IMPERIAL
BUSINESS SCHOOL

Data Structures and Algorithms

Lecture 2

Ryan Cory-Wright
r.cory-wright@imperial.ac.uk

A Brief Statement on The Difficulty of This Class

- Python is not a prerequisite
- If you have coded before, you will have a head start
- If you haven't, that's totally fine—this class is designed to get you there, and the tutorials + exercises are where most of the learning happens

Today: What You Should be Able to do

By the end of today, you should be able to:

- Explain what a **function** is (inputs $\rightarrow$ outputs) and why we use them.
- Write a simple function with a `return` statement.
- Describe an **algorithm** as step-by-step instructions for turning input into output.
- Implement an algorithm in Python and sanity-check it on small examples.

Note: you will learn some of this in the coding part of the lecture.

Today

1. **Recap**
2. Functions
3. A first algorithm

What is the output?

```
x = 5
6 = y
print(x)
print(y)
```

A. 5, then 6

B. 5, then 5

C. 6, then 6

D. An error

E. I don’t know

What is the output?

```python
a = 2
b = a
a = 5
print(a)
print(b)
```

A. 2, then 2

B. 2, then 5

C. 5, then 2

D. 5, then 5

E. I don’t know

What is the output?

```python
x = 5
if x >= 0:
    print(1)
elif x < 20:
    print(2)
else:
    print(3)
print(4)
```

A. 1, then 2, then 4

B. 1, then 4

C. 4

D. 3, then 4

E. I don't know

What is the output?

```python
x = 3
while x > 0:
    print(x)
    x = x - 1
```

A. 3, 2, 1

B. 3, 2, 1, 0

C. 3, 2

D. 2, 1, 0

E. I don’t know

What is the output?

```python
x = 3
while x > 0:
    x = x - 1
print(x)
```

A. 3, 2, 1

B. 2, 1, 0

C. 1

D. 0

E. I don’t know

Today

1. Recap
2. **Functions**
3. A first algorithm
4. Homework 1

We have already been using functions

In Session 1, we used built-in functions:

```
>>> abs(-3)
3
>>> max(5, 3, 10)
10
>>> max(abs(-5), min(3, 9))
```

We say we **call** the function, specifying the arguments within parentheses.

What happens when we do this, and why are functions useful?

We use functions to organise tasks

A function is a named group of statements to perform a specific task.

- Input data $\rightarrow$ function $\rightarrow$ output data

We use functions to organise tasks

A function is a named group of statements to perform a specific task.

- Input data $\rightarrow$ function $\rightarrow$ output data

```python
# Let's define a function abs_value
def abs_value(a):
    if a < 0:
        return -a # The return statement stops function execution, outputs -a
    else:
        return a

# This function call runs the code block inside abs_value for a = -3
# The returned value is assigned to the variable y
y = abs_value(-3)
```

A function is like a **factory**: in goes input data (car parts), out comes output data (car).

A function may have multiple parameters separated by commas. It may return multiple values separated by commas.

Why functions?

1. **Abstraction**: user does not need to know what happens inside
2. Make **code easily re-usable** and modular
3. **Changing code becomes easier**: we don’t have to copy same code in many places
    - Best practice: If you ever find yourself copy-pasting code, stop and write a function instead.
    - Otherwise, you might fix a bug in one part of code and forget to fix it elsewhere.

A Reliable Pattern for Writing Functions

Step 1: What are the Inputs/Outputs? What goes in? What comes out? Any assumptions?

Step 2: Write down the function

```python
def is_even(n: int) -> bool:
    """Return True iff n is an even integer."""
    return (n % 2) == 0
```

Step 3: Test on small instances immediately.

```python
assert is_even(0) is True
assert is_even(1) is False
assert is_even(10) is True
```

Today

1. Recap
2. Functions
3. **A first algorithm**

Solving computational problems

Data = digitised information

Data structures describe ways to organise data

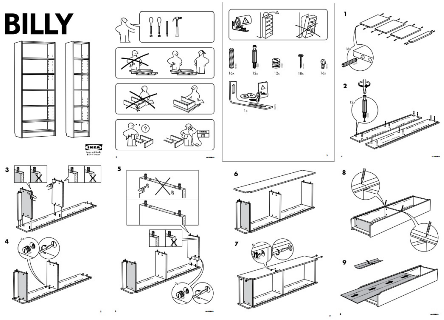
Algorithms describe how we process data:

