Count Chris - Unsplash

Computers?
(and variables)

What is a computer?

What is a computer?

Something that *computes*

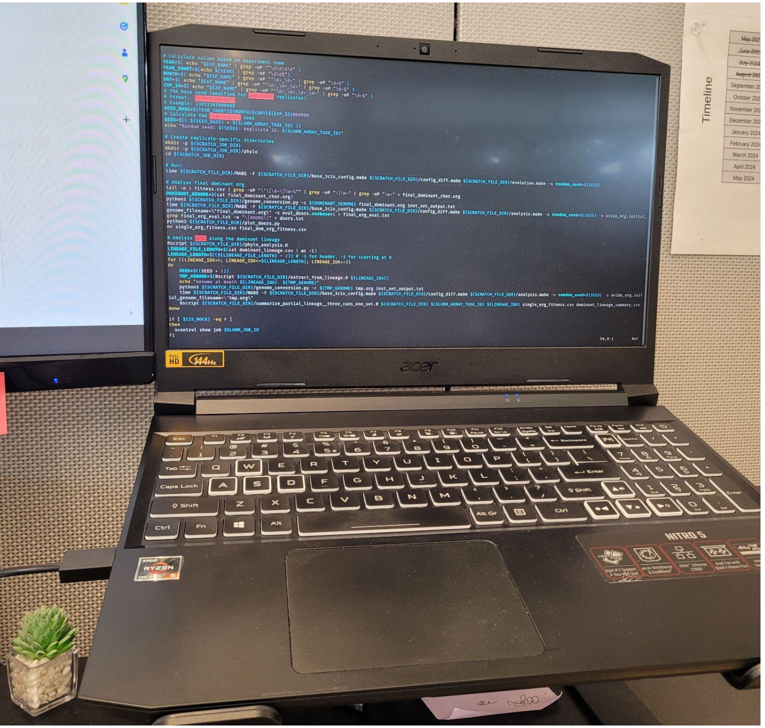
What is a computer?

Something that *computes*

- A machine that performs math calculations or other logic

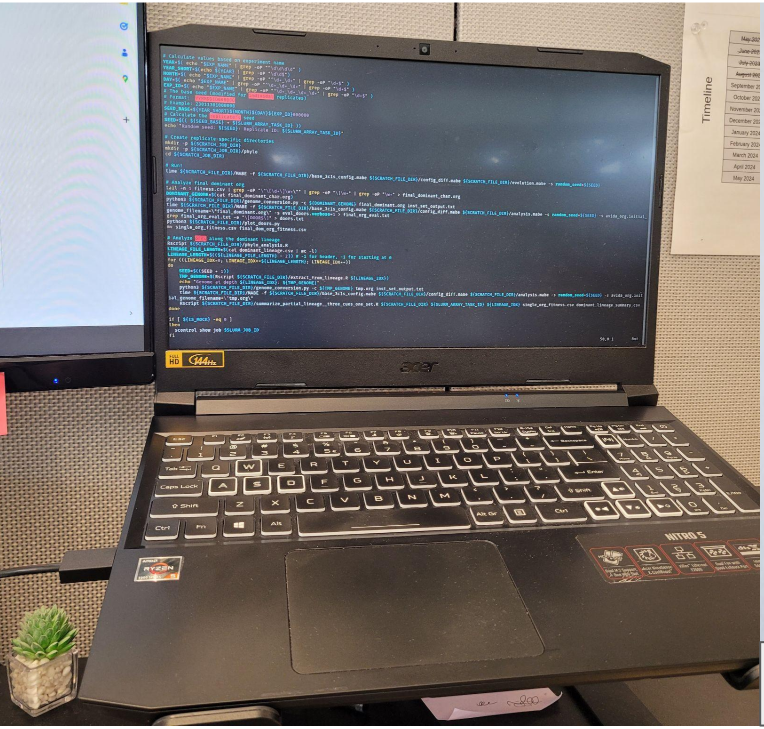
Are these computers?

Are these computers?

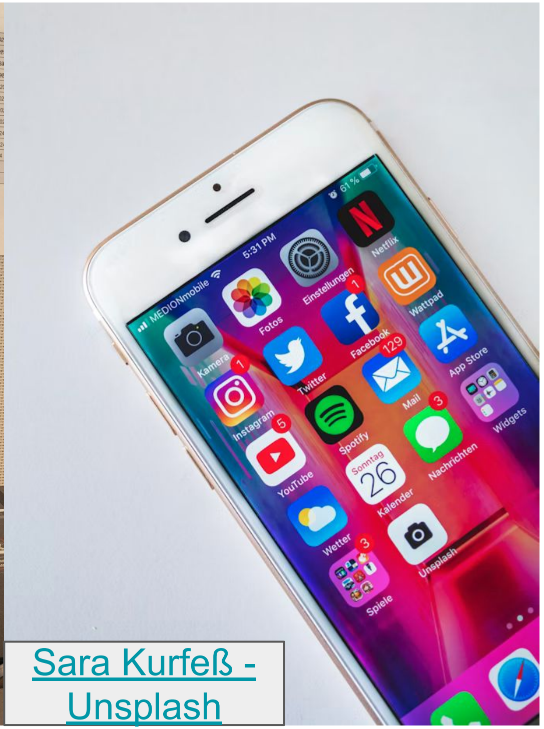


```
# Calculate values based on experiment name
YEAR=$( echo "$EXP_NAME" | grep -oP "^\d\d\d\d" )
YEAR_SHORT=$(echo ${YEAR} | grep -oP "\d\d$")
MONTH=$( echo "$EXP_NAME" | grep -oP "^\d+_\d+" | grep -oP "\d+$" )
DAY=$( echo "$EXP_NAME" | grep -oP "^\d+_\d+_\d+" | grep -oP "\d+$" )
EXP_ID=$( echo "$EXP_NAME" | grep -oP "^\d+_\d+_\d+_\d+" | grep -oP "\d+$" )
# The base seed (modified for individal replicates)
# Format: YYMMDDID000000
# Example: 23011301000000
SEED_BASE=${YEAR_SHORT}${MONTH}${DAY}${EXP_ID}000000
# Calculate the replicate's seed
SEED=$(( ${SEED_BASE} + ${SLURM_ARRAY_TASK_ID} ))
echo "Random seed: ${SEED}: Replicate ID: ${SLURM_ARRAY_TASK_ID}"

# Create replicate-specific directories
mkdir -p ${SCRATCH_JOB_DIR}
mkdir -p ${SCRATCH_JOB_DIR}/phylo
cd ${SCRATCH_JOB_DIR}

# Run!
time ${SCRATCH_FILE_DIR}/MABE -f ${SCRATCH_FILE_DIR}/base_3c1s_config.mabe ${SCRATCH_FILE_DIR}/config_diff.mabe ${SCRATCH_FILE_DIR}/evolution.mabe -s random_seed=${SEED}

# Analyze final dominant org
tail -n 1 fitness.csv | grep -oP "\"\[\d+\]\w+\"" | grep -oP "\]\w+" | grep -oP "\w+" > final_dominant_char.org
DOMINANT_GENOME=$(cat final_dominant_char.org)
python3 ${SCRATCH_FILE_DIR}/genome_conversion.py -c ${DOMINANT_GENOME} final_dominant.org inst_set_output.txt
time ${SCRATCH_FILE_DIR}/MABE -f ${SCRATCH_FILE_DIR}/base_3c1s_config.mabe ${SCRATCH_FILE_DIR}/config_diff.mabe ${SCRATCH_FILE_DIR}/analysis.mabe -s random_seed=${SEED} -s avida_org.initial_
genome_filename=\"final_dominant.org\" -s eval_doors.verbose=1 > final_org_eval.txt
grep final_org_eval.txt -e "\[DOORS\]" > doors.txt
python3 ${SCRATCH_FILE_DIR}/plot_doors.py
mv single_org_fitness.csv final_dom_org_fitness.csv

# Analyze orgs along the dominant lineage
Rscript ${SCRATCH_FILE_DIR}/phylo_analysis.R
LINEAGE_FILE_LENGTH=$(cat dominant_lineage.csv | wc -l)
LINEAGE_LENGTH=$((${LINEAGE_FILE_LENGTH} - 2)) # -1 for header, -1 for starting at 0
for ((LINEAGE_IDX=0; LINEAGE_IDX<=${LINEAGE_LENGTH}; LINEAGE_IDX++))
do
    SEED=$((SEED + 1))
    TMP_GENOME=$(Rscript ${SCRATCH_FILE_DIR}/extract_from_lineage.R ${LINEAGE_IDX})
    echo "Genome at depth ${LINEAGE_IDX}: ${TMP_GENOME}"
    python3 ${SCRATCH_FILE_DIR}/genome_conversion.py -c ${TMP_GENOME} tmp.org inst_set_output.txt
    time ${SCRATCH_FILE_DIR}/MABE -f ${SCRATCH_FILE_DIR}/base_3c1s_config.mabe ${SCRATCH_FILE_DIR}/config_diff.mabe ${SCRATCH_FILE_DIR}/analysis.mabe -s random_seed=${SEED} -s avida_org.init
ial_genome_filename=\"tmp.org\"
    Rscript ${SCRATCH_FILE_DIR}/summarize_partial_lineage__three_cues_one_set.R ${SCRATCH_FILE_DIR} ${SLURM_ARRAY_TASK_ID} ${LINEAGE_IDX} single_org_fitness.csv dominant_lineage_summary.csv
done

if [ ${IS_MOCK} -eq 0 ]
then
  scontrol show job $SLURM_JOB_ID
fi
                                                                                                   50,0-1        Bot
```

