

# Week 1: Introduction to Computation

POP77001 Computer Programming for Social Scientists

Tom Paskhalis

12 September 2022

Module website: [tinyurl.com/POP77001](https://tinyurl.com/POP77001)

# Overview

- Computers and Computational thinking
- Algorithms
- Programming languages and computer programs
- Debugging
- Command-line Interfaces
- Version controlling with Git/GitHub

# Computers



1940

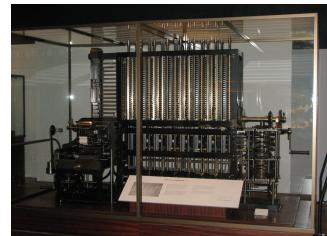


2022

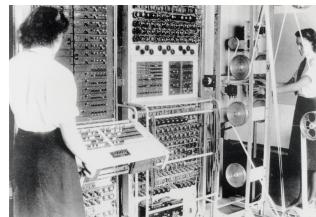
# More Computers



Antikythera mechanism  
(c.100 BC)



Difference Engine  
(1820s)



Collosus (1940s)



Deep Blue (1997)

# Computers

# Computers

- Do two things:

# Computers

- Do two things:

1. Perform calculations

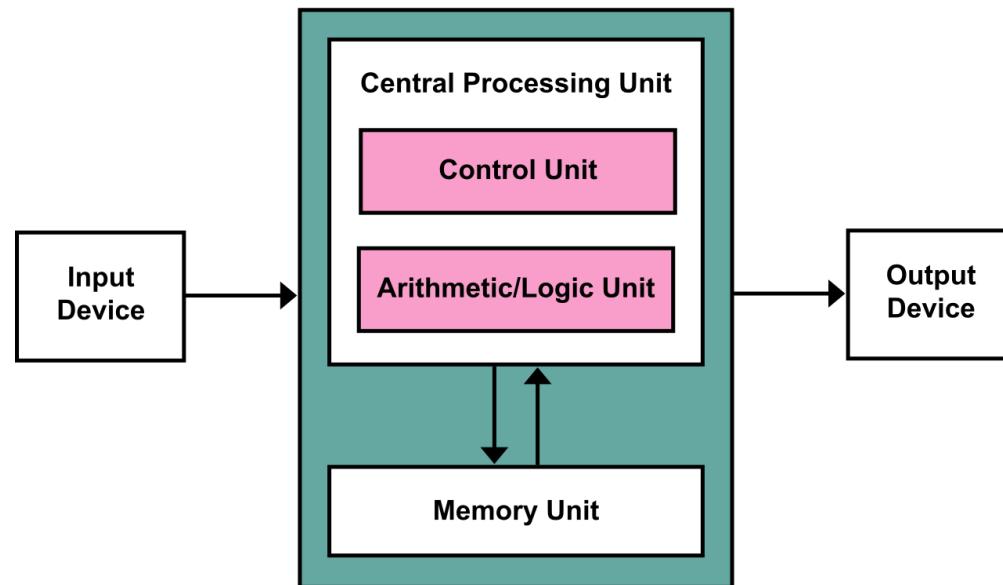
# Computers

- Do two things:

1. Perform calculations

2. Store results of calculations

# von Neumann Architecture



Source: [Wikipedia](#)

# Computational Thinking

Computational thinking is breaking down a problem and formulating a solution in a way that both human and computer can understand and execute.

- *Conceptualizing, not programming* - multiple levels of abstraction
- *A way, that humans, not computers, think* - creatively and imaginatively
- *Complements and combines mathematical and engineering thinking*

Wing, Jeannette M. 2006. "Computational Thinking." Communications of the ACM, 49 (3): 33–35. doi: [10.1145/1118178.1118215](https://doi.org/10.1145/1118178.1118215)

