CS 131 Discussion

Week 4: Type Tantrum

matt wang (he/him)
✉️ matt@matthewwang.me
🔗 mattxw.com/131slides
📣 mattxw.com/131feedback

Discussion Agenda

1. feedback from last week
2. warmup
3. selected hw answers
4. this week's hw // javascript *gone wrong*
5. person of the week

   ~ break ~

6. language of the week
7. project tips
8. project q&a // help session

snacc of the week

Feedback / Iteration

Thank you for giving feedback!

Some things I'm working on this week:

- **continuing last week's pacing**
- this week: "learning a language"
- PDF export for notes:
  - how do we feel about the "print" option on a page?

And a broader point:

- the project, autograder, and recursion limit –
  we know it's been tricky; we're learning with you!

Idk where to find the discussion slides :(

🔗 mattxw.com/131slides
📣 mattxw.com/131feedback

and **the course website has everything!!**

warmup

(subtypes)

Q: What is a subtype?

Answer: Given two types, A and B:

A is a **subtype** of **B if and only if:**

-
-

Q: What is a subtype?

Answer: Given two types, A and B:

A is a **subtype** of **B if and only if:**

- An object of type **A** can be used in *any* code that requires an object of type **B**
- Every element in the values of **A** is *also* in the values of **B**

Q: What is a subtype?

Answer: Given two types, A and B:

A is a **subtype** of **B if and only if:**

- An object of type **A** can be used in *any* code that requires an object of type **B**
- Every element in the values of **A** is *also* in the values of **B**

We can also say that **B is a supertype of A.**

Note: while subtyping and inheritance may seem related,
they aren't guaranteed to be!

Okay, so which of these are subtypes?

In C++ (or similar),

- `int, long`
  - ?
- `long, float`
  - ?

In Python,

- `bool, int`
  - ?

Okay, so which of these are subtypes?

In C++ (or similar),

- `int, long`
  - **Yes!** All `int` vals are also `long` vals, and anything you can do to a `long` you can to an `int`!
- `long, float`
  - ?

In Python,

- `bool, int`
  - ?

Okay, so which of these are subtypes?

In C++ (or similar),

- `int, long`
  - **Yes!** All `int` vals are also `long` vals, and anything you can do to a `long` you can to an `int`!
- `long, float`
  - **No!** `9999999999999999` is a `long` value but can't be represented by `float`; `3.5` vice-versa!

In Python,

- `bool, int`
  - ?

Okay, so which of these are subtypes?

In C++ (or similar),

- `int, long`
  - **Yes!** All `int` vals are also `long` vals, and anything you can do to a `long` you can to an `int`!
- `long, float`
  - **No!** `9999999999999999` is a `long` value but can't be represented by `float`; `3.5` vice-versa!

In Python,

- `bool, int`
  - **Yes!** With `True = 1`, `False = 0`, Python coerces all relevant operations!

Okay, so which of these are subtypes?

In C++ (or similar),

- `int, long`
  - **Yes!** All `int` vals are also `long` vals, and anything you can do to a `long` you can to an `int`!
- `long, float`
  - **No!** `9999999999999999` is a `long` value but can't be represented by `float`; `3.5` vice-versa!

In Python,

- `bool, int`
  - **Yes!** With `True = 1`, `False = 0`, Python coerces all relevant operations!

Challenge: what about `int[], long[]`?

last week’s homework!

(abridged)

People seemed to struggle with...

- Q1: bit logic
- **Q2: doing this *with comprehensions* + list unpacking**
  - please read the question!!!
- Q4: pass-by-object-reference
- **Q5: dynamic versus duck typing**

Suggestion: review these ones in particular when it's midterm time!

Question 2

a) Write a function named `parse_csv` that takes in a list of strings named `lines`. Each string in `lines` contains a word, followed by a comma, followed by some number (e.g. `"apple,8"`). Your function should return a new list where each string has been converted to a tuple containing the word and the integer (i.e. the tuple should be of type `(string, int)`). **Your function's implementation should be a single, one-line nested list comprehension**.

Example:
`parse_csv(["apple,8", "pear,24", "gooseberry,-2"]` should return `[("apple", 8), ("pear", 24), ("gooseberry", -2)]`.

Hint: You may find [list unpacking] useful.