Are these computers?

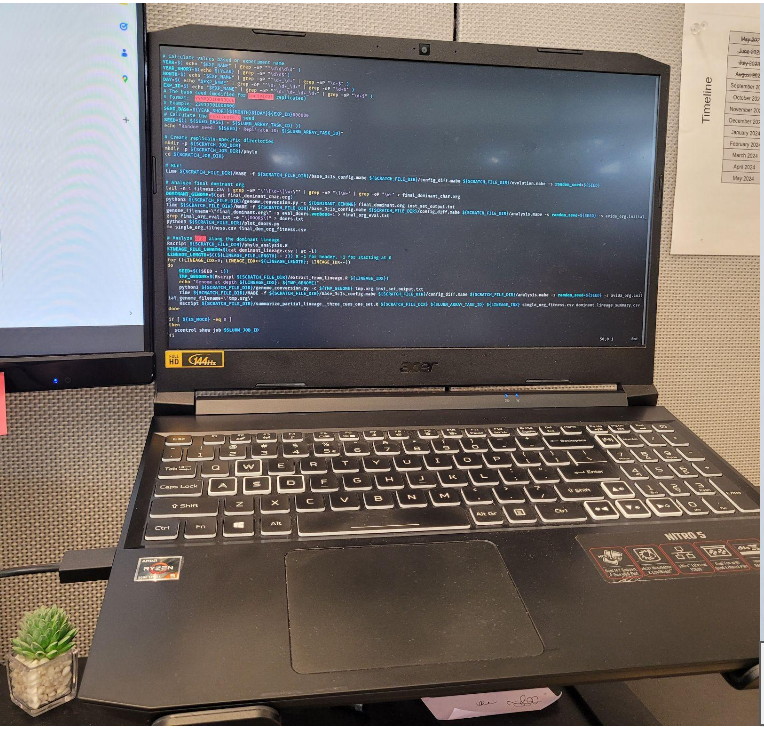


```
# Calculate values based on experiment name
YEAR=$( echo "$EXP_NAME" | grep -oP "^\d\d\d\d" )
YEAR_SHORT=$(echo ${YEAR} | grep -oP "\d\d$")
MONTH=$( echo "$EXP_NAME" | grep -oP "^\d+_\d+" | grep -oP "\d+$" )
DAY=$( echo "$EXP_NAME" | grep -oP "^\d+_\d+_\d+" | grep -oP "\d+$" )
EXP_ID=$( echo "$EXP_NAME" | grep -oP "^\d+_\d+_\d+_\d+" | grep -oP "\d+$" )
# The base seed (modified for individal replicates)
# Format: YYMMDDID000000
# Example: 23011301000000
SEED_BASE=${YEAR_SHORT}${MONTH}${DAY}${EXP_ID}000000
# Calculate the replicate's seed
SEED=$(( ${SEED_BASE} + ${SLURM_ARRAY_TASK_ID} ))
echo "Random seed: ${SEED}: Replicate ID: ${SLURM_ARRAY_TASK_ID}"

# Create replicate-specific directories
mkdir -p ${SCRATCH_JOB_DIR}
mkdir -p ${SCRATCH_JOB_DIR}/phylo
cd ${SCRATCH_JOB_DIR}

# Run!
time ${SCRATCH_FILE_DIR}/MABE -f ${SCRATCH_FILE_DIR}/base_3c1s_config.mabe ${SCRATCH_FILE_DIR}/config_diff.mabe ${SCRATCH_FILE_DIR}/evolution.mabe -s random_seed=${SEED}

# Analyze final dominant org
tail -n 1 fitness.csv | grep -oP "\"\[\d+\]\w+\"" | grep -oP "\]\w+" | grep -oP "\w+" > final_dominant_char.org
DOMINANT_GENOME=$(cat final_dominant_char.org)
python3 ${SCRATCH_FILE_DIR}/genome_conversion.py -c ${DOMINANT_GENOME} final_dominant.org inst_set_output.txt
time ${SCRATCH_FILE_DIR}/MABE -f ${SCRATCH_FILE_DIR}/base_3c1s_config.mabe ${SCRATCH_FILE_DIR}/config_diff.mabe ${SCRATCH_FILE_DIR}/analysis.mabe -s random_seed=${SEED} -s avida_org.initial_
genome_filename=\"final_dominant.org\" -s eval_doors.verbose=1 > final_org_eval.txt
grep final_org_eval.txt -e "\[DOORS\]" > doors.txt
python3 ${SCRATCH_FILE_DIR}/plot_doors.py
mv single_org_fitness.csv final_dom_org_fitness.csv

# Analyze orgs along the dominant lineage
Rscript ${SCRATCH_FILE_DIR}/phylo_analysis.R
LINEAGE_FILE_LENGTH=$(cat dominant_lineage.csv | wc -l)
LINEAGE_LENGTH=$((${LINEAGE_FILE_LENGTH} - 2)) # -1 for header, -1 for starting at 0
for ((LINEAGE_IDX=0; LINEAGE_IDX<=${LINEAGE_LENGTH}; LINEAGE_IDX++))
do
    SEED=$((SEED + 1))
    TMP_GENOME=$(Rscript ${SCRATCH_FILE_DIR}/extract_from_lineage.R ${LINEAGE_IDX})
    echo "Genome at depth ${LINEAGE_IDX}: ${TMP_GENOME}"
    python3 ${SCRATCH_FILE_DIR}/genome_conversion.py -c ${TMP_GENOME} tmp.org inst_set_output.txt
    time ${SCRATCH_FILE_DIR}/MABE -f ${SCRATCH_FILE_DIR}/base_3c1s_config.mabe ${SCRATCH_FILE_DIR}/config_diff.mabe ${SCRATCH_FILE_DIR}/analysis.mabe -s random_seed=${SEED} -s avida_org.init
ial_genome_filename=\"tmp.org\"
    Rscript ${SCRATCH_FILE_DIR}/summarize_partial_lineage__three_cues_one_set.R ${SCRATCH_FILE_DIR} ${SLURM_ARRAY_TASK_ID} ${LINEAGE_IDX} single_org_fitness.csv dominant_lineage_summary.csv
done

if [ ${IS_MOCK} -eq 0 ]
then
  scontrol show job $SLURM_JOB_ID
fi
                                                                                                    50,0-1        Bot
```



