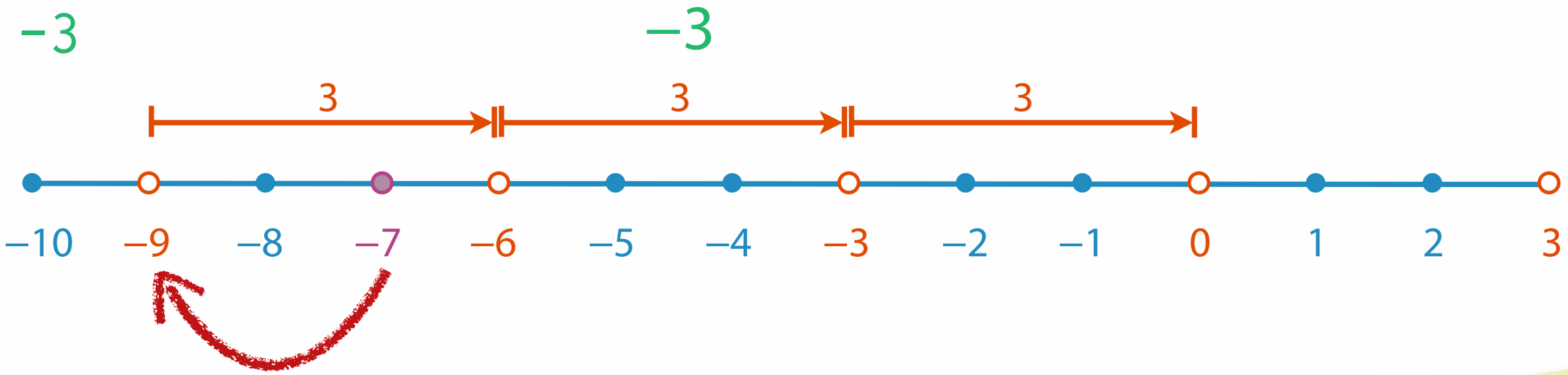
Approved September 10, 1987
Reaffirmed August 23, 1994
American National Standards Institute

This important
identity is preserved

```
x == (x // y) * y + x % y
```

so integer division and modulus
are consistent

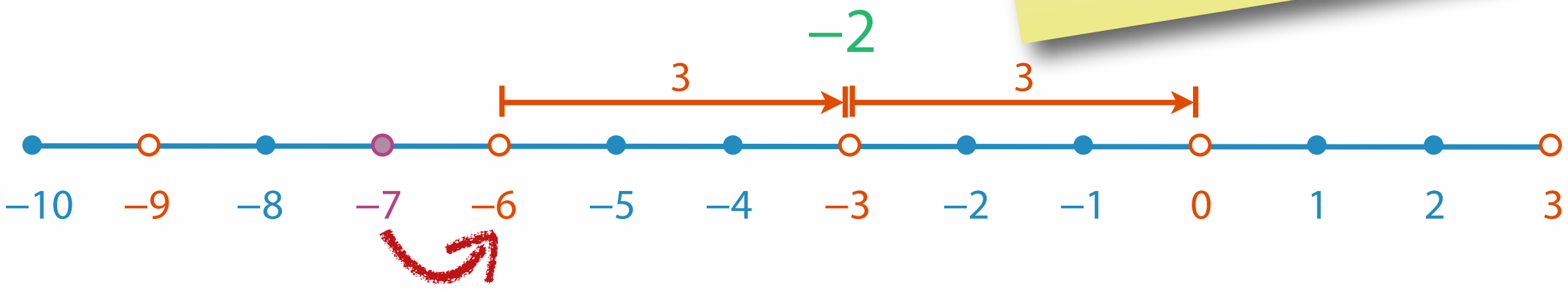
```
>>> (-7) // 3
-3
```



Largest multiple of 3 less than -7

The **floor division** operator // is a **misnomer** for Decimal

```
>>> Decimal(-7) // Decimal(3)
Decimal('-2')
```



