Procedure

Procedure: a description of a computation that, given an input, produces an output.

Example: `def mul(p,q): return p*q`

Computational Problem: an input-output specification that a procedure might be required to satisfy.

Example: Integer factoring

- *input:* an integer m greater than 1.
- *output:* a pair of integers (p, q) greater than 1 such that $p \times q = m$.

Differences:

- Functions/computational problems don't tell us *how* to compute output from input.
- Many procedures might exist for the same spec.
- A computational problem may allow several possible outputs for each input.
  For integer factoring, for input 12 could have output $(2, 6)$ or $(3, 4)$ or...

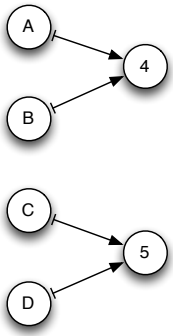
Procedures and functions in Python

We will write procedures in Python, e.g. `def mul(p,q): return p*q`

Often these are called *functions* but we will reserve that term for the mathematical objects.

In Python, we will often use a *dictionary* to represent a function with a finite domain.

The function



can be represented in Python by the dictionary

`{'A':4, 'B':4, 'C':5, 'D':5}`

Probability distribution function

A special kind of function is a *probability distribution function*.
It is used to specify relative likelihood of different outcomes of a single experiment.
It assigns a *probability* (a nonnegative number) to each possible outcome.
The probabilities of all the possible outcomes must sum to 1.

Example:

Probability distribution function for drawing a letter at beginning of the board game Scrabble:

A	9	B	2	C	2	D	4	E	12	F	2	G	3	H	2
I	9	J	1	K	1	L	4	M	2	N	6	O	8	P	2
Q	1	R	6	S	4	T	6	U	4	V	2	W	2	X	1
Y	2	Z	1												

Since the total number of tiles is 98, the probability of drawing an E is 12/98, the probability of drawing an A is 9/98, etc. In Python:

```
{'A':9/98, 'B':2/98, 'C':2/98, 'D':4/98, 'E':12/98, 'F':2/98,
 'G':3/98, 'H':2/98, 'I':9/98, 'J':1/98, 'K':1/98,  'L':1/98,
 'M':2/98, 'N':6/98, 'O':8/98, 'P':2/98, 'Q':1/98,  'R':6/98,
```

Probability distribution function: Uniform distributions

Often the probability distribution is a *uniform distribution*. That means it assigns the same probability to each outcome.

To
model rolling a die, the possible outcomes are 1, 2, 3, 4, 5, and 6, and the probabilities are

$$\Pr(1) = \Pr(2) = \Pr(3) = \Pr(4) = \Pr(5) = \Pr(6) = 1/6 \quad .$$

In Python,

```
>>> Pr = {1:1/6, 2:1/6, 3:1/6, 4:1/6, 5:1/6, 6:1/6}
```

To model the flipping of two coins, the possible outcomes are HH, HT, TH, TT and the probability of all outcomes is $1/4$.
In Python,

```
>>> Pr = {('H', 'H'):1/4, ('H', 'T'):1/4,
          ('T', 'H'):1/4, ('T', 'T'):1/4}
```

[1] The Field

The Field: Introduction to complex numbers

Solutions to $x^2 = -1$?

Mathematicians invented $\mathbf{i}$ to be one solution

Guest Week: Bill Amend (excerpt, `http://xkcd.com/824`)

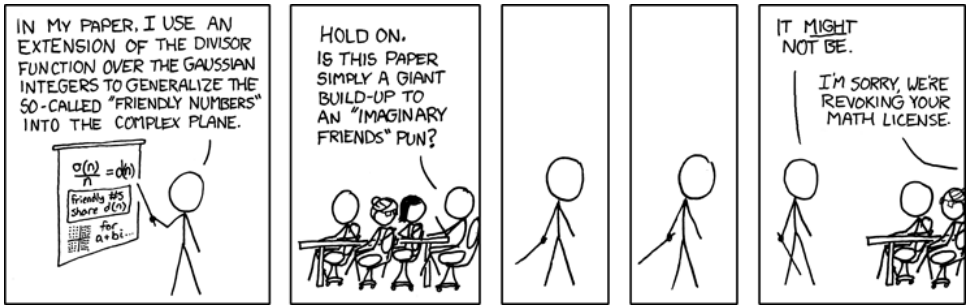
Can use $\mathbf{i}$ to solve other equations, e.g.:

$$x^2 = -9$$

Solution is $x = 3\mathbf{i}$

Introduction to complex numbers

Numbers such as $\mathbf{i}$, $-\mathbf{i}$, $3\mathbf{i}$, $2.17\mathbf{i}$ are called *imaginary* numbers.