Sara Kurfeß - Unsplash

Are these computers?



```
# Calculate values based on experiment name
YEAR=$( echo "$EXP_NAME" | grep -oP "^\d\d\d\d" )
YEAR_SHORT=$(echo ${YEAR} | grep -oP "\d\d$")
MONTH=$( echo "$EXP_NAME" | grep -oP "^\d+_\d+" | grep -oP "\d+$" )
DAY=$( echo "$EXP_NAME" | grep -oP "^\d+_\d+_\d+" | grep -oP "\d+$" )
EXP_ID=$( echo "$EXP_NAME" | grep -oP "^\d+_\d+_\d+_\d+" | grep -oP "\d+$" )
# The base seed (modified for individal replicates)
# Format: YYMMDDID000000
# Example: 23011301000000
SEED_BASE=${YEAR_SHORT}${MONTH}${DAY}${EXP_ID}000000
# Calculate the replicate's seed
SEED=$(( ${SEED_BASE} + ${SLURM_ARRAY_TASK_ID} ))
echo "Random seed: ${SEED}: Replicate ID: ${SLURM_ARRAY_TASK_ID}"

# Create replicate-specific directories
mkdir -p ${SCRATCH_JOB_DIR}
mkdir -p ${SCRATCH_JOB_DIR}/phylo
cd ${SCRATCH_JOB_DIR}

# Run!
time ${SCRATCH_FILE_DIR}/MABE -f ${SCRATCH_FILE_DIR}/base_3c1s_config.mabe ${SCRATCH_FILE_DIR}/config_diff.mabe ${SCRATCH_FILE_DIR}/evolution.mabe -s random_seed=${SEED}

# Analyze final dominant org
tail -n 1 fitness.csv | grep -oP "\"\[\d+\]\w+\"" | grep -oP "\]\w+" | grep -oP "\w+" > final_dominant_char.org
DOMINANT_GENOME=$(cat final_dominant_char.org)
python3 ${SCRATCH_FILE_DIR}/genome_conversion.py -c ${DOMINANT_GENOME} final_dominant.org inst_set_output.txt
time ${SCRATCH_FILE_DIR}/MABE -f ${SCRATCH_FILE_DIR}/base_3c1s_config.mabe ${SCRATCH_FILE_DIR}/config_diff.mabe ${SCRATCH_FILE_DIR}/analysis.mabe -s random_seed=${SEED} -s avida_org.initial_
genome_filename=\"final_dominant.org\" -s eval_doors.verbose=1 > final_org_eval.txt
grep final_org_eval.txt -e "\[DOORS\]" > doors.txt
python3 ${SCRATCH_FILE_DIR}/plot_doors.py
mv single_org_fitness.csv final_dom_org_fitness.csv

# Analyze orgs along the dominant lineage
Rscript ${SCRATCH_FILE_DIR}/phylo_analysis.R
LINEAGE_FILE_LENGTH=$(cat dominant_lineage.csv | wc -l)
LINEAGE_LENGTH=$((${LINEAGE_FILE_LENGTH} - 2)) # -1 for header, -1 for starting at 0
for ((LINEAGE_IDX=0; LINEAGE_IDX<=${LINEAGE_LENGTH}; LINEAGE_IDX++))
do
    SEED=$((SEED + 1))
    TMP_GENOME=$(Rscript ${SCRATCH_FILE_DIR}/extract_from_lineage.R ${LINEAGE_IDX})
    echo "Genome at depth ${LINEAGE_IDX}: ${TMP_GENOME}"
    python3 ${SCRATCH_FILE_DIR}/genome_conversion.py -c ${TMP_GENOME} tmp.org inst_set_output.txt
    time ${SCRATCH_FILE_DIR}/MABE -f ${SCRATCH_FILE_DIR}/base_3c1s_config.mabe ${SCRATCH_FILE_DIR}/config_diff.mabe ${SCRATCH_FILE_DIR}/analysis.mabe -s random_seed=${SEED} -s avida_org.init
ial_genome_filename=\"tmp.org\"
    Rscript ${SCRATCH_FILE_DIR}/summarize_partial_lineage__three_cues_one_set.R ${SCRATCH_FILE_DIR} ${SLURM_ARRAY_TASK_ID} ${LINEAGE_IDX} single_org_fitness.csv dominant_lineage_summary.csv
done

if [ ${IS_MOCK} -eq 0 ]
then
  scontrol show job $SLURM_JOB_ID
fi
                                                                    50,0-1        Bot
```

Sara Kurfe
Unsplash

publicresourceorg - Flickr

Are these computers?

Carlo G Croce - Wikimedia

Logg Tandy - Wikimedia

Are these computers?

Orrery / Astrarium

Carlo G Croce - Wikimedia

Antikythera mechanism

Logg Tandy - Wikimedia

Are these computers? - All yes!

Orrery / Astrarium

Carlo G Croce - Wikimedia

Antikythera mechanism

Logg Tandy - Wikimedia

When we discuss “computers” today, what do we mean?

When we discuss “computers” today, what do we mean?

Digital computers!

When we discuss “computers” today, what do we mean?

Digital computers!

Computers that work *symbolically*

What is the “language of computers”?

What is the “language of computers”?

What is the “language of computers”?