Next multiple of 3 towards zero

python™

The standard library module

fractions

containing the class

Fraction

for rational numbers

numerator

denominator

2/3

4/5

denominator cannot be zero

The built-in type

complex

for complex numbers

Python uses the electrical engineering notation for **imaginary numbers**

$$j = \sqrt{-1}$$

complex construction

string argument **may** have parentheses

but **must not** contain spaces

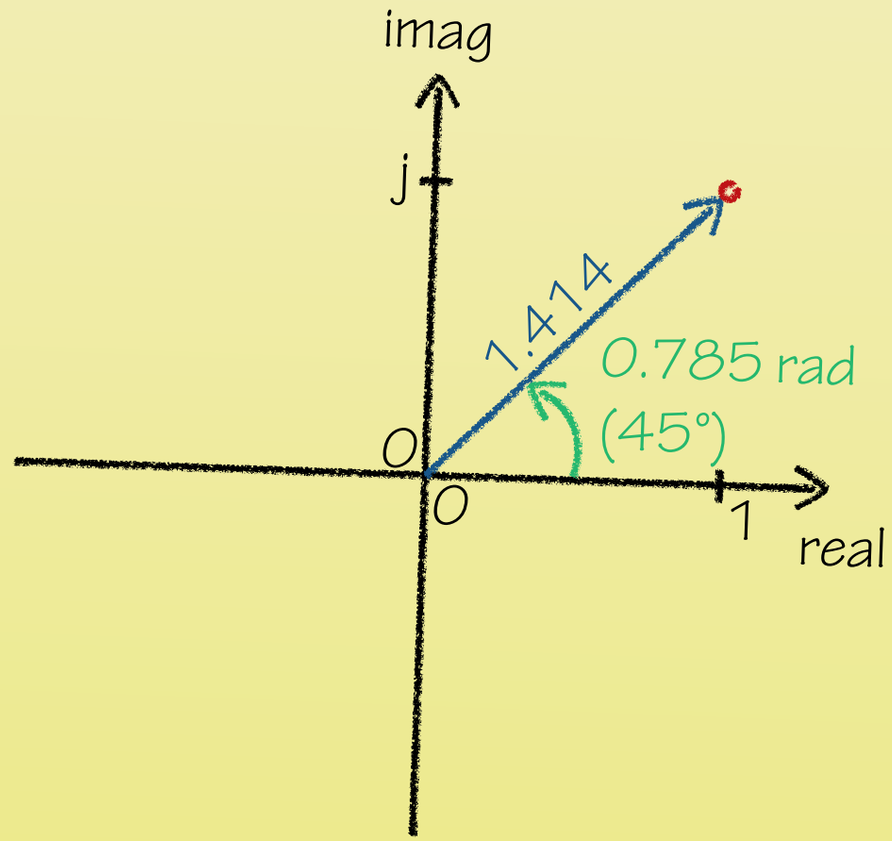
`cmath`

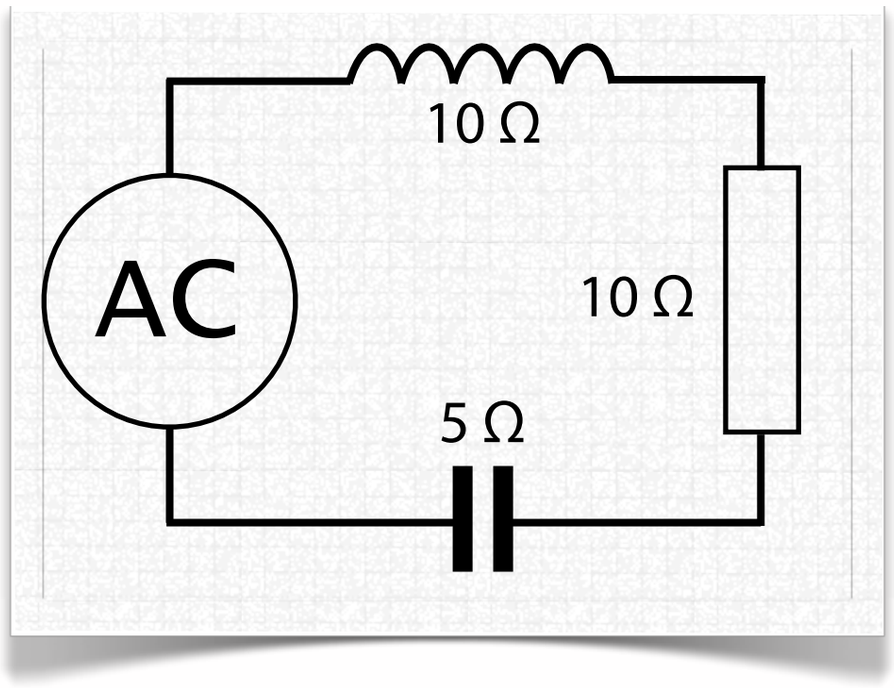
Standard Library module contains

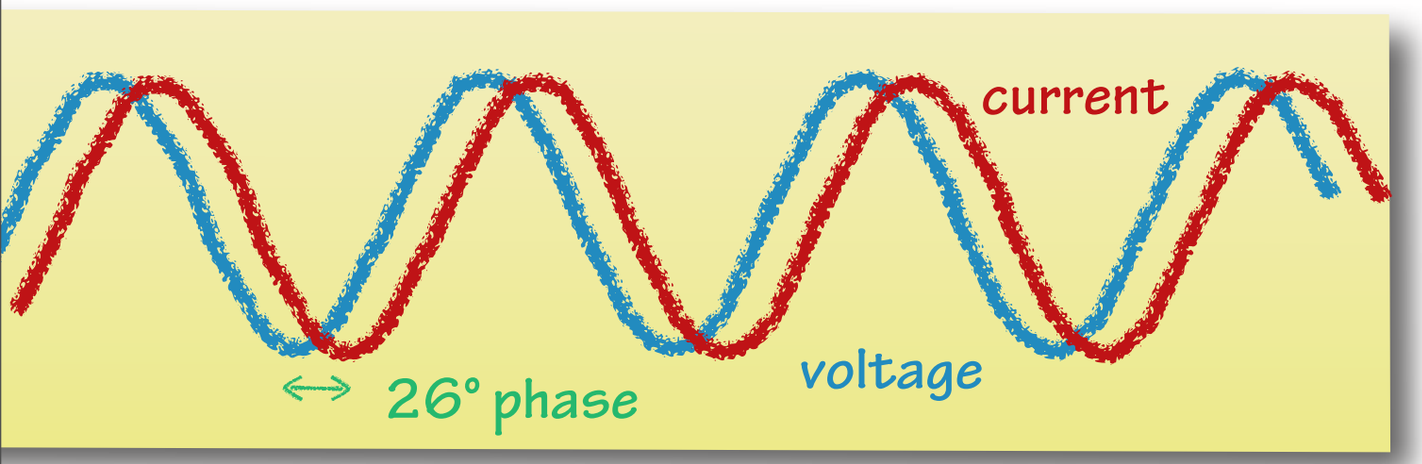
complex equivalents of `math`

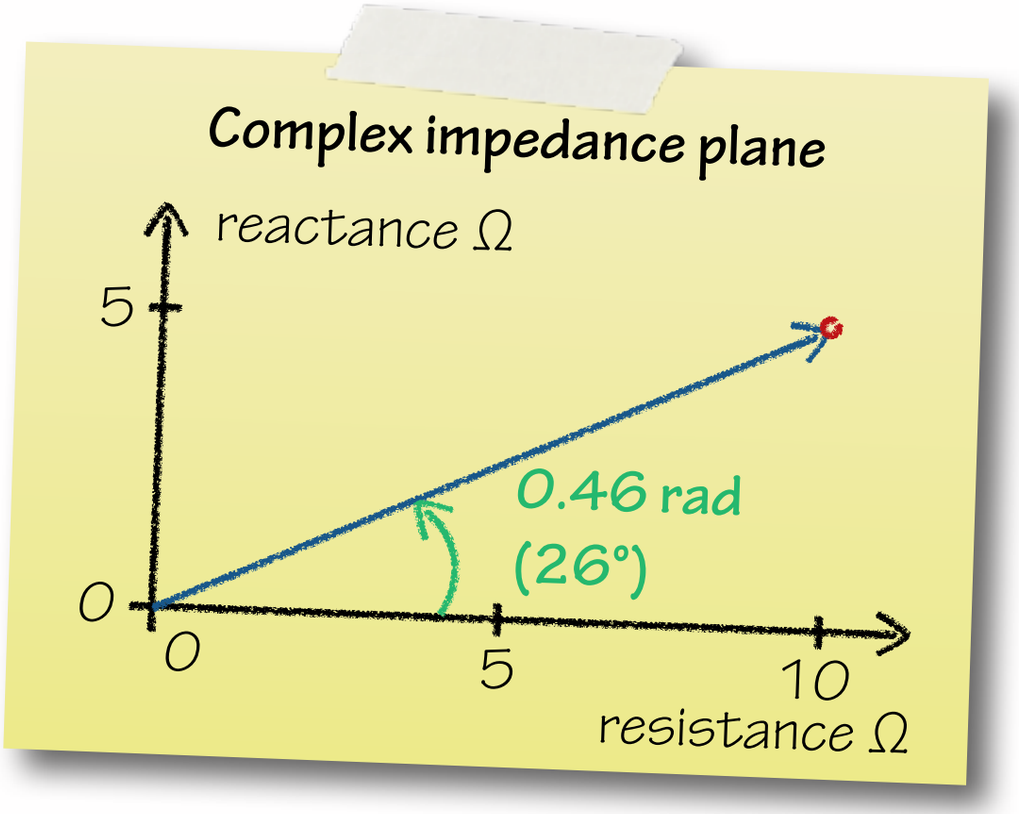
complex

rectangular to **polar** co-ordinates




10 Ω
AC
10 Ω
5 Ω


current
voltage
26° phase


Complex impedance plane
reactance Ω
5
0
0
5
10
resistance Ω
0.46 rad
(26°)

The built-in function

abs()

gives the distance from zero

The **built-in** function

round()

performs **decimal** rounding
for all scalar number types

`round()`
can show **surprising**
behaviour with `float` values
which can't be represented
exactly in binary.

Number base conversions

`bin()`

base 2

`oct()`

base 8

`hex()`

base 16

`int(x, base)`

bases 2 to 36

`int(x, base)`

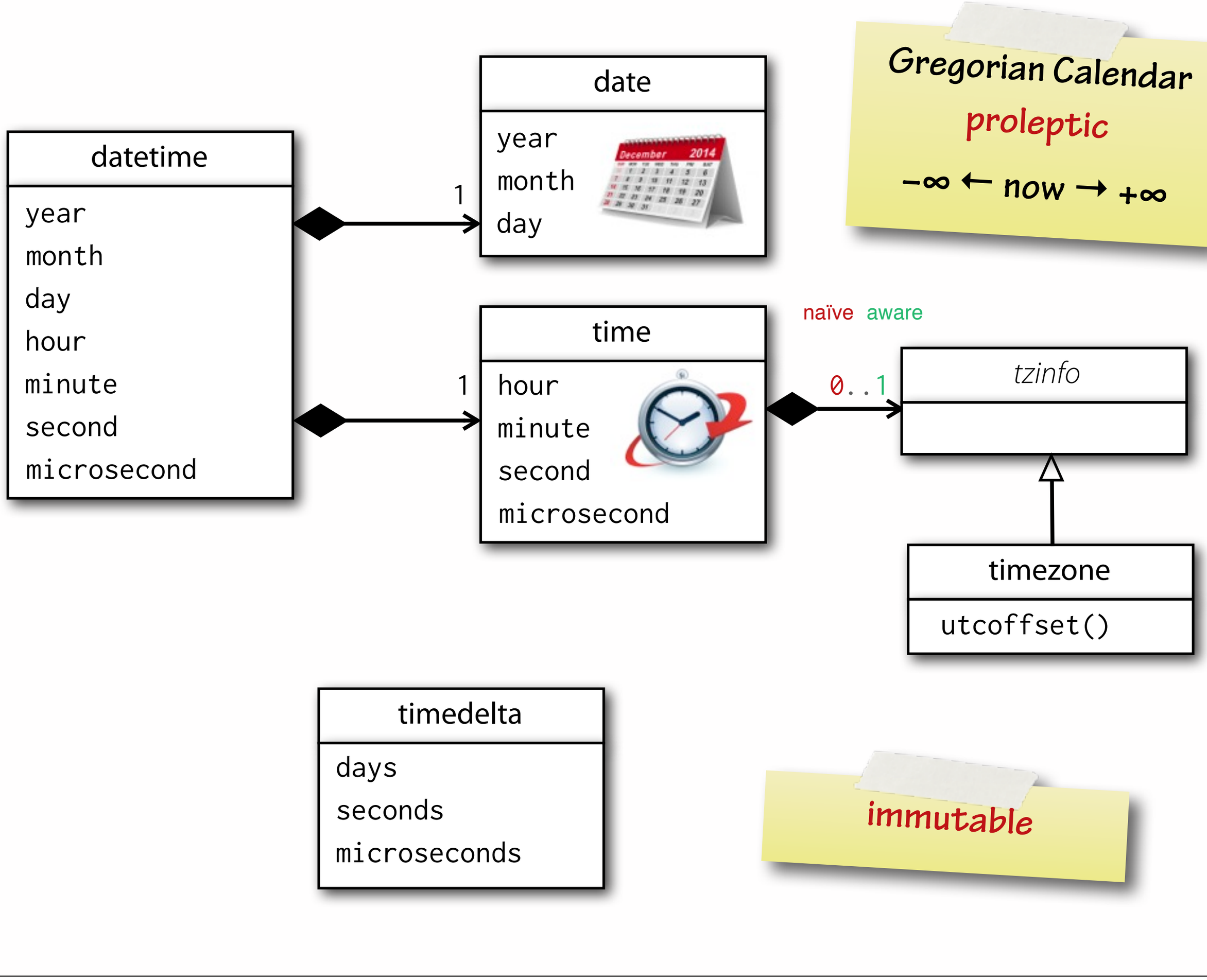
uses `0-9` and `a-z` for digits in bases from 2 to 36.