Math Paper (`http://xkcd.com/410`)

The Field: Introduction to complex numbers

- Solution to $(x-1)^2 = -9$?
- One is $x = 1 + 3\mathbf{i}$.

- A real number plus an imaginary number is a *complex number*.
- A complex number has a *real part* and an *imaginary part*.

$$\text{complex number} = (\text{real part}) + (\text{imaginary part})\,\mathbf{i}$$

The Field: Complex numbers in Python

Abstracting over *Fields*

- *Overloading:* Same names (+, etc.) used in Python for operations on real numbers and for operations complex numbers
- Write procedure `solve(a,b,c)` to solve $ax + b = c$:

  ```
  >>> def solve(a,b,c): return (c-b)/a
  ```

  Can now solve equation $10x + 5 = 30$:

  ```
  >>> solve(10, 5, 30)
  2.5
  ```
- Can also solve equation $(10 + 5\mathbf{i})x + 5 = 20$:

  ```
  >>> solve(10+5j, 5, 20)
  (1.2-0.6j)
  ```
- Same procedure works on complex numbers.

Abstracting over *Fields*

Why does procedure works with complex numbers?

Correctness based on:

- / is inverse of *
- - is inverse of +

Similarly, much of linear algebra based just on +, -, *, / and algebraic properties

- / is inverse of *
- - is inverse of +
- *addition is commutative:* $a + b = b + a$
- *multiplication distributes over addition:* $a * (b + c) = a * b + a * c$
- etc.

You can plug in any collection of "numbers" with arithmetic operators +, -, *, / satisfying the algebraic properties— and much of linear algebra will still "work".

Such a collection of "numbers" with +, -, *, / is called a *field*.
Different fields are like different classes obeying the same interface.

Field notation

When we want to refer to a field without specifying which field, we will use the notation $\mathbb{F}$.

Abstracting over *Fields*

We study three fields:

- The field $\mathbb{R}$ of real numbers
- The field $\mathbb{C}$ of complex numbers
- The finite field $GF(2)$, which consists of 0 and 1 under mod 2 arithmetic.

Reasons for studying the field $\mathbb{C}$ of complex numbers:

- $\mathbb{C}$ is similar enough to $\mathbb{R}$ to be familiar but different enough to illustrate the idea of a field.
- Complex numbers are built into Python.
- Complex numbers are the intellectual ancestors of vectors.
- In more advanced parts of linear algebra, complex numbers play an important role.

Rotation of Complex Numbers activity

You want a function that rotates by angle $\pi/4$.

1. For what number τ is the function $z \mapsto e^{\tau \mathbf{i}} z$?
2. For input $z = 3e^{(\pi/3)\mathbf{i}}$, what is the output of the rotation function?
3. Draw a diagram of the complex plane showing
   - the circle of radius 1, and
   - the input and output points.

Playing with $GF(2)$

Galois Field 2

has just two elements: 0 and 1

Addition is like exclusive-or:

$+$	0	1
0	0	1
1	1	0

Multiplication is like ordinary multiplication

$\times$	0	1
0	0	0
1	0	1

Evariste Galois, 1811-1832

Usual algebraic laws still hold, e.g. multiplication distributes over addition

$a \cdot (b + c) = a \cdot b + a \cdot c$

$GF(2)$ in Python

We provide a module `GF2` that defines a value `one`.
This value acts like 1 in $GF(2)$:

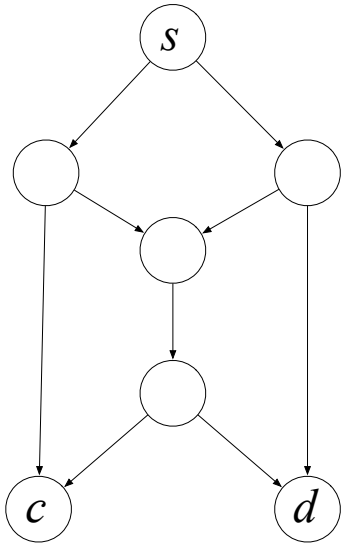
```
>>> from GF2 import one
>>> one + one
0
>>> one * one
one
>>> one * 0
0
>>> one/one
one
```

We will use `one` in coding with $GF(2)$.