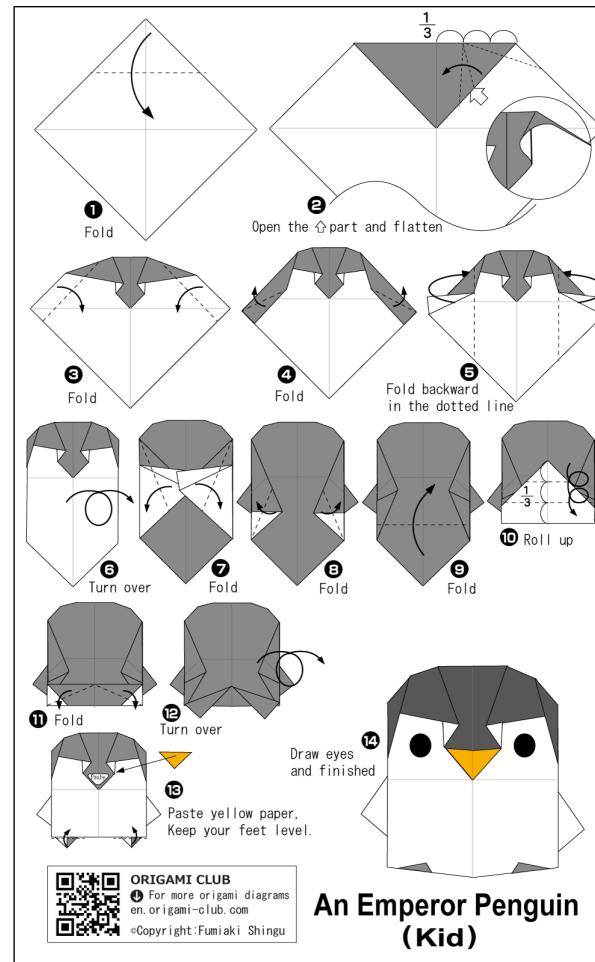
# Computational Thinking

- All knowledge can be thought of as:
  1. Declarative (statement of fact, e.g. square root of 25 equals 5)
  2. Imperative (how to, e.g. to find a square root of x, start with a guess g, check whether  $g*g$  is close, ...)

# Algorithm

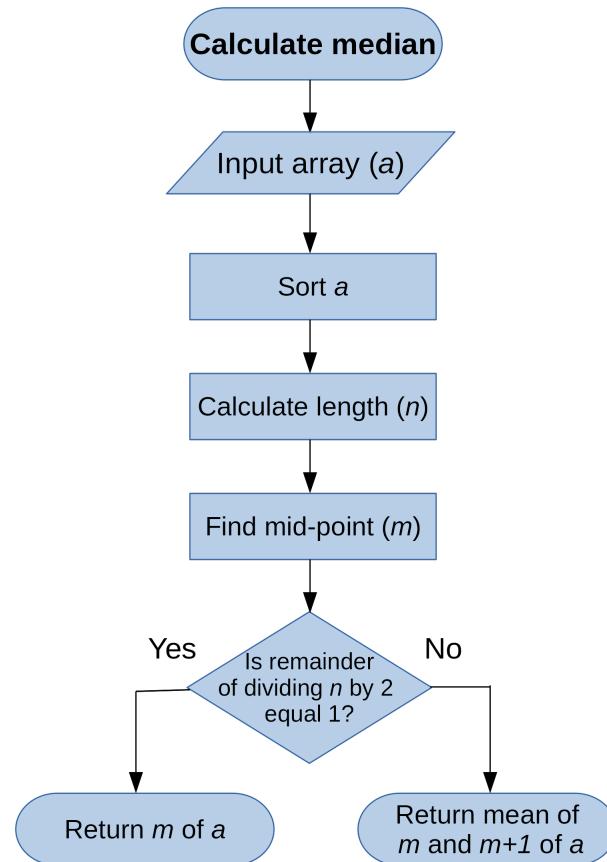
- *Finite list of well-defined instructions that take input and produce output.*
- Consists of a sequence of simple steps that start from input, follow some control flow and have a stopping rule.