Question 2

Example:
`parse_csv(["apple,8", "pear,24", "gooseberry,-2"]`

should return
`[("apple", 8), ("pear", 24), ("gooseberry", -2)]`.

```
def parse_csv(lines):
  return [(x, int(y)) for x, y in [line.split(",") for line in lines]]
```

Question 5

Consider the following output from the Python 3 REPL:

```
>>> class Foo:
...   def __len__(self):
...     return 10
...
>>> len(Foo())
10
>>> len("blah")
4
>>> len(5)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: object of type 'int' has no len()
```

Explain why Python allows you to pass an object of type `Foo` or `str` to `len`, but not one of type `int`. Your answer should explicitly reference Python's typing system.

Question 5

Consider the following output from the Python 3 REPL:

```
>>> class Foo:
...   def __len__(self):
...     return 10
...
>>> len(Foo())
10
>>> len("blah")
4
>>> len(5)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: object of type 'int' has no len()
```

This feature does not require dynamic typing!!

(traits, interfaces, etc.)

Explain why Python allows you to pass an object of type `Foo` or `str` to `len`, but not one of type `int`. Your answer should explicitly reference Python's typing system.

Question 5

Consider the following output from the Python 3 REPL:

```
>>> class Foo:
...   def __len__(self):
...     return 10
...
>>> len(Foo())
10
>>> len("blah")
4
>>> len(5)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: object of type 'int' has no len()
```

Saying "5 doesn't have a length" is correct, but **does not explicitly reference the type system!**

Explain why Python allows you to pass an object of type `Foo` or `str` to `len`, but not one of type `int`. **Your answer should explicitly reference Python's typing system.**

Question 5, Bigger Picture

Dynamic and duck typing **are not the same!** In this class:

- dynamic: variable types are checked *at runtime*
- duck: type equivalence is determined by available methods – not name

In fact:

- Statically typed languages – like D – support duck typing
  - In some languages (C++?), templates could be duck typing :o
- Dynamically typed languages – like Julia – can disallow duck typing (nominal-first)

Other interesting keywords: **nominal typing**, **structural typing**.

Even Bigger Picture

Some tips for the midterm:

- if we ask you: in reference to ___, **we're expecting you to namedrop some concepts**
  - ex: pass by object reference, duck typing, lexical scoping, pattern matching
  - your explanation still *also* needs to be correct!
- if we ask you: your function's implementation should use ___, **please use the thing!!**
  - generally, there is no "trick"

Broadly, I think people are struggling more with the "one-liner" questions and conceptual free-response than the "code a bunch of stuff" questions. Focus on those for the midterm!

overview: this week's HW!

- type systems
- scoping and binding
- ~~memory management~~ (rip)

Matt wrote **all** the questions so … roast him!

doing things a lil' differently...

javascript *gone wrong**

(and how to infer language behaviour by example)

aside: matt has given variations of this talk a couple of times!

two key goals:

1. learn a teensy bit about JavaScript
2. reasoning about language behaviour,

   even without all the info!

related: hw & midterm problems :)

and … job interviews, internships, etc.

quick baseline

JavaScript is...

- C-like in syntax
- designed to be “error resistant”

(and many other things, but we don’t care!)

Base assumption: you’re in this class and remember stuff from CS 33.

No JS or PL background required!

warmup - numbers

```
> typeof 18
'number'
> typeof 18.0
'number'
```

so far...

JavaScript:

- seems to not differentiate between integers and floats

Let’s do some more spelunking!

equality, 1

```
> 1 + 2 == 3
```

What we know @ JS:

Integers and floats seem similar?

equality, 1

```
> 1 + 2 == 3
true
```

What we know @ JS:

Integers and floats seem similar?

equality, 1

```
> 1 + 2 == 3
true
> 0.1 + 0.2 == 0.3
```

What we know @ JS:

Integers and floats seem similar?

equality, 1

```
> 1 + 2 == 3
true
> 0.1 + 0.2 == 0.3
false
```

What we know @ JS:

Integers and floats seem similar?

equality, 1

```
> 0.1 + 0.2 == 0.3
false
> 0.1 + 0.2
0.30000000000000004
```

What we know @ JS:

Integers and floats seem similar?

interlude

```
> 99999999999999999999
100000000000000000000
```

What we know @ JS:

Integers and floats seem similar?

so far...