Specifying **base zero** uses the **base prefix**, defaulting to decimal.

No support for base one (tallying)

The standard library module

datetime


Gregorian Calendar
proleptic
−∞ ← now → +∞
datetime
year
month
day
hour
minute
second
microsecond
1
1
date
year
month
day
December
2014
time
hour
minute
second
microsecond
naïve
aware
0..1
tzinfo
timezone
utcoffset()
timedelta
days
seconds
microseconds
immutable

year: 1-9999
month: 1-12
day: 1-31

`weekday()`

0 Monday
1 Tuesday
2 Wednesday
3 Thursday
4 Friday
5 Saturday
6 Sunday

`isoweekday()`

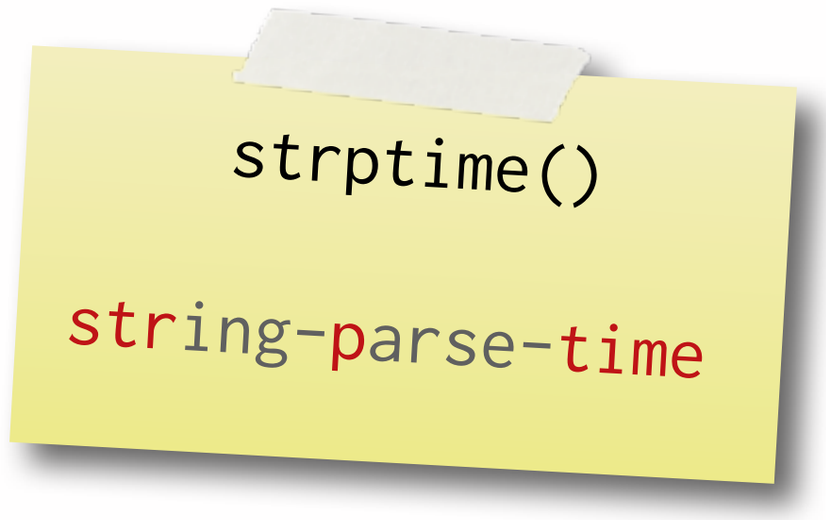
1 Monday
2 Tuesday
3 Wednesday
4 Thursday
5 Friday
6 Saturday
7 Sunday

ISO 8601:2004

Representation of dates and times

strftime()

string-format-time


strptime()
string-parse-time

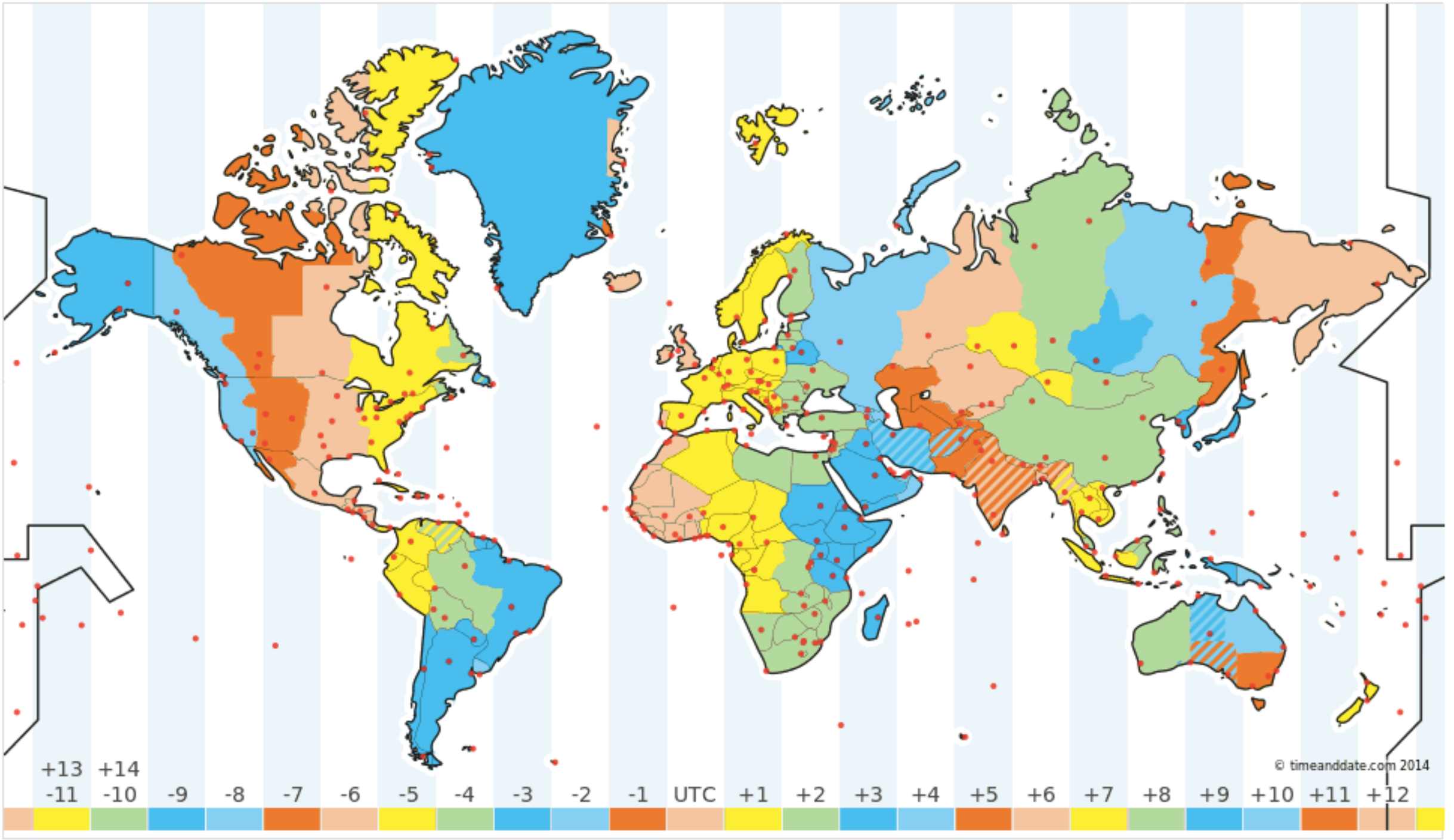
`timedelta`

Constructor accepts and **sums**

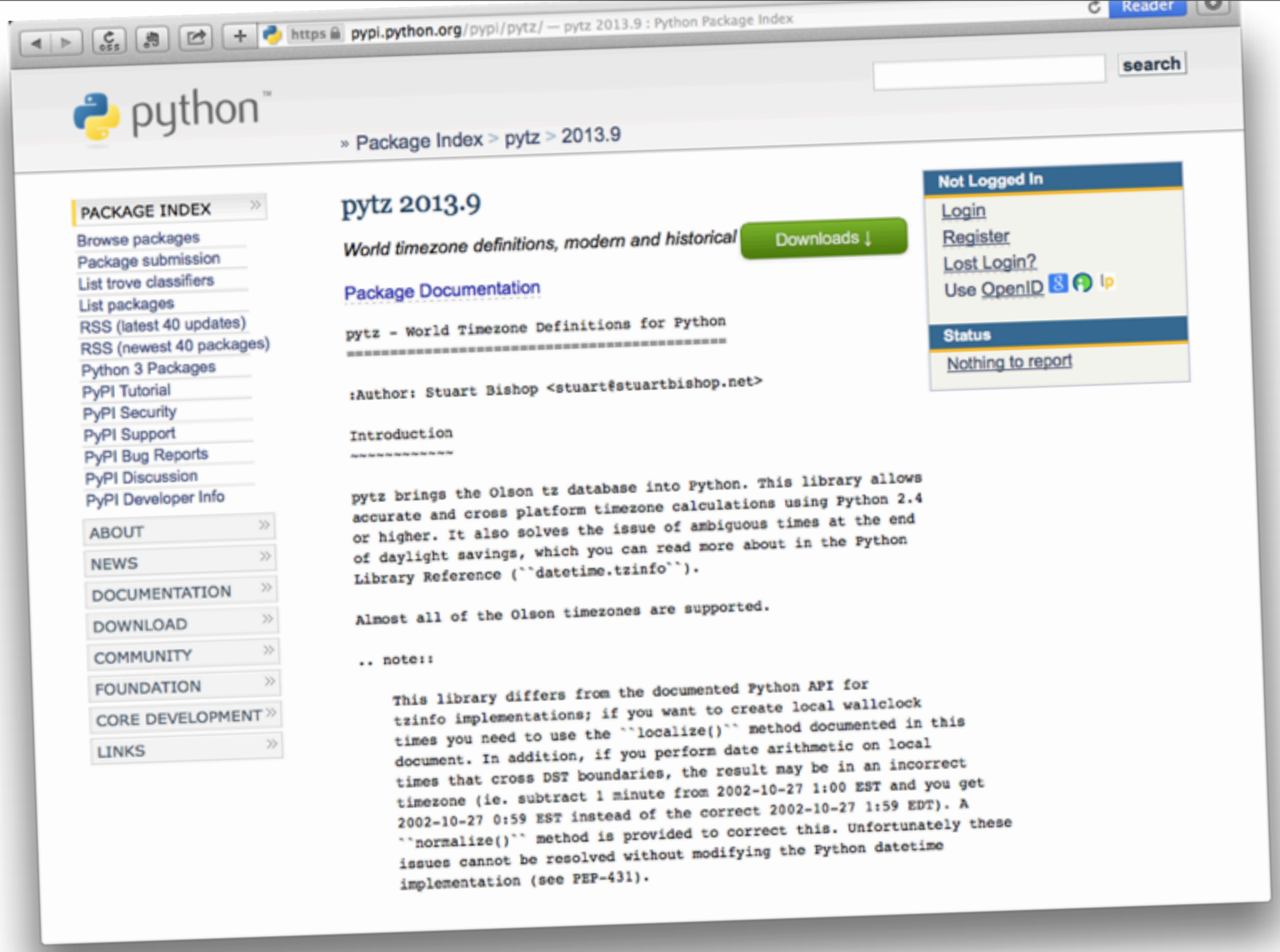
- days
- seconds
- microseconds
- milliseconds
- minutes
- hours
- weeks