- Step-by-step instructions
- Take input data and produce output data

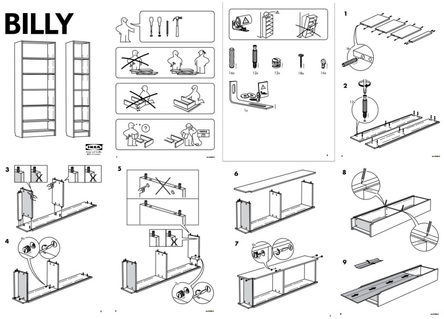
We write algorithms into **programs** (eg in Python)

Computers interpret and execute programs

An algorithm is a recipe



An algorithm is a recipe



Algorithm:

- Step-by-step instructions
- Takes input (data) and produces output (data)

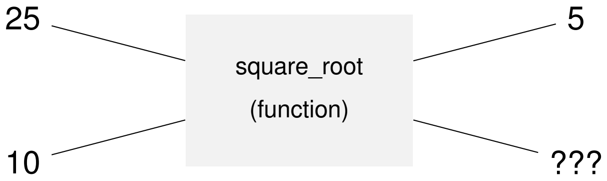
Pics: Hungry Gals, IKEA.

How do you calculate a square root?

The square root of a number x is a number y such that $y \times y = x$ (let's focus on positive roots)

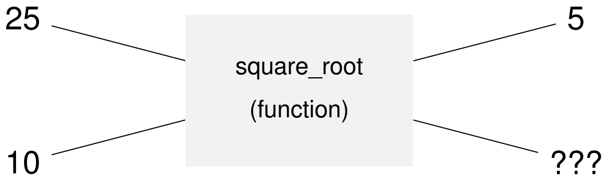
How do you calculate a square root?

The square root of a number x is a number y such that $y \times y = x$ (let's focus on positive roots)



How do you calculate a square root?

The square root of a number x is a number y such that $y \times y = x$ (let's focus on positive roots)



A function is like a factory

- In goes number, out comes square root
- Inside the factory, there's an **algorithm**

Square-root algorithm

The square root of x is y such that $y \times y = x$

Algorithm (Heron of Alexandria, first century AD):

- Make a guess, for example $x/2$

Square-root algorithm

The square root of x is y such that $y \times y = x$

Algorithm (Heron of Alexandria, first century AD):

- Make a guess, for example $x/2$
- Repeat three times:

Square-root algorithm

The square root of x is y such that $y \times y = x$

Algorithm (Heron of Alexandria, first century AD):

- Make a guess, for example $x/2$
- Repeat three times:
    - Divide the original number x by the guess to get a ratio

Square-root algorithm

The square root of x is y such that $y \times y = x$

Algorithm (Heron of Alexandria, first century AD):

- Make a guess, for example $x/2$
- Repeat three times:
    - Divide the original number x by the guess to get a ratio
    - Find the average of the guess and the ratio

Square-root algorithm

The square root of x is y such that $y \times y = x$

Algorithm (Heron of Alexandria, first century AD):

- Make a guess, for example $x/2$
- Repeat three times:
    - Divide the original number x by the guess to get a ratio
    - Find the average of the guess and the ratio
    - Use this average as the next guess

Square-root algorithm

The square root of x is y such that $y \times y = x$

Algorithm (Heron of Alexandria, first century AD):

- Make a guess, for example $x/2$
- Repeat three times:
    - Divide the original number x by the guess to get a ratio
    - Find the average of the guess and the ratio
    - Use this average as the next guess

	x	g	g*g	x/g	(g+x/g)/2
$i = 1$	10	5	25	2	3.5
$i = 2$	10	3.5	12.25	2.857	3.179
$i = 3$	10	3.179	10.103	Close	enough!