# Algorithm Example



Source: [Origami Club](#)

# Algorithm Example



# Programming Language

Formal language used to define sequences of instructions (for computers to execute) that includes:

- Primitive constructs
- Syntax
- Static semantics
- Semantics

# Types of Programming Languages

- Low-level vs high-level
  - E.g. available procedures for moving bits vs calculating a mean
- General vs application-domain
  - E.g. general-purpose vs statistical analysis
- Interpreted vs compiled
  - Source code executed directly vs translated into machine code

# Primitive Constructs in R/Python

- Literals

# Primitive Constructs in R/Python

- Literals

```
In [1]: 77001
```

```
Out[1]: 77001
```

# Primitive Constructs in R/Python

- Literals

```
In [1]: 77001
```

```
Out[1]: 77001
```

```
In [2]: 'POP'
```

```
Out[2]: 'POP'
```

# Primitive Constructs in R/Python

- Literals

```
In [1]: 77001
```

```
Out[1]: 77001
```

```
In [2]: 'POP'
```

```
Out[2]: 'POP'
```

- Infix operators

# Primitive Constructs in R/Python

- Literals

```
In [1]: 77001
```

```
Out[1]: 77001
```

```
In [2]: 'POP'
```

```
Out[2]: 'POP'
```

- Infix operators

```
In [3]: 77001 + 23
```

```
Out[3]: 77024
```

# Syntax in R/Python

- Defines which sequences of characters and symbols are well-formed
- E.g. in English sentence "Cat dog saw" is invalid, while "Cat saw dog" is.

# Syntax in R/Python

- Defines which sequences of characters and symbols are well-formed
- E.g. in English sentence "Cat dog saw" is invalid, while "Cat saw dog" is.

```
In [4]: 77001 + 23
```

```
Out[4]: 77024
```

# Syntax in R/Python

- Defines which sequences of characters and symbols are well-formed
- E.g. in English sentence "Cat dog saw" is invalid, while "Cat saw dog" is.

```
In [4]: 77001 + 23
```

```
Out[4]: 77024
```

```
In [5]: 77001 23 +
```

```
File "<ipython-input-5-7df1296e3e0d>", line 1
 77001 23 +
 ^
SyntaxError: invalid syntax
```

# Static Semantics in R/Python

- Defines which syntactically valid sequences have a meaning
- E.g. in English sentence "Cat seen dog" is invalid, while "Cat saw dog" is.

# Static Semantics in R/Python

- Defines which syntactically valid sequences have a meaning
- E.g. in English sentence "Cat seen dog" is invalid, while "Cat saw dog" is.

```
In [6]: 'POP' + 77001
```

```
-----  
-----  
TypeError  
t call last)
```

Traceback (most recent

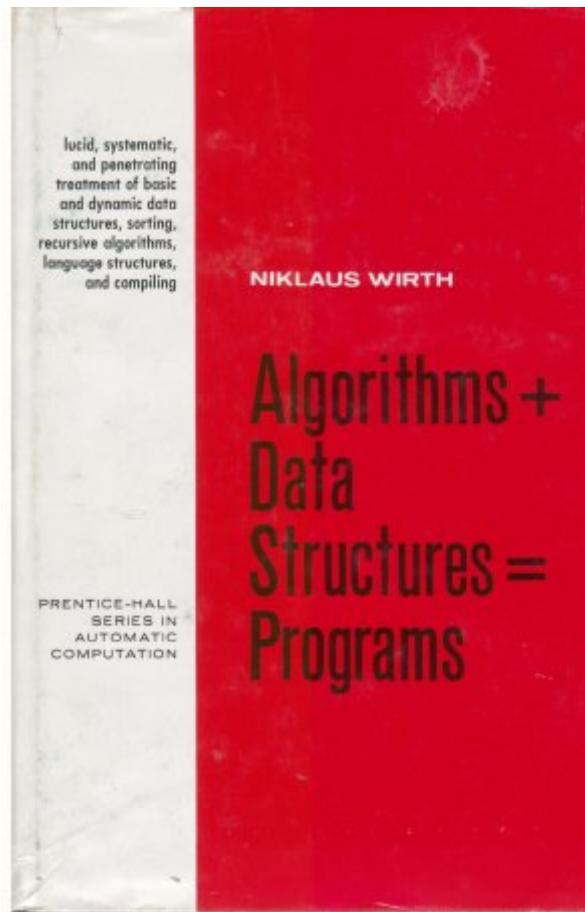
```
<ipython-input-6-7d1b83a55295> in <module>  
----> 1 'POP' + 77001
```

```
TypeError: can only concatenate str (not "int") to str
```