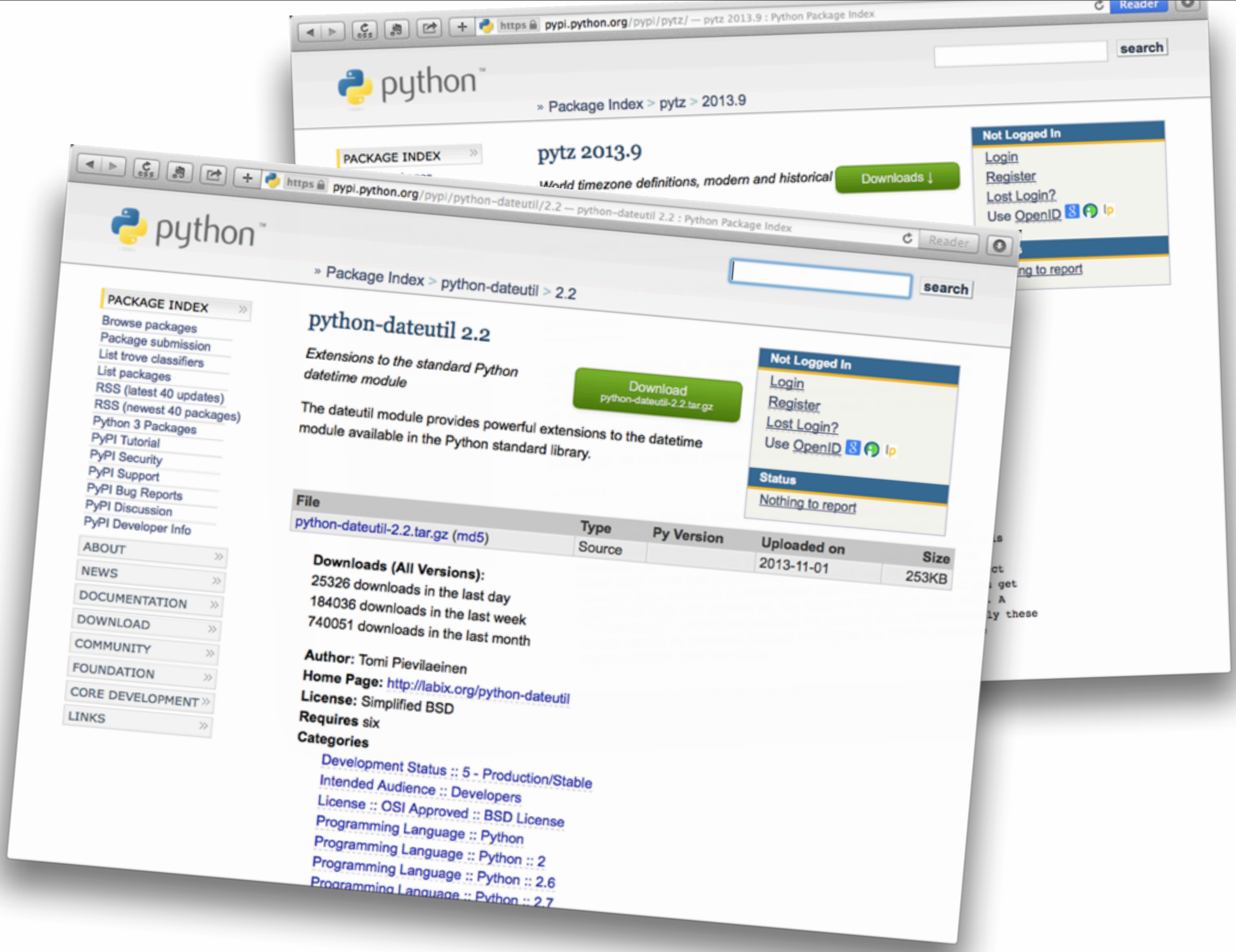
Instances **store** only

- days
- seconds
- microseconds



timeanddate.com

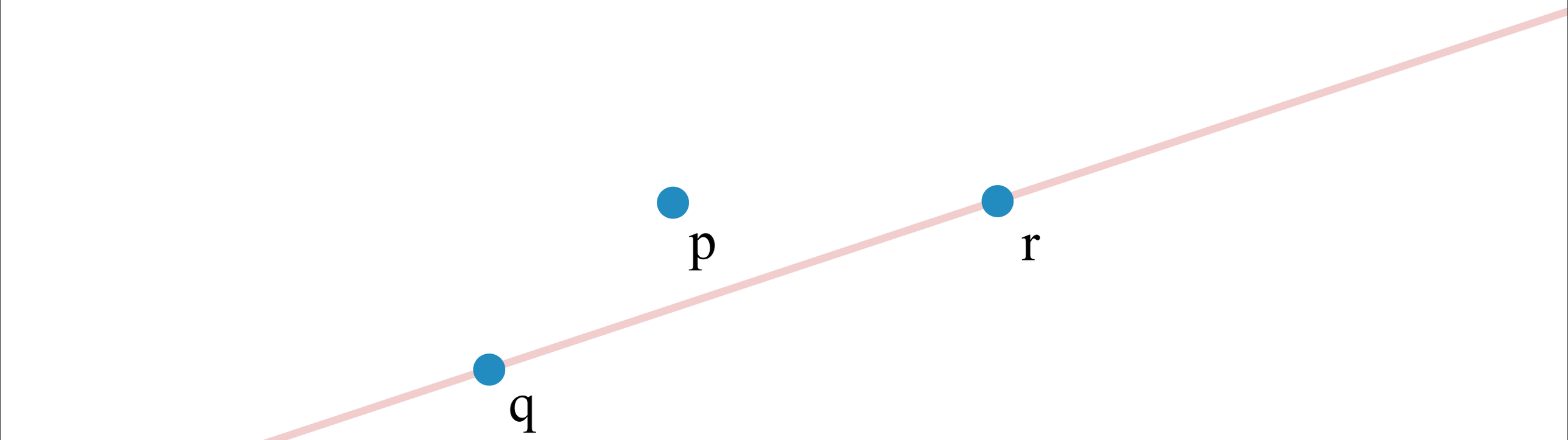

https pypi.python.org/pypi/pytz/ — pytz 2013.9 : Python Package Index
Reader
python
search
» Package Index > pytz > 2013.9
PACKAGE INDEX
Browse packages
Package submission
List trove classifiers
List packages
RSS (latest 40 updates)
RSS (newest 40 packages)
Python 3 Packages
PyPI Tutorial
PyPI Security
PyPI Support
PyPI Bug Reports
PyPI Discussion
PyPI Developer Info
ABOUT
NEWS
DOCUMENTATION
DOWNLOAD
COMMUNITY
FOUNDATION
CORE DEVELOPMENT
LINKS
pytz 2013.9
World timezone definitions, modern and historical
Downloads ↓
Package Documentation
pytz - World Timezone Definitions for Python
:Author: Stuart Bishop <stuart@stuartbishop.net>
Introduction
pytz brings the Olson tz database into Python. This library allows accurate and cross platform timezone calculations using Python 2.4 or higher. It also solves the issue of ambiguous times at the end of daylight savings, which you can read more about in the Python Library Reference (``datetime.tzinfo``).
Almost all of the Olson timezones are supported.
.. note::
This library differs from the documented Python API for tzinfo implementations; if you want to create local wallclock times you need to use the ``localize()`` method documented in this document. In addition, if you perform date arithmetic on local times that cross DST boundaries, the result may be in an incorrect timezone (ie. subtract 1 minute from 2002-10-27 1:00 EST and you get 2002-10-27 0:59 EST instead of the correct 2002-10-27 1:59 EDT). A ``normalize()`` method is provided to correct this. Unfortunately these issues cannot be resolved without modifying the Python datetime implementation (see PEP-431).
Not Logged In
Login
Register
Lost Login?
Use OpenID
Status
Nothing to report