Binary! 1s and 0s

What is the “language of computers”?

Binary! 1s and 0s

Everything a non-quantum computer does is through binary!

Good news!

Good news!

You do NOT need to understand binary for this class!

Wait!!

Wait!!

If *<u>everything</u>* a computer does is through binary, how can we code computers without it???

Wait!!

If *everything* a computer does is through binary, how can we code computers without it???

Layers of abstraction!

Wait!!

If *everything* a computer does is through binary, how can we code computers without it???

Layers of abstraction!

Each layer hides some details to make your life easier!

Wait!!

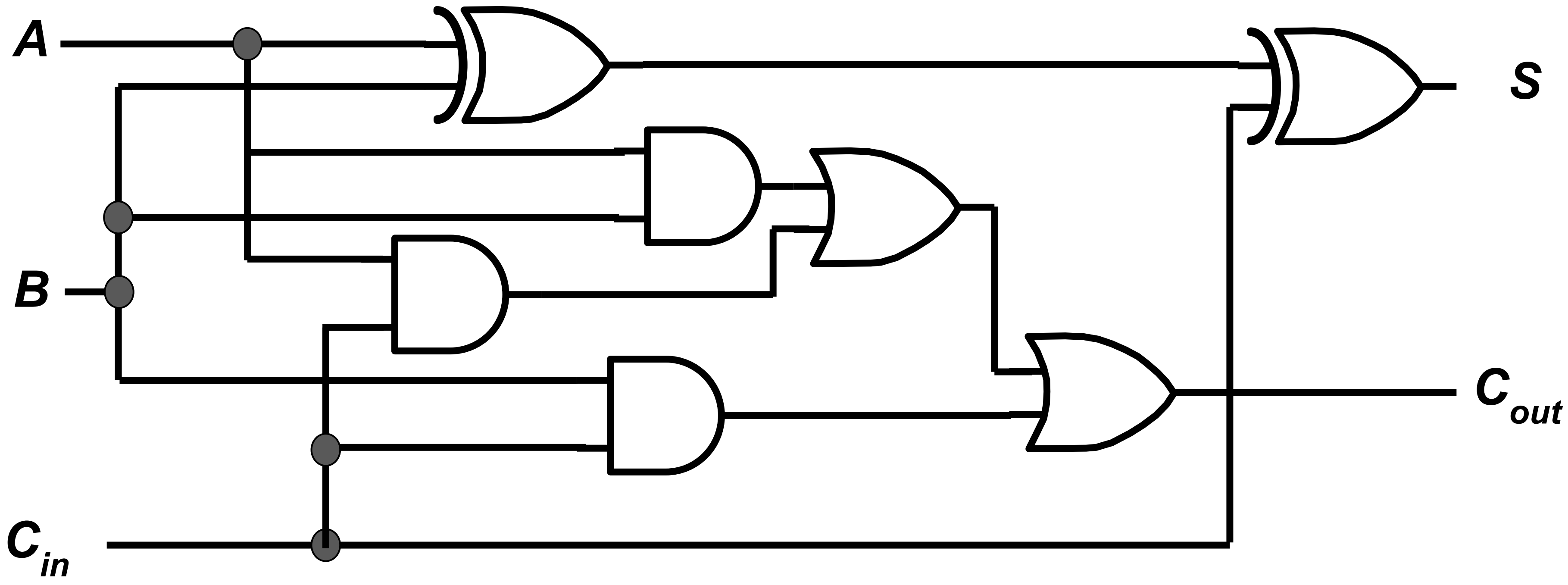
If *everything* a computer does is through binary, how can we code computers without it???

Layers of abstraction!

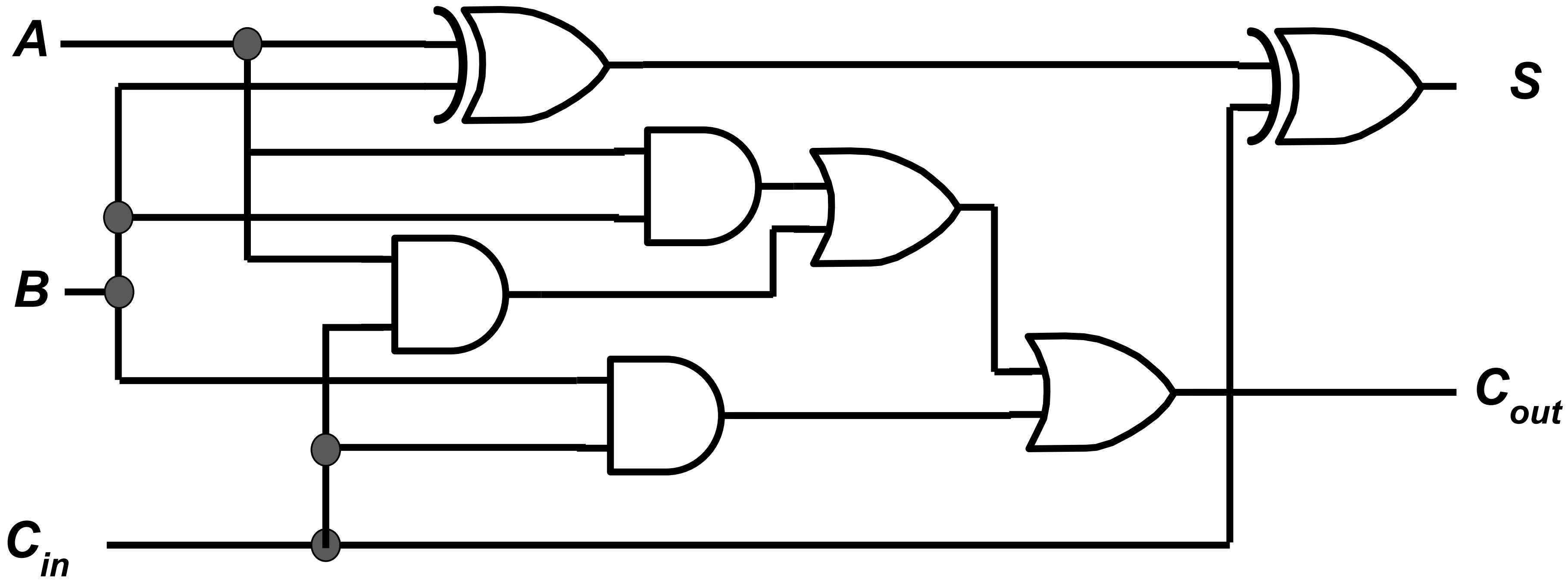
Each layer hides some details to make your life easier!