# Semantics in Programming Languages

- Associates a meaning with each syntactically correct sequence of symbols that has no static semantic errors
- Programming languages are designed so that each legal program has exactly one meaning
- **This meaning, however, does not, necessarily, reflect the intentions of the programmer**
- Syntactic errors are much easier to detect

# Algorithms + Data Structures = Programs



# Computer Program

- A collection of instructions that can be executed by computer to perform a specific task
- For interpreted languages (e.g. Python, R, Julia) instructions (source code)
  - Can be executed directly in the interpreter
  - Can be stored and run from the terminal

# Programming Errors

- Often, programs would run with errors or behave in an unexpected way
- Programs might crash
- They might run too long or indefinitely
- **Run to completion and produce an incorrect output**

# Computer Bugs

92

9/9

0800 Antcom started  
1000 " stopped - antcom ✓  
13' sec (032) MP - MC { 1.2700 9.037 847 025  
033 PRO 2 2.130476415 9.037 846 985 correct  
correct 2.130676415  
Relays 6-2 in 033 failed special speed test  
in relay 10.000 test.  
1700 Started Cosine Tape (Sine check)  
1525 Started Multi Adder Test.  
1545 Relay #70 Panel F  
(Moth) in relay.  
1630 First actual case of bug being found.  
1700 closed down.



Grace Murray Hopper popularised the term *bug* after in 1947 her team traced an error in the Mark II to a moth trapped in a relay.

Source: [US Naval History and Heritage Command](#)

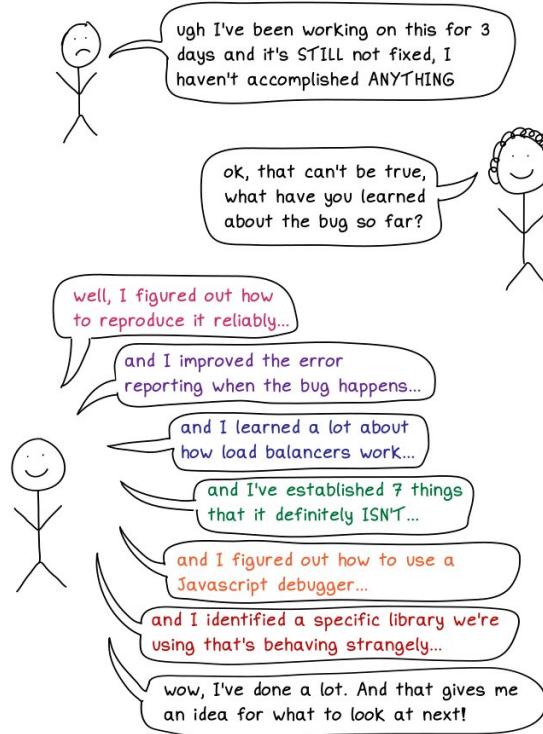
# How to Debug

- Search error message online (e.g. [StackOverflow](#) or, indeed, [#LMDDGTFY](#))
- Insert `print()` statement to check the state between procedures
- Use built-in debugger (stepping through procedure as it executes)
- More to follow!

# Debugging

JULIA EVANS  
@bork track your progress

It's normal to get discouraged while debugging sometimes.



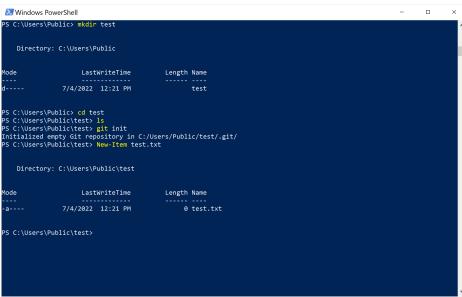
Source: [Julia Evans](#)

# Command-line Interface (aka terminal/console/shell/command line/command prompt)

- Most users today rely on graphical interfaces
- Command line interpreters (CLIs) provide useful shortcuts
- Computer programs can be run or scheduled in terminal/CLI
- **CLI/terminal is usually the only available interface if you work in the cloud (AWS, Microsoft Azure, etc.)**