https pypi.python.org/pypi/pytz/ — pytz 2013.9 : Python Package Index
Reader
search
python
» Package Index > pytz > 2013.9
PACKAGE INDEX
pytz 2013.9
World timezone definitions, modern and historical
Downloads ↓
Not Logged In
Login
Register
Lost Login?
Use OpenID
https pypi.python.org/pypi/python-dateutil/2.2 — python-dateutil 2.2 : Python Package Index
Reader
search
python
» Package Index > python-dateutil > 2.2
PACKAGE INDEX
Browse packages
Package submission
List trove classifiers
List packages
RSS (latest 40 updates)
RSS (newest 40 packages)
Python 3 Packages
PyPI Tutorial
PyPI Security
PyPI Support
PyPI Bug Reports
PyPI Discussion
PyPI Developer Info
ABOUT
NEWS
DOCUMENTATION
DOWNLOAD
COMMUNITY
FOUNDATION
CORE DEVELOPMENT
LINKS
python-dateutil 2.2
Extensions to the standard Python datetime module
Download
python-dateutil-2.2.tar.gz
The dateutil module provides powerful extensions to the datetime module available in the Python standard library.
Not Logged In
Login
Register
Lost Login?
Use OpenID
Status
Nothing to report
File
Type
Py Version
Uploaded on
Size
python-dateutil-2.2.tar.gz (md5)
Source
2013-11-01
253KB
Downloads (All Versions):
25326 downloads in the last day
184036 downloads in the last week
740051 downloads in the last month
Author: Tomi Pievilaeinen
Home Page: http://labix.org/python-dateutil
License: Simplified BSD
Requires six
Categories
Development Status :: 5 - Production/Stable
Intended Audience :: Developers
License :: OSI Approved :: BSD License
Programming Language :: Python
Programming Language :: Python :: 2
Programming Language :: Python :: 2.6
Programming Language :: Python :: 2.7