Playing with $GF(2)$: Network coding

Streaming video through a network

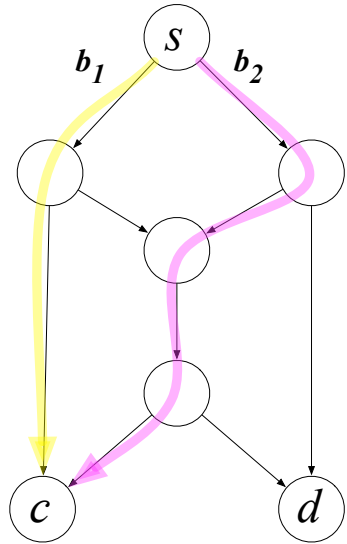
- one customer—no problem
- two customers—contention! ☹
- do computation at intermediate nodes — avoids contention
- Network coding doubles throughput in this example!



Playing with $GF(2)$: Network coding

Streaming video through a network

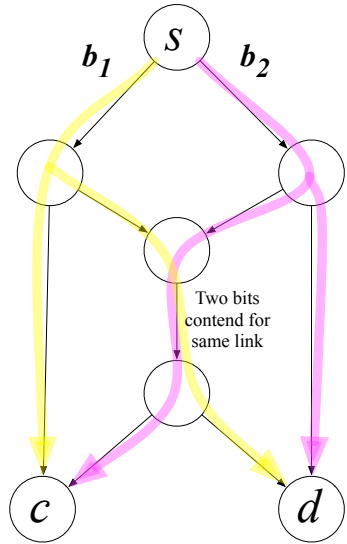
- one customer—no problem
- two customers—contention! ☹
- do computation at intermediate nodes — avoids contention
- Network coding doubles throughput in this example!



Playing with $GF(2)$: Network coding

Streaming video through a network

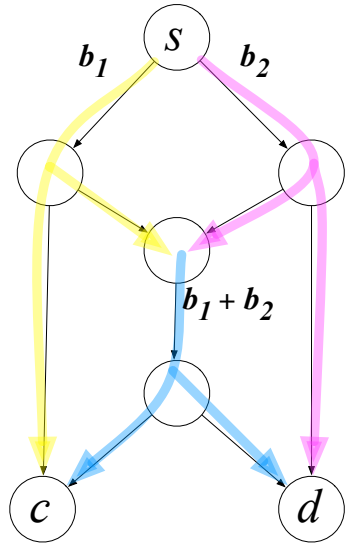
- one customer—no problem
- two customers—contention! ☹
- do computation at intermediate nodes — avoids contention
- Network coding doubles throughput in this example!



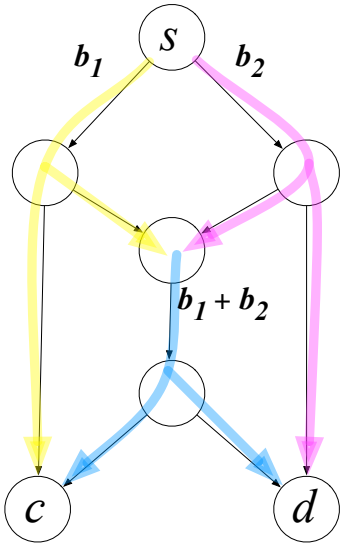
Playing with $GF(2)$: Network coding

Streaming video through a network

- one customer—no problem
- two customers—contention! ☹
- do computation at intermediate nodes — avoids contention
- Network coding doubles throughput in this example!



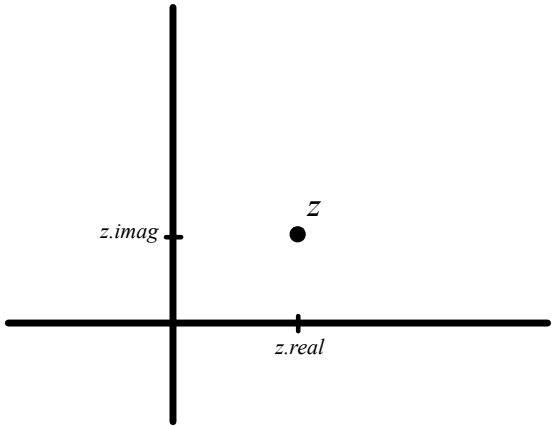
Network Coding activity



Suppose the bits that need to be transmitted in a given moment are $b_1 = 1$ and $b_2 = 1$. Label each link of the network with the bit transmitted across it according to the network-coding scheme. Show how the customer nodes c and d can recover b_1 and b_2.