Hold on!! :^)

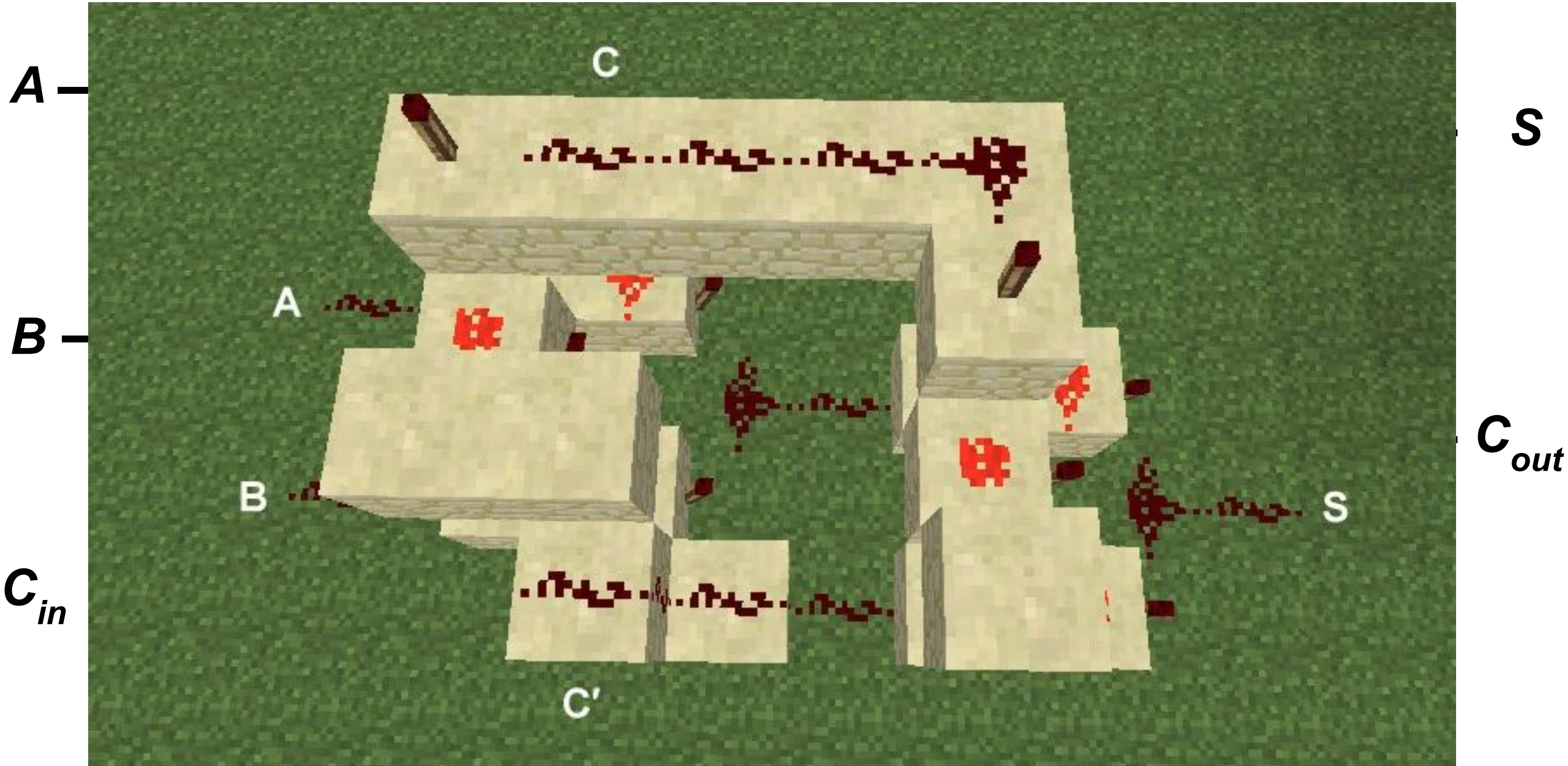
Any idea what this is?



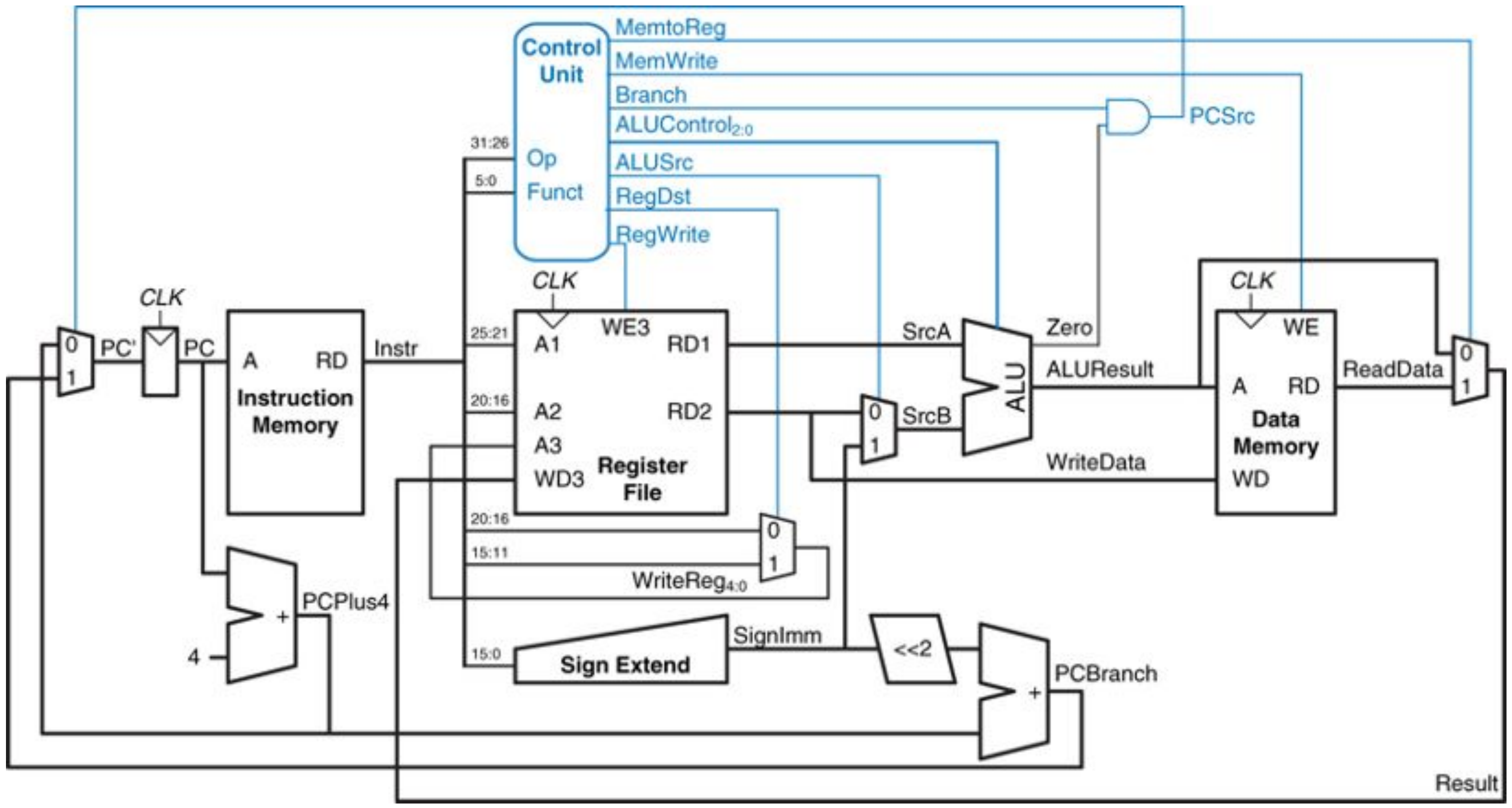
Any idea what this is? An adder circuit!