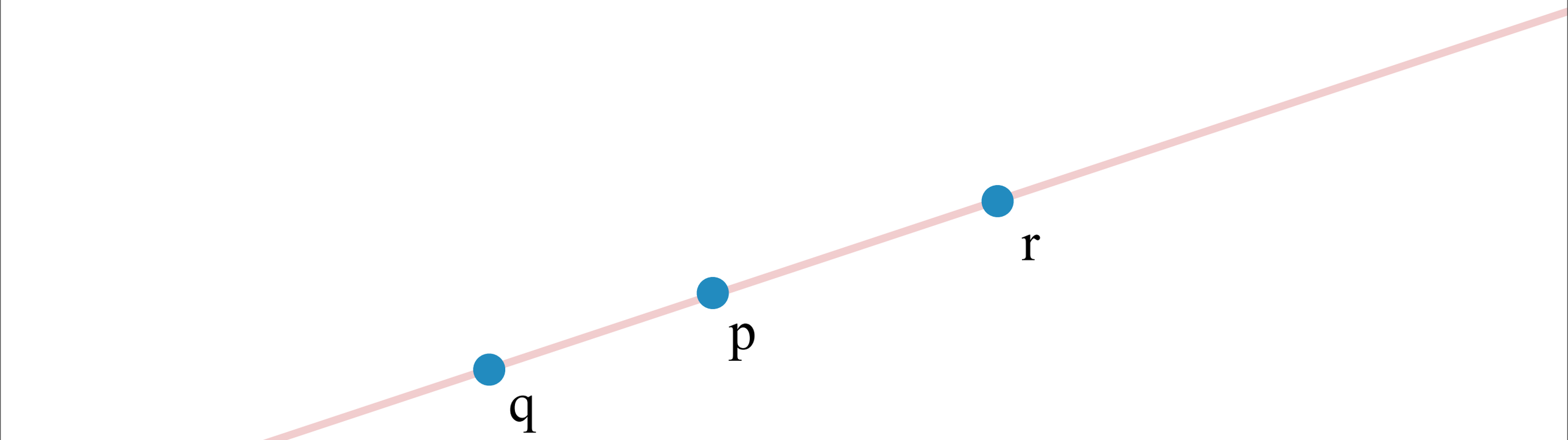
Duck Tails

Floating Point Versus Rational Numbers for Computational Geometry

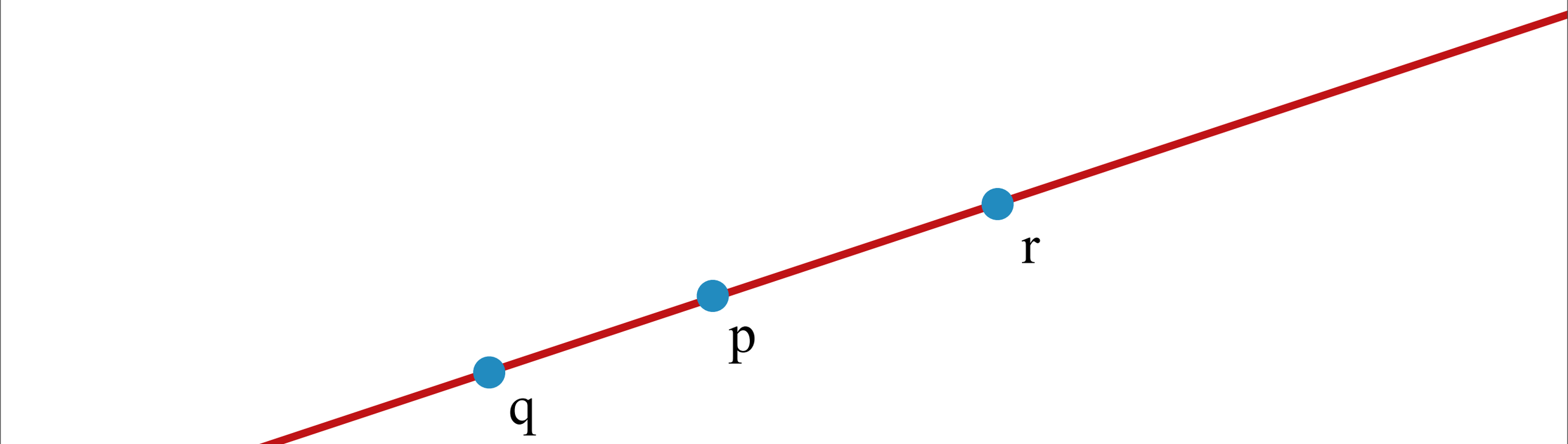
Collinearity Predicate



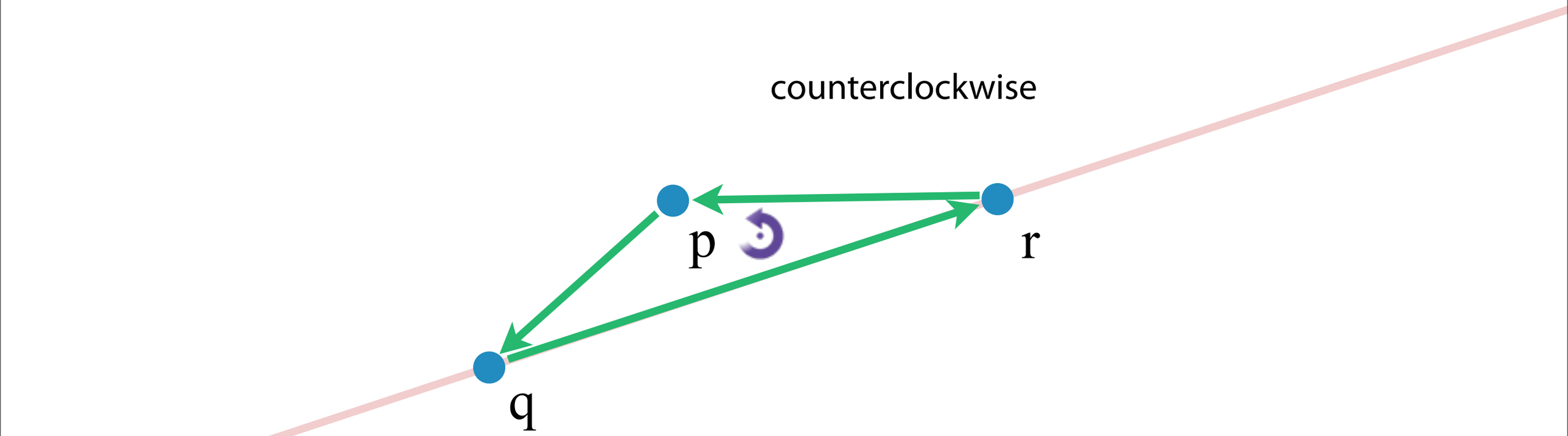
Collinearity Predicate



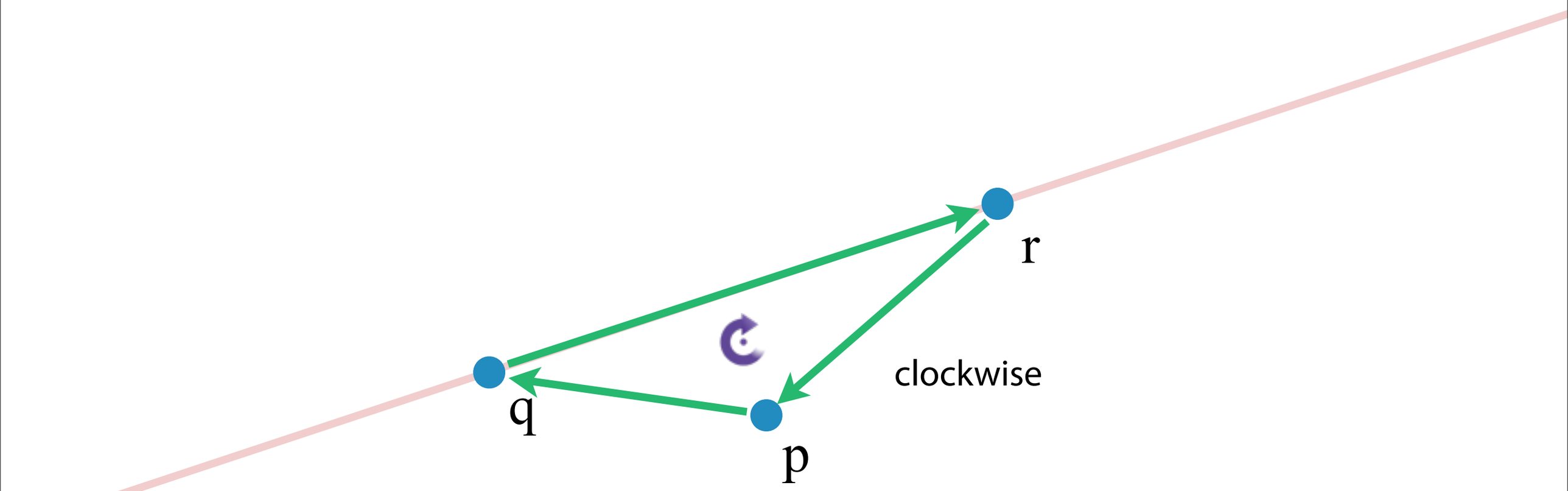
Collinearity Predicate



Orientation Test



Orientation Test

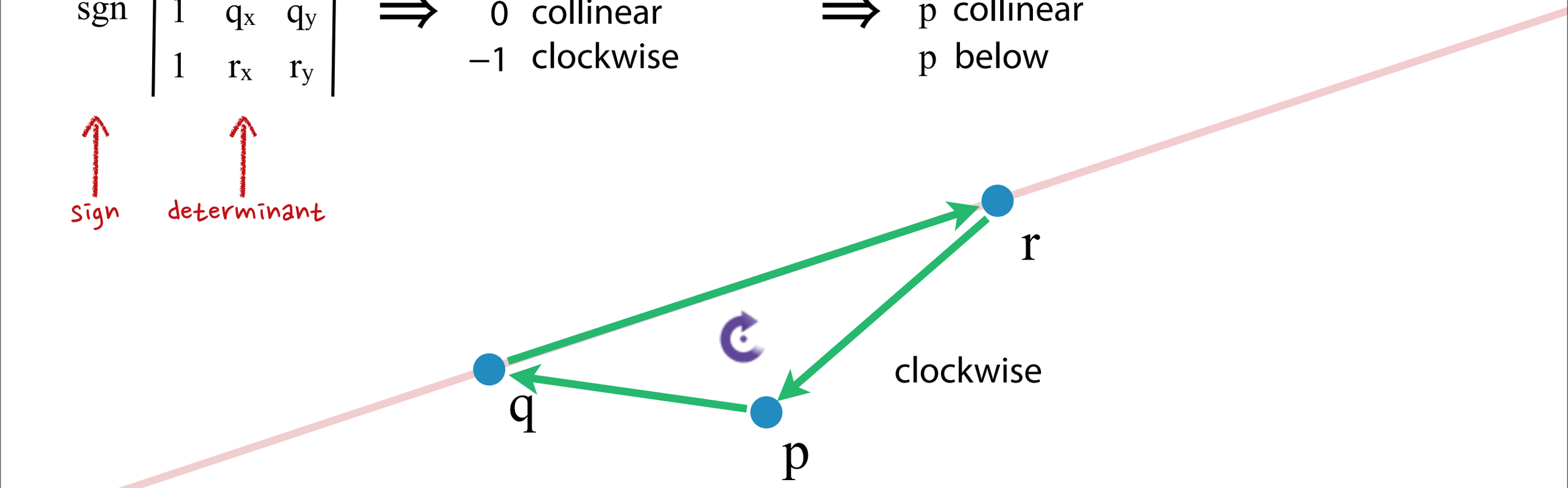


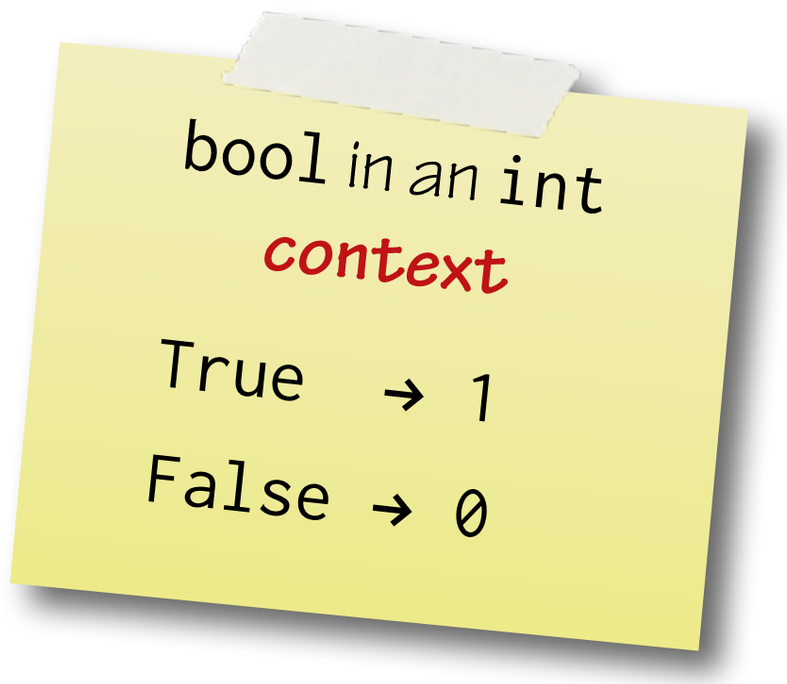
Orientation Test

$$\operatorname{sgn}\begin{vmatrix} 1 & p_x & p_y \\ 1 & q_x & q_y \\ 1 & r_x & r_y \end{vmatrix} \Rightarrow \begin{array}{rl} +1 & \text{counterclockwise} \\ 0 & \text{collinear} \\ -1 & \text{clockwise} \end{array} \Rightarrow \begin{array}{l} p \text{ above} \\ p \text{ collinear} \\ p \text{ below} \end{array}$$