Let’s use Python

Square-root function

```python
def square_root(x):
    guess = x/2
    eps = 0.01
    while abs(guess*guess-x) >= eps:
        guess = (guess + x/guess)/2
    return guess

z = 20
y = square_root(z)
```

Square-root function

```python
def square_root(x):
    guess = x/2
    eps = 0.01
    while abs(guess*guess-x) >= eps:
        guess = (guess + x/guess)/2
    return guess

z = 20
y = square_root(z)
```

- Takes input *x* and outputs its square root
- Note: uses another function inside it: built-in function `abs`
- Abstraction, reusability, reliability

How Good is This Algorithm? (Taylor's Version)

Recall from Harjoat's Class: Taylor Series Expansion of a Function

$$f(x) = f(a) + \sum_{k=1}^{\infty} \frac{f^{(k)}(a)(x-a)^k}{k!} \tag{1}$$

And we have:

- $f(x) = \sqrt{x}$
- $f'(x) = \frac{1}{2\sqrt{x}}$
- $f''(x) = \frac{-1}{4x^{3/2}} \ldots$

How Good is This Algorithm? (Taylor's Version)

Recall from Harjoat's Class: Taylor Series Expansion of a Function

$$f(x) = f(a) + \sum_{k=1}^{\infty} \frac{f^{(k)}(a)(x-a)^k}{k!} \tag{1}$$

And we have:

- $f(x) = \sqrt{x}$
- $f'(x) = \frac{1}{2\sqrt{x}}$
- $f''(x) = \frac{-1}{4x^{3/2}} \ \ldots$

Approximate $\sqrt{x}$ about $x = h_0^2$ using first two terms in Taylor series:

$$\sqrt{x} \approx h_0 + \frac{x - h_0^2}{2h_0} = \frac{\frac{x}{h_0} + h_0}{2}$$

Which is Heron's formula. So:

- Heron implicitly used a Taylor Series expansion!
- Formula accurate up to second-order terms in Taylor series
- To get a better formula: use more terms from Taylor series

A Second Example: Primality Testing

Problem setting: given an integer n, decide whether n is prime.

Relevance: highly related to the way that we keep information private on the internet (Google RSA Encryption for more on this)

Algorithm (Brute-Force):

- Check whether n is exactly divisible by any integer between 2 and $\lfloor \sqrt{n} \rfloor$ (inclusive)

A Second Example: Primality Testing

Problem setting: given an integer n, decide whether n is prime.

Relevance: highly related to the way that we keep information private on the internet (Google RSA Encryption for more on this)

Algorithm (Brute-Force):

- Check whether n is exactly divisible by any integer between 2 and $\lfloor \sqrt{n} \rfloor$ (inclusive)

Improvements?

A Second Example: Primality Testing

Problem setting: given an integer n, decide whether n is prime.

Relevance: highly related to the way that we keep information private on the internet (Google RSA Encryption for more on this)

Algorithm (Brute-Force):

- Check whether n is exactly divisible by any integer between 2 and $\lfloor \sqrt{n} \rfloor$ (inclusive)

Improvements?

- Need only check whether n divisible by *prime number*. So, can accelerate the method using *recursion* (see later).

A Second Example: Primality Testing

Problem setting: given an integer n, decide whether n is prime.

Relevance: highly related to the way that we keep information private on the internet (Google RSA Encryption for more on this)

Algorithm (Brute-Force):

- Check whether n is exactly divisible by any integer between 2 and $\lfloor \sqrt{n} \rfloor$ (inclusive)

Improvements?

- Need only check whether n divisible by *prime number*. So, can accelerate the method using *recursion* (see later).
- If $n > 2$ and even then not prime, so can restrict to odd numbers and searching between 3 and $\lfloor \sqrt{n} \rfloor$

A Second Example: Primality Testing

Problem setting: given an integer n, decide whether n is prime.

Relevance: highly related to the way that we keep information private on the internet (Google RSA Encryption for more on this)

Algorithm (Brute-Force):

- Check whether n is exactly divisible by any integer between 2 and $\lfloor\sqrt{n}\rfloor$ (inclusive)

Improvements?