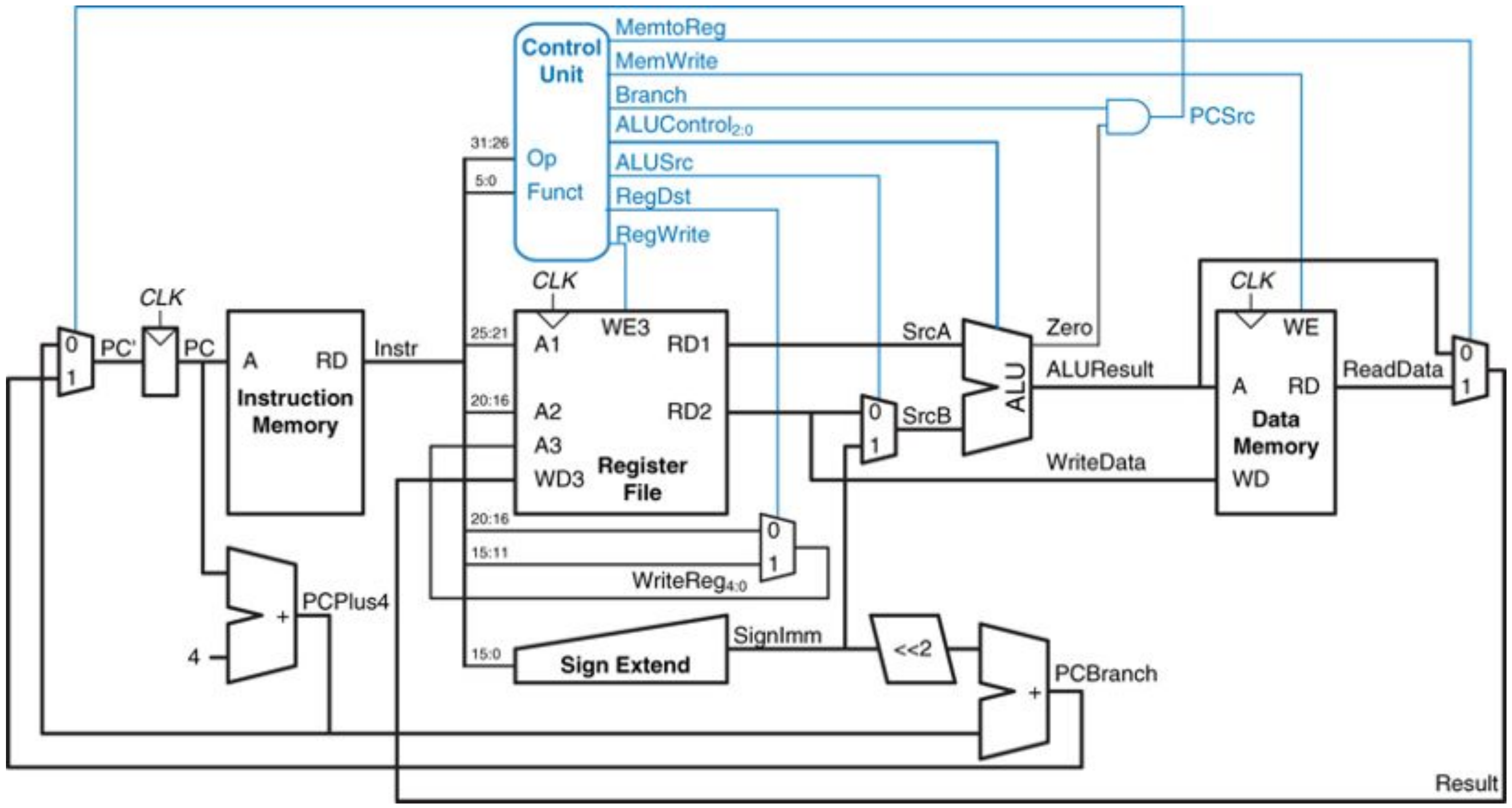
Any idea what this is? An adder circuit!



What about this?



What about this? A simple CPU!



And this?

```
addi  $s0,  $s0,  1
```

And this? Assembly code, adding one to a number

```
addi  $s0,  $s0,  1
```

Last one, I promise!

```
#include <stdio.h>

int main( ){
	int x = 18;
	printf(“Original number is: %d\n”, x);
	x = x + 1;
	printf(“New number is: %d\n”, x);
	return 0;
}
```

Last one, I promise! C code!

```
#include <stdio.h>

int main( ){
        int x = 18;
        printf(“Original number is: %d\n”, x);
        x = x + 1;
        printf(“New number is: %d\n”, x);
        return 0;
}
```

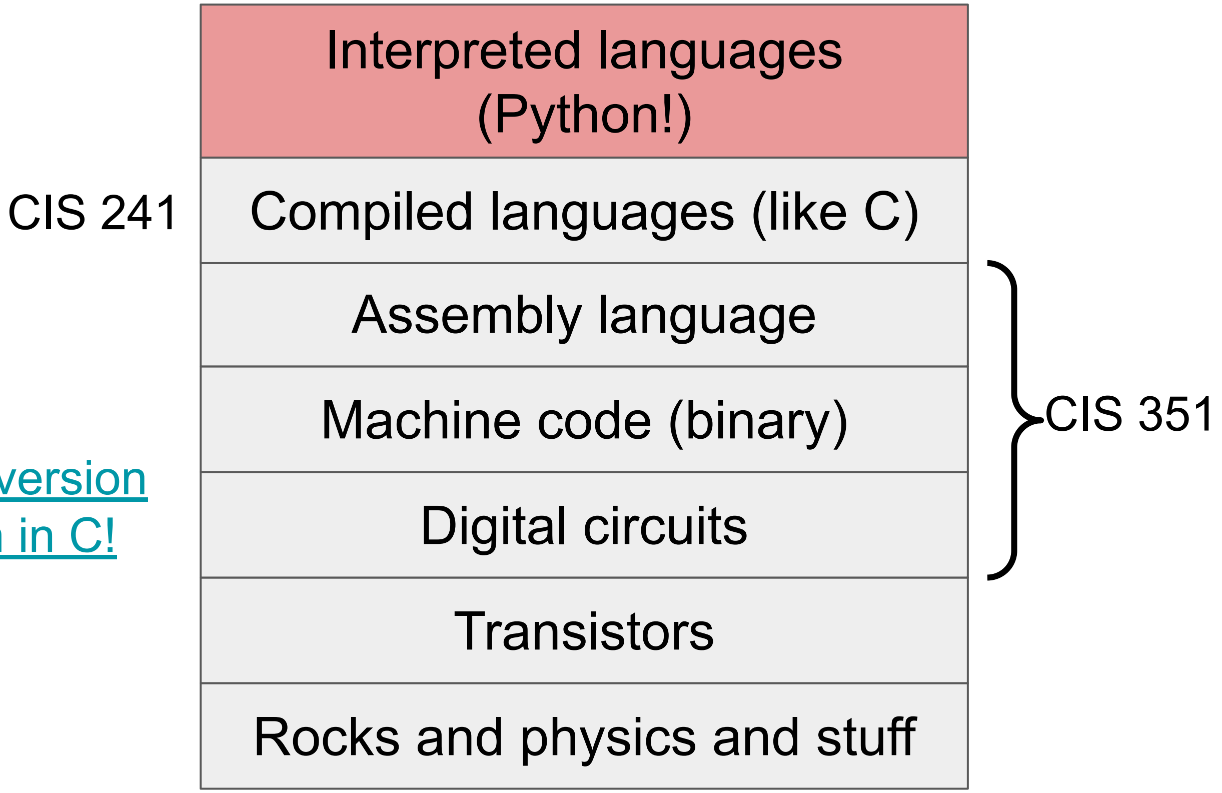
Why tell you all of this?