Extra: [Five reasons why researchers should learn to love the command line](#)

# CLI Examples



```
Windows PowerShell
PS C:\Users\Public> #cd\test
Directory: C:\Users\Public

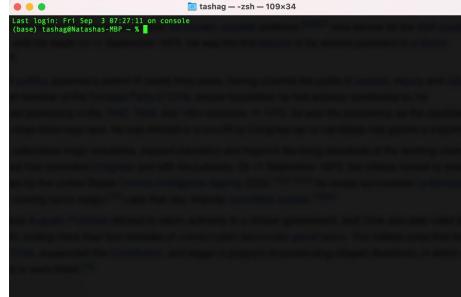
Mode LastWriteTime Length Name
-a---- 7/4/2022 12:31 PM 0 test.txt

PS C:\Users\Public> cd\test
PS C:\Users\Public\test> ls
PS C:\Users\Public\test> git init
Initialized empty Git repository in C:/Users/Public/test/.git/
PS C:\Users\Public\test> New-Item test.txt

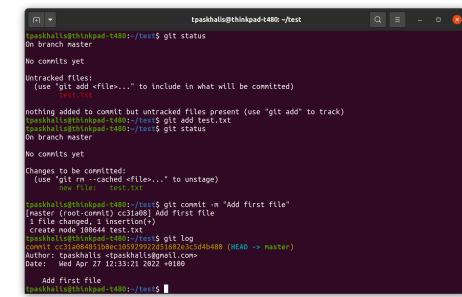
Directory: C:\Users\Public\test

Mode LastWriteTime Length Name
-a---- 7/4/2022 12:31 PM 0 test.txt

PS C:\Users\Public\test>
```



```
tashag -- zsh -- 109x34
Last login: Fri Sep 3 07:27:11 on console
(base) tashag@Natashas-MBP ~ %
```



```
tpaskhalis@thinkpad:~/test$ git status
On branch master
No commits yet
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    .gitignore
nothing added to commit but untracked files present (use "git add" to track)
tpaskhalis@thinkpad:~/test$ git add test.txt
tpaskhalis@thinkpad:~/test$ git status
On branch master
No commits yet
Changes to be committed:
  (use "git add <file>..." to unstage)
    new file:   test.txt
tpaskhalis@thinkpad:~/test$ git commit -m "Add first file"
[master (root-commit) cs10a80] Add first file
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 test.txt
tpaskhalis@thinkpad:~/test$ git log
commit cs10a80 [tpaskhalis@thinkpad ~] - 2022-04-27 22:55:04 +0000 (HEAD -> master)
Author: tpaskhalis <tpaskhalis@gmail.com>
Date:  Wed Apr 27 22:55:04 2022 +0000
  Add first file
tpaskhalis@thinkpad:~/test$
```

Microsoft PowerShell  
(Windows)

Z shell, zsh (macOS)

bash (Linux/UNIX)