- Need only check whether n divisible by *prime number*. So, can accelerate the method using *recursion* (see later).
- If $n > 2$ and even then not prime, so can restrict to odd numbers and searching between 3 and $\lfloor\sqrt{n}\rfloor$
- Many, many more refinements where that came from…

Primality Function

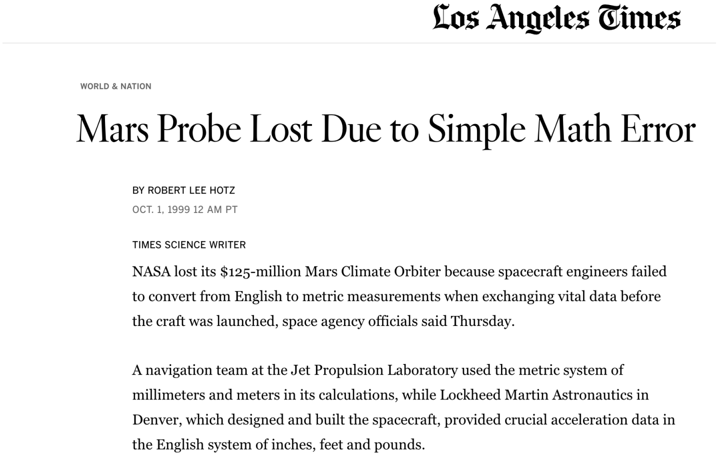
```python
import math
def is_prime(n):
    if isinstance(n, int):
        if n <= 1:
            return False
        elif n == 2:
            return True
        elif n % 2 == 0:
            return False
        else:
            for i in range(3, int(math.sqrt(n)) + 1, 2):
                if n % i == 0:
                    return False
            return True
    else:
        return 'Please ensure n is an Int'
```

Primality Function

```python
import math
def is_prime(n):
    if isinstance(n, int):
        if n <= 1:
            return False
        elif n == 2:
            return True
        elif n % 2 == 0:
            return False
        else:
            for i in range(3, int(math.sqrt(n)) + 1, 2):
                if n % i == 0:
                    return False
            return True
    else:
        return 'Please ensure n is an Int'
```

- Includes a check that *n* is of the correct type, i.e., some error handling in case it isn't.

Why is Checking Inputs Important?



Los Angeles Times

WORLD & NATION

Mars Probe Lost Due to Simple Math Error

BY ROBERT LEE HOTZ

OCT. 1, 1999 12 AM PT

TIMES SCIENCE WRITER

NASA lost its $125-million Mars Climate Orbiter because spacecraft engineers failed to convert from English to metric measurements when exchanging vital data before the craft was launched, space agency officials said Thursday.

A navigation team at the Jet Propulsion Laboratory used the metric system of millimeters and meters in its calculations, while Lockheed Martin Astronautics in Denver, which designed and built the spacecraft, provided crucial acceleration data in the English system of inches, feet and pounds.

(This is about 230M USD in 2024 dollars)

Failing to check inputs for correct units can be expensive!

We'll discuss debugging in more detail later in the class

How to read Python error messages

A traceback is Python telling you: **(1) what failed, (2) where, and (3) why.** When debugging:

- Start at the **bottom line**, which tells you the error (e.g., `NameError`, `TypeError`).
- Then look **one step up**: failing line of code.
- Ask yourself: *what type did Python think this thing was?* (use `type(x)` or print it)

Common beginner errors:

- `NameError`: a variable is not defined (typo or scope issue)
- `TypeError`: a wrong type (e.g., adding a string to a number)
- `IndentationError`: Python can't parse the block structure

Stuck? Some advice

Here are some strategies for if you get stuck:

- Check for typos/spelling/the type of each variable.
- Read each line and ask yourself what it does, then run it in the command prompt and check whether it does what you think it does.
- Read the docs
- Read error messages carefully, and Google if you don’t understand what they mean
- Try to see whether your code solves a particularly simple version of the problem and then go from there.
- For a given input, write out what each step of the answer should be by hand and see whether the command prompt matches what you wrote
- Try to print variables at intermediate points in the code, and see if what they say is what you think they should say

Marking and submission

Tutorial leads will explain how this works—then we will start coding!