When we code with Python, we leverage ALL these layers!

When we code with Python, we leverage ALL these layers!

Interpreted languages (Python!)
Compiled languages (like C)
Assembly language
Machine code (binary)
Digital circuits
Transistors
Rocks and physics and stuff

When we code with Python, we leverage ALL these layers!


Interpreted languages (Python!)
CIS 241
Compiled languages (like C)
Assembly language
Machine code (binary)
Digital circuits
CIS 351
Transistors
Rocks and physics and stuff

Python (or one version of it), is written in C!

Shew!

Shew!
Now back to Python…

What *entities* might we represent in a program?

What *entities* might we represent in a program?

- Numbers
- Text
- People
- Places
- Molecules
- Populations
- Shapes
- etc!

How do we represent these things in a computer?

- Numbers are straightforward
- Text is a sequence of letters, numbers, and symbols
- What about something more complex?
    - A (polygon) shape is a *number* of points
    - Each point can be defined as 2D coordinates
        - I.e., with two numbers!

My claim: we can represent any entity as a combination of one or more numbers/text!

We will discuss *how* to do this much later!

We may have a LOT of numbers and text, how do we keep track of it all?

We may have a LOT of numbers and text, how do we keep track of it all?

You also know a lot of people. How do you keep track of them?

We may have a LOT of numbers and text, how do we keep track of it all?

You also know a lot of people. How do you keep track of them?

Names!

We may have a LOT of numbers and text, how do we keep track of it all?

You also know a lot of people. How do you keep track of them?

Names!

In Python, we use *variables,* which have a name, a type, and a value

Assigning variables

```
num_dogs = 2
```

Assigning variables

Name

```
num_dogs = 2
```

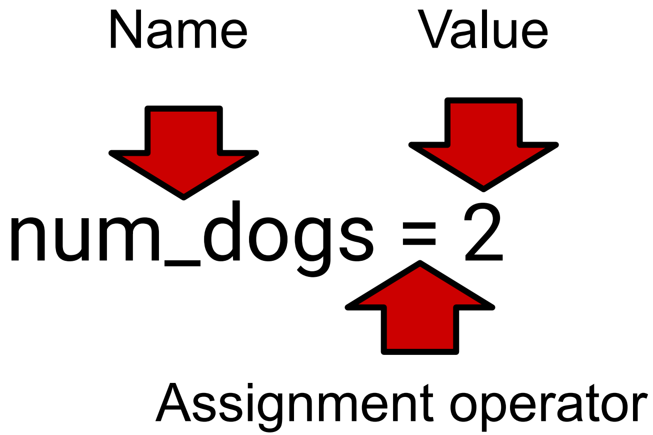
Assigning variables

Name

```
num_dogs = 2
```

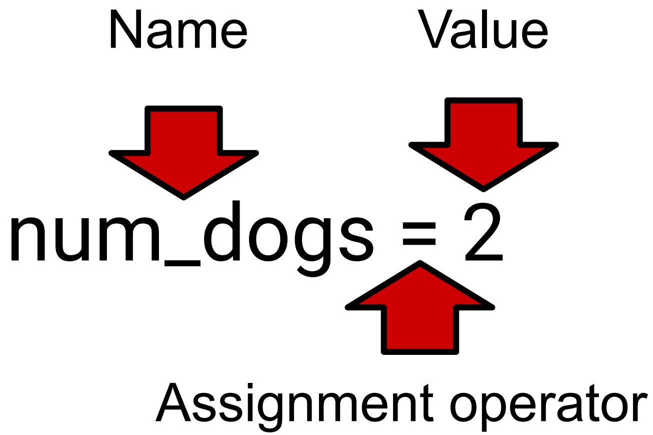
Assignment operator

Assigning variables



```
num_dogs = 2
```

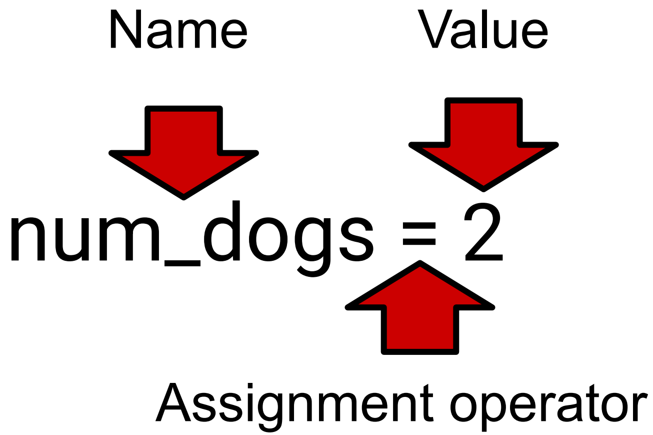
Assigning variables



```
num_dogs = 2
```

What *type* is the variable?

Assigning variables



```
num_dogs = 2
```

What *type* is the variable?
It’s a number, but specifically it is an integer (int) because it has no decimal part!

Other variables

username = “hello 2 u!”

Name:

Other variables

```
username = “hello 2 u!”
```

Name: username

Value:

Other variables

username = “hello 2 u!”

Name: username

Value: “hello 2 u!”

Type:

Other variables

username = “hello 2 u!”

Name: username

Value: “hello 2 u!”

Type: A *string*

Other variables

username = “hello 2 u!”

Name: username

Value: “hello 2 u!”

Type: A *string*

A string is a sequence of characters (letters, numbers, and symbols)

Other variables

```
username = “hello 2 u!”
```

Must begin and end with quotes (double or single)

Name: username
Value: “hello 2 u!”
Type: A ***string***

A string is a sequence of characters (letters, numbers, and symbols)

Even more variables

```
pizzas_remaining = 3.33
```

Even more variables

```
pizzas_remaining = 3.33
```

This is a *float*!

Floats are also numbers, but they can have decimals

Variable names

Variable names should be helpful!!

