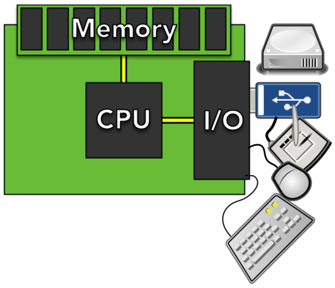
Last time

Introduction to Introduction to Programming

Knowledge and Computation



A simple algorithm

e.g., find the sum of the set $S = 4, 7, 2, 11, 5, 8, 1$

Mathematically:

$$\sum_{i=1}^{n} S_{1..n}$$

Algorithmically?

(do this as an exercise)

Suggested methodology:

- work out an example or two
- break it into steps
- explain the steps to someone else (pretend they're a computer)

Software

Description of instructions for a computer

You express meaning using a *programming language*

Python:

```python
y = x / 2
if (2 * y == x):
    print(x, 'is even')
```

C++:

```cpp
int y = x / 2;
if (2 * y == x)
    cout << x << " is even\n";
```

Also:

Assembly, C, C#, Go, Java, Matlab, Perl, R, Rust, Scala, SPIN...

Your code is translated into *machine instructions*

What is software?

Softer than hardware?

There's also "firmware", but...

Software, whether written in C++, Java, Python or another programming language, is a way of ______________________________. Consider the following implementations of the algorithm from above that checks whether or not a number is even.

You should note that these two code snippets look a bit different, as the details of the two programming languages are different, but the __________________________ ______.

In both the Python and C++ examples, a program has to translate the programmer-readable *source code* into computer-readable *machine code* for the CPU to execute. This process will happen mostly transparently to us as we work with Python via online Python environments or *integrated development environments* (IDEs). I have provided information about getting started with Python tools on the tools page.

Programming languages

vs natural languages

- like natural languages
- unlike natural languages

Syntax and semantics

- *syntax*: rules of **well-formed** language
- *semantics*: the **meaning** of it all

Like natural languages, a medium for expressing semantics

Unlike natural languages, highly constrained (more like math). Allows succinct yet powerful constructions.

Write some software

Yes, right now!

1. Think about a problem, e.g., what is $1 + 2 \times 3 - 4$?
2. Compute an answer
3. Check your answer with Python

Type `1 + 2 * 3 - 4` into pythonmorsels.com/repl, then press Enter

What did you just do?

Wrote an *expression*

Expression was *evaluated*

What is an expression?

Algebra: values, operators that **evaluate** to a **value**

Programming: the same! (even operator precedence)

Exercise 0

Submit a Python expression that evaluates to 42

Submit `.py` file to Gradescope

Not:

```
Python 3.9.1
>>> x = 21
>>> y = 21
>>> x + y
42
```

Or:

```
x = 21
y = 21
print(x + y)
```

Just:

```
21 + 21
```

Or even:

```
42
```

See: "Resources" > "Tools"

Expressions

Values and **operations** that **evaluate** to a **value**

Let's consider each of these words in turn

Literals

Literally mean what they literally say

- `42` : an integer literal
- `3.14` : a real-number (*floating-point*) literal
- `'hello'` : a *string* literal
- `True` : a logical (*Boolean*) literal

Expression: ***values*** *and* ***operations*** *that* ***evaluate*** *to a* ***value***

Integer literals

`1`, `2`, `42`... (ok, so like math!)

`1_000_000` (ok, so a bit like math...)

`0b10`, `0o10`, `0x10` (what!?)

Expression: ***values*** *and* ***operations*** *that* ***evaluate*** *to a* ***value***

You can use underscores in the middle of a Python integer literal to help group numbers and keep things clear. These underscores can go ________________: they're not tied to thousands, so be careful! (e.g., `1_00_000` looks a lot like `1_000_000`, but its meaning is quite different)

We'll come back to what these different ways of writing integers mean when we get to talking about how numbers are represented. For now, just know that there are lots of ways to write integers! (exercise for the keen: what do these "funny" integer literals evaluate to?)

Floating-point literals

`3.0` — is this the same as `3`?

`3.1` — ok, definitely not the same as `3`

`3.1415927` — *definitely* not the same as `3`

(Professor Frink notwithstanding)

Scientific notation: `3.14e0`, `1e100`…

Expression: ***values*** *and* ***operations*** *that* ***evaluate*** *to a* ***value***

Imaginary literals

With integer prefix: `1j`, `2j`, ...

With floating-point prefix: `1.0j`, `1.1j`...

Not *complex* literals, *imaginary* literals

- `1+2j` is actually an *expression*

Expression: ***values*** *and* ***operations*** *that* ***evaluate*** *to a* ***value***

Variables

Named values

Actually, it's slightly more complicated than that, but...

```
>>> from math import *
>>> pi
3.141592653589793
>>> 2j * pi
6.283185307179586j
```

```
r1 + r2
```

Expression: ***values*** *and* ***operations*** *that* ***evaluate*** *to a* ***value***

We'll talk more about variables in later lectures when we talk about how to ____________ them. For now we will just focus on ____________ them.

Expressions

Values and **operations** that **evaluate** to a **value**

~~Values~~ ✔

- ~~literals~~ ✔
- ~~variables~~ ✔

Operations

Operations

Starting with arithmetic operators:

Symbol	Meaning	Usage	Math
+	addition	1 + 2	$1 + 2$
-	subtraction	3 - 4	$3 - 4$
*	multiplication	5 * 6	5×6
/	division	7 / 8	$7 \div 8$ or $\frac{7}{8}$

Evaluation: precedence

Order of operations **matters**

Just like math! (for now)

Operation	Kind
( )	parenthetical
∗, /	multiplicative
+, −	additive

Top Hat question: Order of Operations (literals only)

Division operator

$x \div y$ or $y\overline{)x}$

Integers and real (*floating-point*) numbers

```
>>> 7 / 2
3.5
```

```
>>> 7 // 2
3
>>> 7 % 2
1
```

We can perform division on ________________ and _____________________. It doesn't make sense to divide, say, a string and an integer. Syntactically, `x / y` is valid, but it could be semantically nonsense.

Q: what is 7 ÷ 2?

How about in long division?

When performing long division, we will often leave the result as 3 with a remainder of 1.