# Some Useful CLI Commands

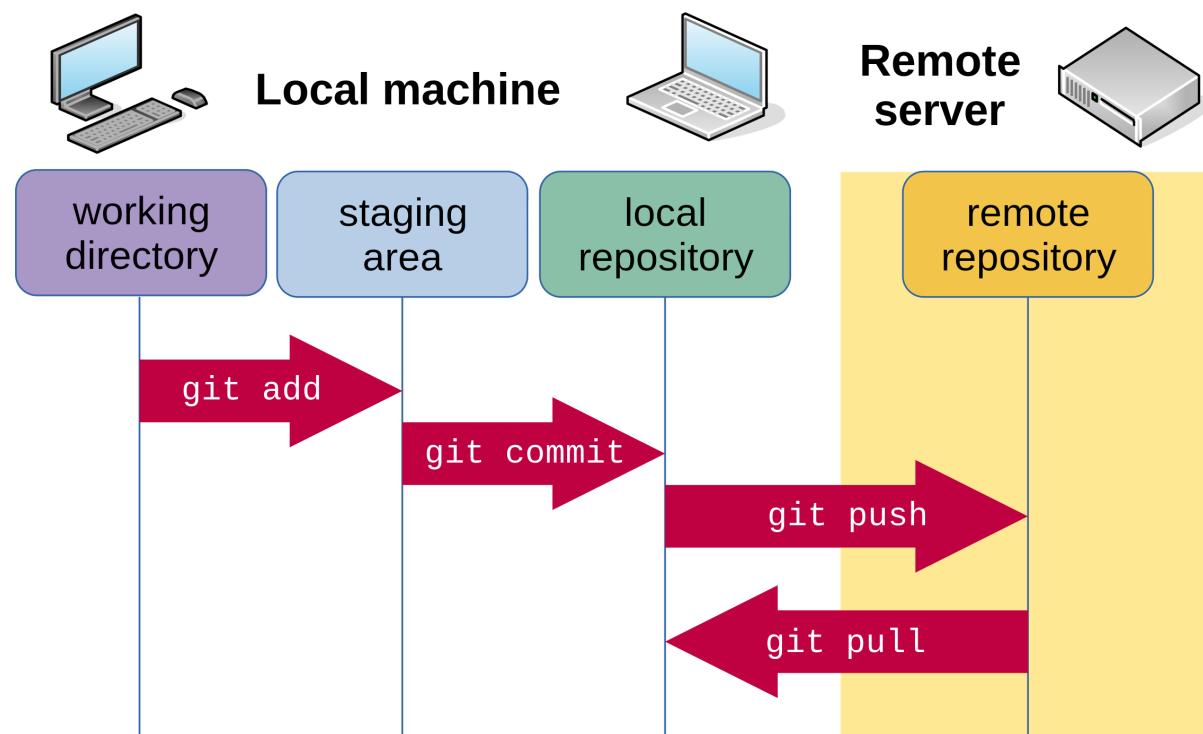
Command (Windows)	Command (macOS/Linux)	Description
exit	exit	close the window
cd	cd	change directory
cd	pwd	show current directory
dir	ls	list directories/files
copy	cp	copy file
move	mv	move/rename file
mkdir	mkdir	create a new directory
del	rm	delete a file

Extra: [Introduction to CLI](#)

# Version Control and Git

- Version control systems (VCSs) allow automatic tracking of changes in files and collaboration
- Git is one of several major version control systems (VCSs, see also Mercurial, Subversion)
- [GitHub](#) is an online hosting platform for projects that use Git for version control

# Git/GitHub Workflow



# Some Useful Git Commands

Command	Description
<code>git init &lt;project name&gt;</code>	Create a new <i>local repository</i>
<code>git clone &lt;project url&gt;</code>	Download a project from <i>remote repository</i>
<code>git status</code>	Check project status
<code>git diff &lt;file&gt;</code>	Show changes between <i>working directory</i> and <i>staging area</i>
<code>git add &lt;file&gt;</code>	Add a file to the <i>staging area</i>
<code>git commit -m "&lt;commit message&gt;"</code>	Create a new <i>commit</i> from changes added to the <i>staging area</i>
<code>git pull &lt;remote&gt; &lt;branch&gt;</code>	Fetch changes from <i>remote</i> and merge into <i>merge</i>
<code>git push &lt;remote&gt; &lt;branch&gt;</code>	Push local branch to <i>remote repository</i>

Extra: [Git Cheatsheet](#)

# Things to Do

- Make sure you have installed R, Python and Jupyter.
- Make sure you can run R code through Jupyter (requires extra steps).
- Do the readings.

# Things to Try (CLI)

- Identify an appropriate CLI for your OS.
- Try navigating across folders and files using CLI.
- Try creating a test folder and test file inside it.

# Things to Try (git)

- Register on [GitHub](#) and [GitHub Education](#) (for free goodies!)
- Create a test repository in CLI and initialise as a Git repository
- Or create a repository on GitHub and clone to your local machine
- Create test.txt file, add it and commit
- Push the file to GitHub

# Next

- Tutorial: Jupyter Notebooks, CLIs, Git/GitHub
- Next week: R Basics