Variables names:

- Can contain letters, numbers, and underscores (_)
- Cannot start with a number
- Cannot be a “special” word in Python
- Usually are in snake_case
    - Not camelCase or PascalCase

Working with variables

We can print variables to see their value:

```
num_dogs = 2
print(num_dogs)
```

Working with variables

We can print variables to see their value:

```
num_dogs = 2
print(num_dogs)
```

We can also re-assign variables:

```
num_dogs = 0
num_dogs = 1
print(num_dogs)
```

Working with variables

We can use variables in many places!

```
x = 5 + 8
y = x / 2
print(x)
print(y)
```

Working with variables

We can use variables in many places!

```
x = 5 + 8
y = x / 2
print(x)
print(y)
```

```
z = 2 + 8 / 2
```

Working with variables

We can use variables in many places!

```
x = 5 + 8
y = x / 2
print(x)
print(y)
```

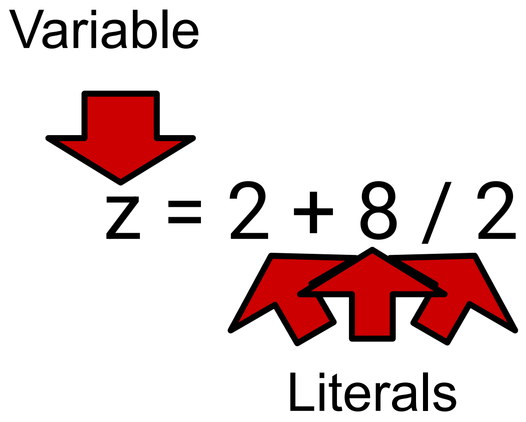
Variable

$$\overset{\downarrow}{z} = 2 + 8 / 2$$

Working with variables

We can use variables in many places!

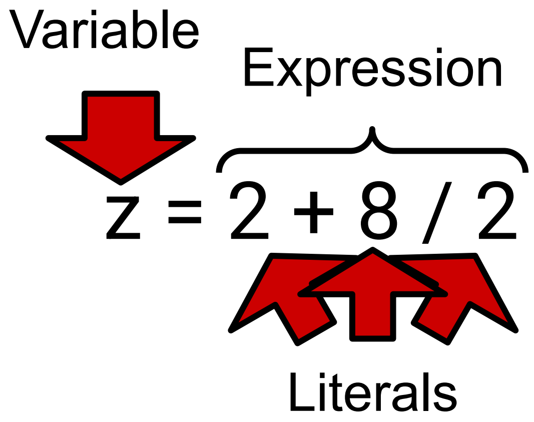
```
x = 5 + 8
y = x / 2
print(x)
print(y)
```



Working with variables

We can use variables in many places!

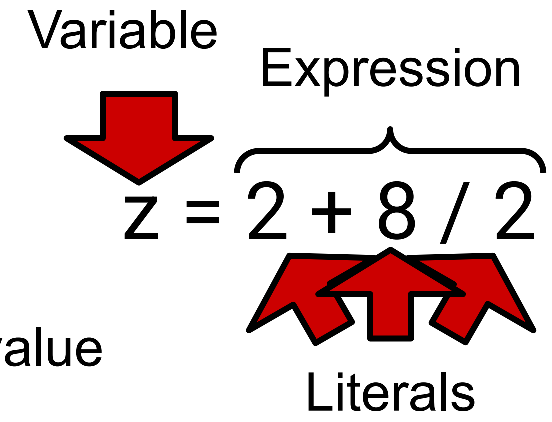
```
x = 5 + 8
y = x / 2
print(x)
print(y)
```



Working with variables

We can use variables in many places!

```
x = 5 + 8
y = x / 2
print(x)
print(y)
```

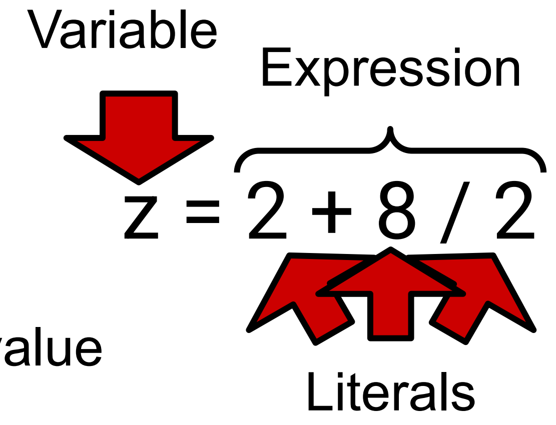


Expressions *evaluate* to a value

Working with variables

We can use variables in many places!

```
x = 5 + 8
y = x / 2
print(x)
print(y)
```



Expressions *evaluate* to a value

Value here?

Working with variables

We can use variables in many places!

```
x = 5 + 8
y = x / 2
print(x)
print(y)
```

Variable

$$z = \overbrace{2 + 8 / 2}^{\text{Expression}}$$

Literals

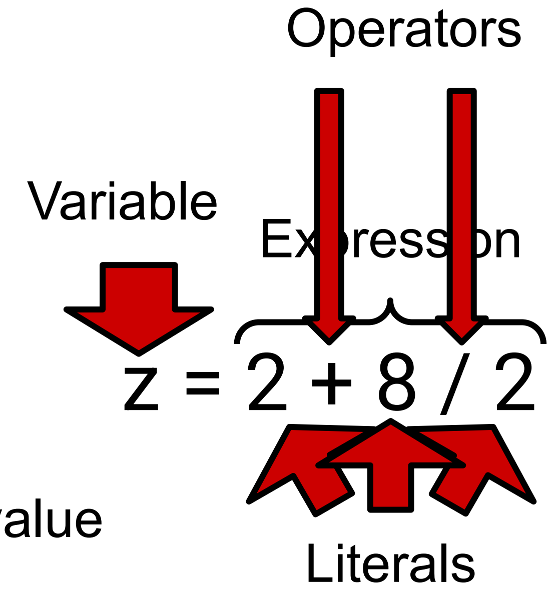
Expressions *evaluate* to a value

Value here? 6

Working with variables

We can use variables in many places!

```
x = 5 + 8
y = x / 2
print(x)
print(y)
```



Expressions *evaluate* to a value

Value here? 6

Math in Python

List of operators with examples:

a + b, Addition, x = 3 + 4 → x is 7

a - b, Subtraction, x = 7 - 2 → x is 5

a * b, Multiplication, x = 2 * 4 → x is 8

a / b, Division, x = 5 / 3 → x is 1.6667

Math in Python

Fancier operators with examples:

a // b, Integer division, x = 5 / 3 → x is 1, ignore remainder

a % b, Modulo,   x = 5 % 3 → x is 2, keep just the remainder

-a, Negation, x = -2 → x is negative 2

a ** b, Exponentiation, x = 7 **2 → x is 49

Math in Python

Order of operations **DO** apply!

Math in Python

Order of operations **DO** apply!

Order of precedence:

1. ( ) (parentheses)
2. ** (exponentiation)
3. Unary - (negation)
4. * (multiply), / (divide), % (modulo), // (integer divide)
5. + (addition), - (subtraction)