sign

determinant

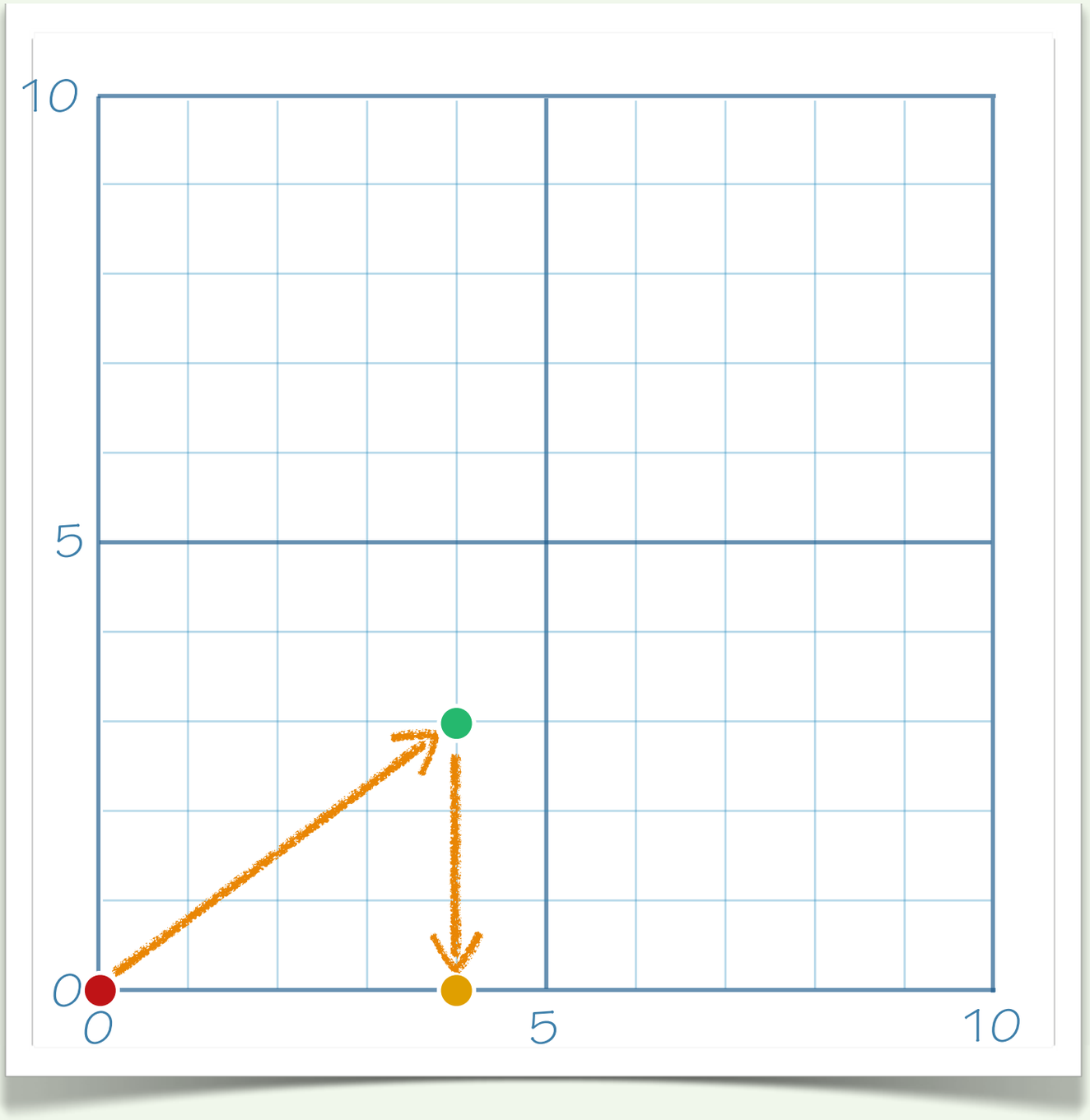


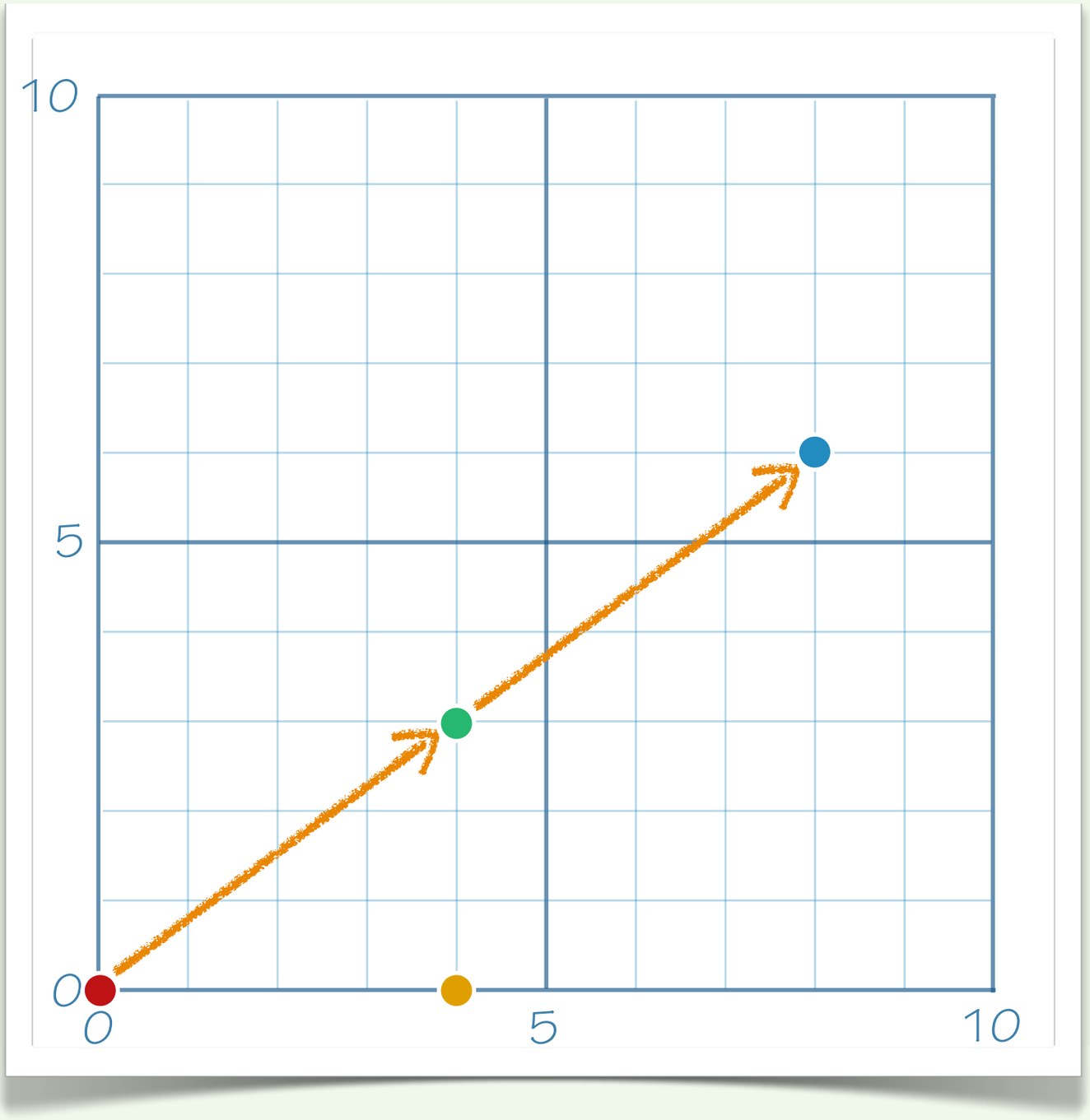

bool in an int
context
True → 1
False → 0

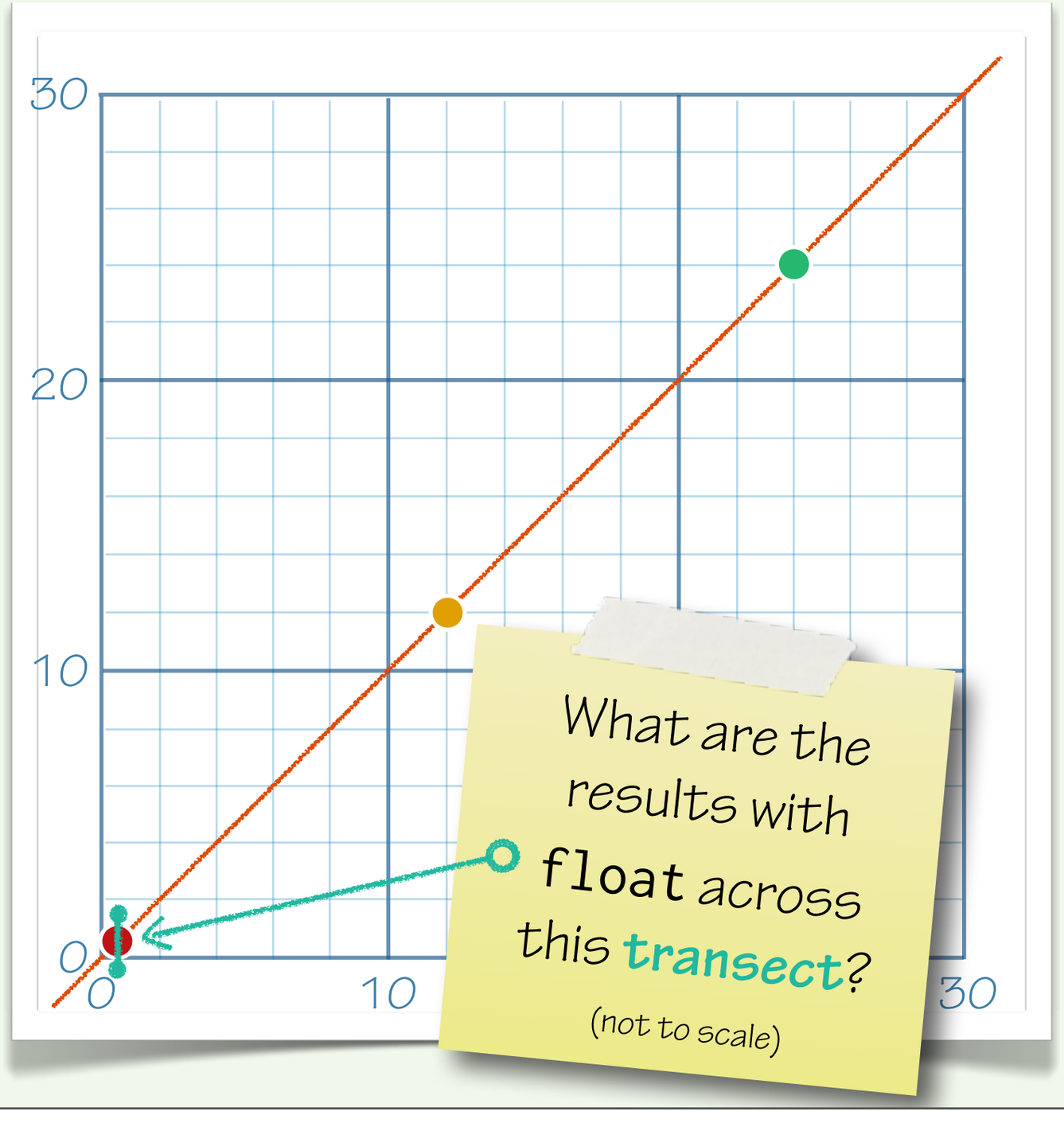
determinant

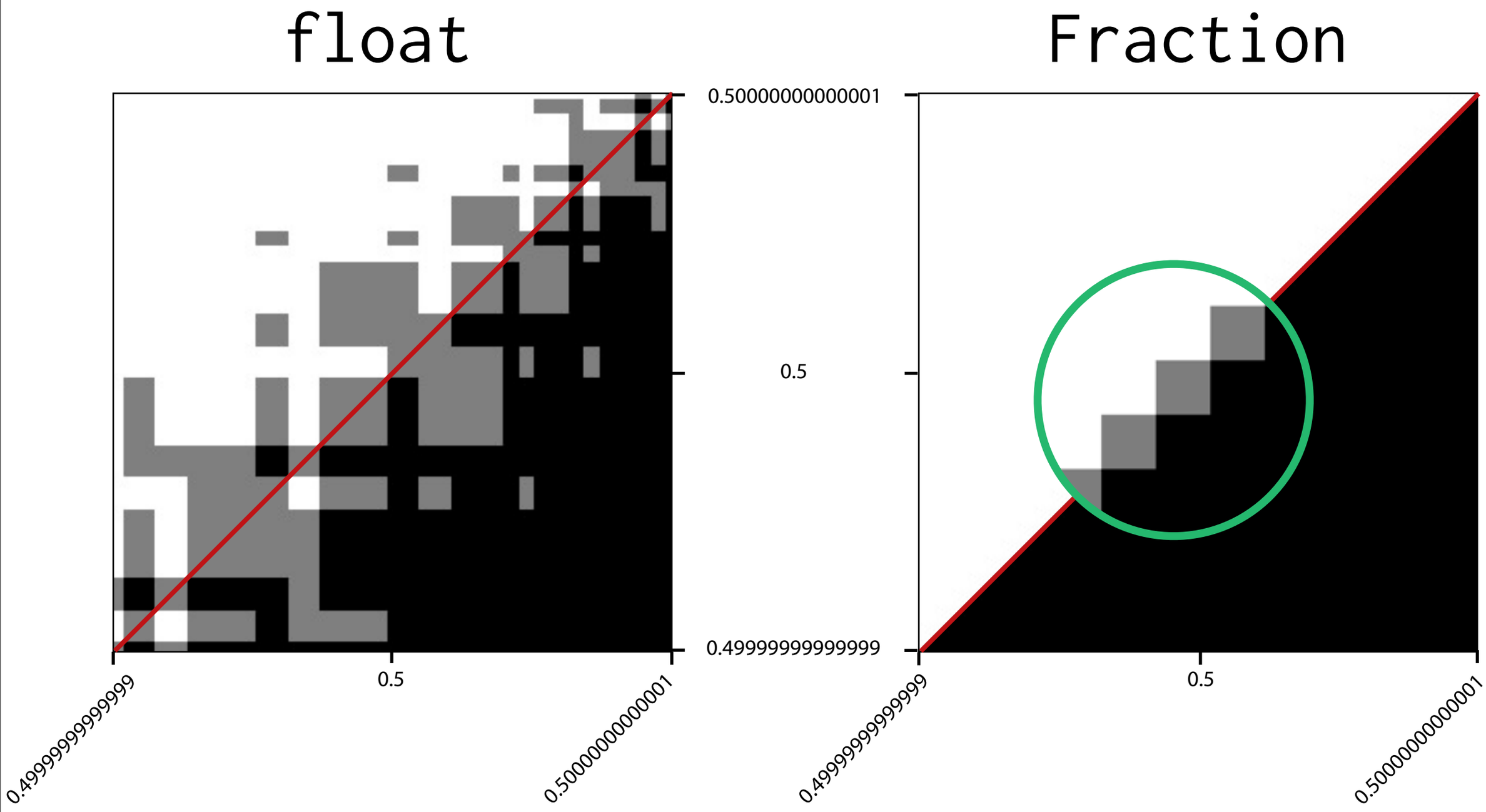
$$\begin{vmatrix} 1 & p_x & p_y \\ 1 & q_x & q_y \\ 1 & r_x & r_y \end{vmatrix}$$

$$= (q_x - p_x)(r_y - p_y) - (q_y - p_y)(r_x - p_x)$$


10
5
0
0
5
10


10
5
0
0
5
10

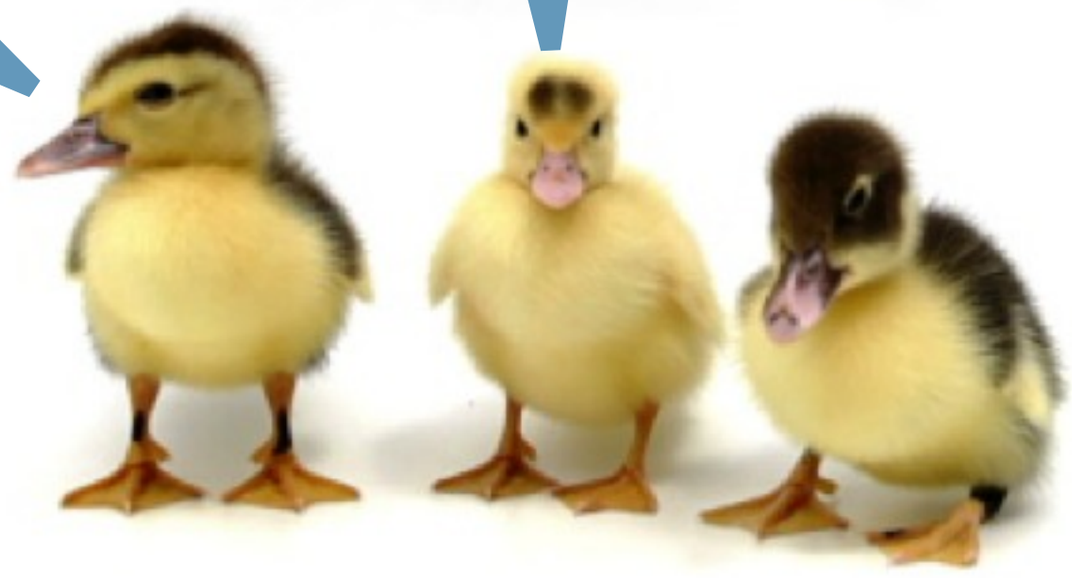

30
20
10
0
0
10
30
What are the results with float across this transect?
(not to scale)


float
Fraction
0.50000000000001
0.5
0.49999999999999
0.4999999999999
0.5
0.5000000000001
0.4999999999999
0.5
0.5000000000001
above
collinear
below

Duck Tails

"Understand the tradeoffs in choosing number types"

"That's what every duckling knows about floating point arithmetic"



Numeric and Scalar Types

```
int
float
sys.float_info
```

```
complex("3+4j")
from cmath import (phase, polar, rect)
```

```
from decimal import Decimal
£ $ ¥
Decimal % → 0
int, float % → -∞
```

```
abs(-5)
round(0.6)
bin(100)
oct(100)
hex(100)
int("100", base=5)
```

```
from fractions import Fraction
f = Fraction("2/3")
```

```
from datetime import (date, time)
from datetime import datetime as Datetime
from datetime import timedelta
from datetime import (tzinfo, timezone)
```