Complex numbers as points in the complex plane

Can interpret *real* and *imaginary* parts of a complex number as x and y coordinates. Thus can interpret a complex number as a *point* in the plane

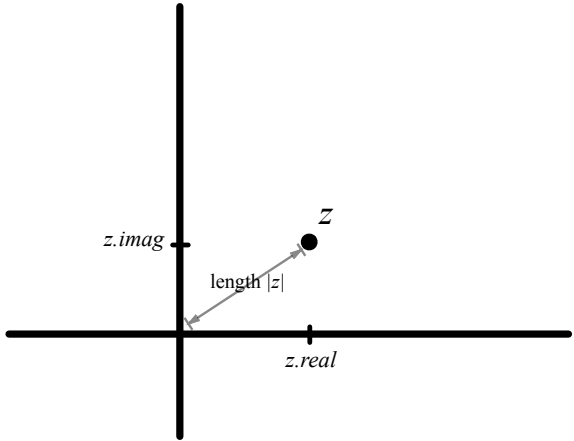


(the *complex plane*)

Playing with $\mathbb{C}$

Playing with $\mathbb{C}$: The absolute value of a complex number

Absolute value of z = distance from the origin to the point z in the complex plane.

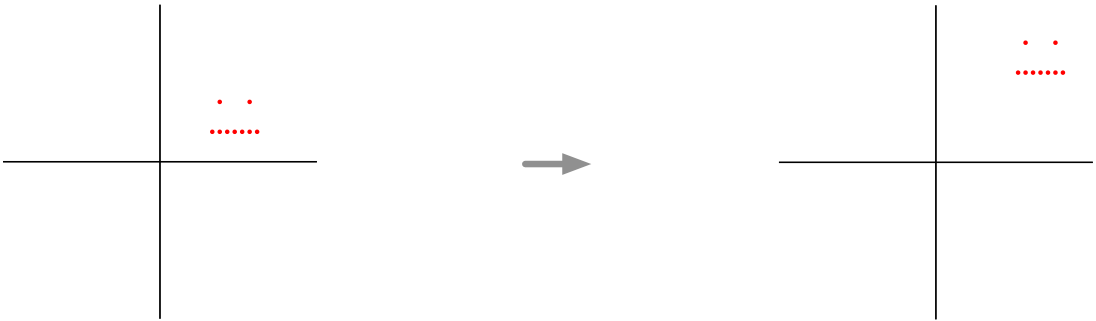


- In Mathese, written $|z|$.
- In Python, written `abs(z)`.

Playing with $\mathbb{C}$: Adding complex numbers

Geometric interpretation of $f(z) = z + (1 + 2\mathbf{i})$?

Increase each real coordinate by 1 and increases each imaginary coordinate by 2.



$f(z) = z + (1 + 2\mathbf{i})$ is called a *translation.*

Playing with $\mathbb{C}$: Adding complex numbers

- *Translation in general:*

$$f(z) = z + z_0$$

  where z_0 is a complex number.

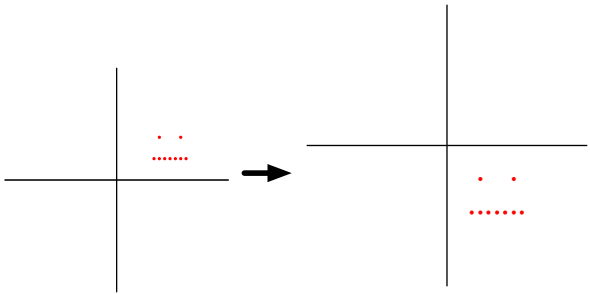
- A translation can "move" the picture anywhere in the complex plane.

Playing with $\mathbb{C}$: Adding complex numbers

- *Quiz:* The "left eye" of the list L of complex numbers is located at $2 + 2\mathbf{i}$. For what complex number z_0 does the translation

$$f(z) = z + z_0$$

move the left eye to $1 - 1\mathbf{i}$?

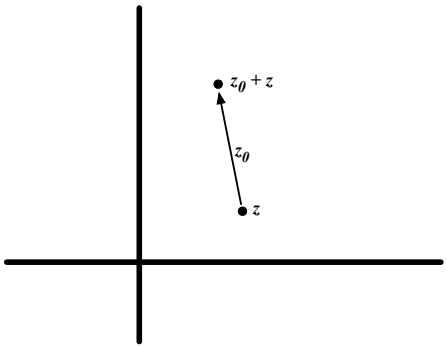


- *Answer:* $z_0 = -1 - 3\mathbf{i}$

Playing with $\mathbb{C}$: Adding complex numbers: Complex numbers as arrows

Interpret z_0 as representing the translation $f(z) = z + z_0$.

- Visualize a complex number z_0 as an arrow.
- Arrow's tail located an any point z
- Arrow's head located at $z + z_0$
- Shows an example of what the translation $f(z) = z + z_0$ does



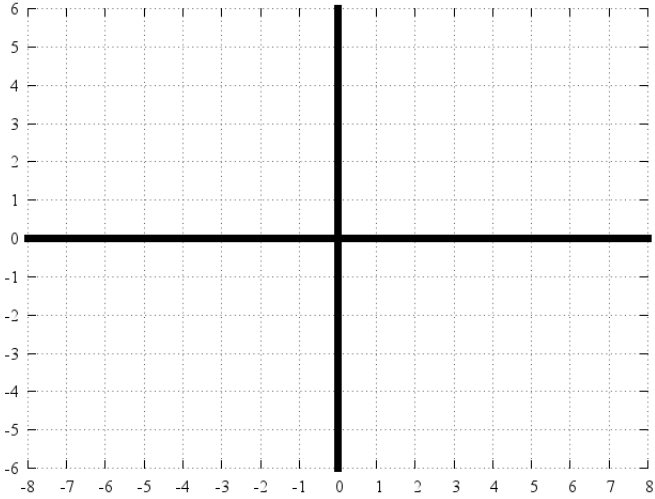
Complex Translation activity

Consider the translation that maps $1 + 1\mathbf{i}$ to $5 + 3\mathbf{i}$.

1. Apply that translation to $4 + 4\mathbf{i}$. What is the result?
2. Apply that translation to 0. What is the result?
3. The translation can be written as $z \mapsto z + z_0$ for some fixed complex number z_0. What is z_0?
4. In the complex plane, draw two arrows corresponding to this translation. One arrow should have its tail at $-2 - 2\mathbf{i}$. The other should have its tail at $-1 + \mathbf{i}$.

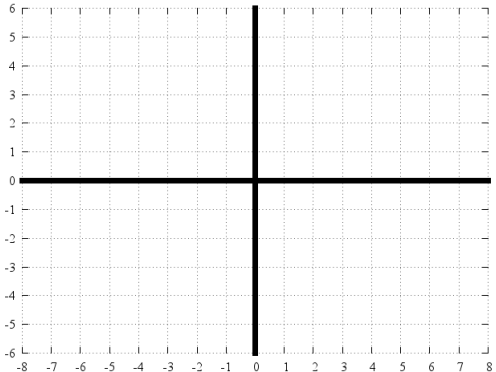
Playing with $\mathbb{C}$: Adding complex numbers: Complex numbers as arrows

Example: Represent $-6 + 5\mathbf{i}$ as an arrow.



Playing with $\mathbb{C}$: Adding complex numbers: Composing translations, adding arrows

- Consider two complex numbers z_1 and z_2.
- They correspond to translations $f_1(z) = z + z_1$ and $f_2(z) = z + z_2$
- Functional composition: $(f_1 \circ f_2)(z) = z + z_1 + z_2$
- Represent functional composition by adding arrows.
- *Example:* $z_1 = 2 + 3\mathbf{i}$ and $z_2 = 3 + 1\mathbf{i}$



Composition of Translations activity

Consider the points $z_1 = -1 + 2\mathbf{i}$ and $z_2 = 3 + 1\mathbf{i}$. You will make three different plots.

1. Plot the points in the complex plane.
2. Draw arrows representing the translations $z \mapsto z + z_1$ and $z \mapsto z + z_2$. The tails of these arrows can be anywhere except the origin.
3. Draw the arrows again, but this time the tail of the second arrow should be at the head of the first arrow. Then draw the arrow whose tail is the tail of the first arrow and whose head is the head of the second arrow. What is the translation represented by this arrow?
4. Check that the complex number you get for this last arrow is the complex number $z_1 + z_2$