JavaScript:

- ~~seems to not differentiate between integers and floats~~
- **does not have Integers - all numbers are floats!**

operators, 1

```
> 3 - '3'
```

What we know @ JS:

All numbers are floats

operators, 1

```
> 3 - '3'
0
> 3 + '3'
```

What we know @ JS:

All numbers are floats

operators, 1

```
> 3 - '3'
0
> 3 + '3'
'33'
```

What we know @ JS:

All numbers are floats

so far...

JavaScript:

- does not have Integers - all numbers are floats!
- **coerces types in operations**
  - **specifically, prioritizes string coercion**

operators, 2

```
> x = 3
3
> '5' + x - x
```

What we know @ JS:

All numbers are floats

Coerces types in ops; priorities strings

operators, 2

```
> x = 3
3
> '5' + x - x
50
```

What we know @ JS:

All numbers are floats

Coerces types in ops; priorities strings

operators, 2

```
> x = 3
3
> '5' + x - x
50
> '5' - x + x
```

What we know @ JS:

All numbers are floats

Coerces types in ops; priorities strings

operators, 2

```
> x = 3
3
> '5' + x - x
50
> '5' - x + x
5
```

What we know @ JS:

All numbers are floats

Coerces types in ops; priorities strings

so far...

JavaScript:

- does not have Integers - all numbers are floats!
- coerces types in operations
  - specifically, prioritizes string coercion
  - **operations, and thus coercion, happens from left-to-right**

brain tickler

```
> '5' + - '2'
```

What we know @ JS:

All numbers are floats

Coerces types in ops; priorities strings

Operations + coercion are left-to-right

brain tickler

```
> '5' + - '2'
'5-2'
```

What we know @ JS:

All numbers are floats

Coerces types in ops; priorities strings

Operations + coercion are left-to-right

brain tickler

```
> '5' + - '2'
'5-2'
> '5' + - + - - + - - + + - + - + - + - - - '2'
```

brain tickler

```
> '5' + - '2'
'5-2'
> '5' + - + - - + - - + + - + - + - + - - - '2'
'5-2'
> '5' + - + - - + - - + + - + - + - - - '2'
'52'
```

in closing...

JavaScript:

- does not have Integers - all numbers are floats!
- coerces types in operations
  - specifically, prioritizes string coercion
  - operations, and thus coercion, happens from left-to-right
- **??????????**
  - hint: unary operators!

and, bigger picture…

Try things!

- REPLs and type introspection (`:t` in Haskell, `typeof` in JS) are your friend!
- IMO, trying things *builds intuition* – not reading Wikipedia articles

You know more than you think!

- your mental models from this class – Python, C++, Haskell, etc. – work!
- some models are set in stone, some aren't
  - Ex: floats? Standard (literally). Type coercion rules? Lots of variance!

person of the week

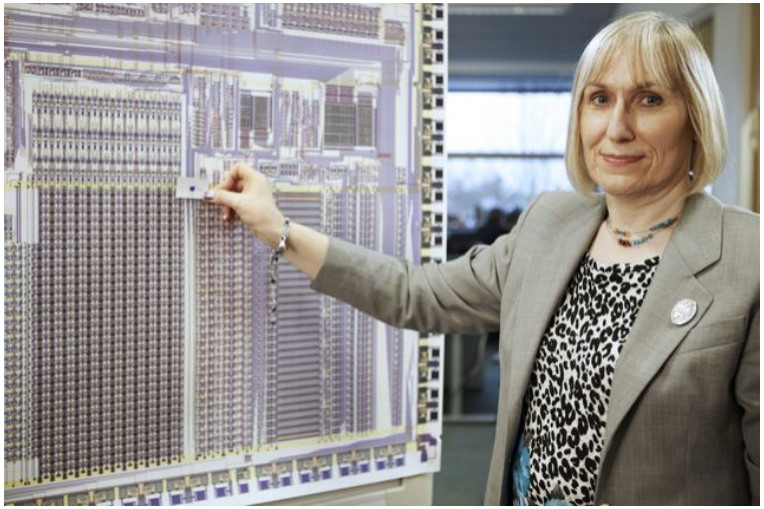
🚧🚧🚧🚧🚧

Sophie Wilson (1957-)

- Prolific architecture / assembly language designer
  - **Designed the ARM RISC!**
  - ARM is used in almost all smartphones, M1s
  - Also helped create BBC BASIC
- Worked at Acorn Computers & ARM Ltd
  - **ARM**: **A**corn **R**ISC **M**achine
- Transitioned in 1994 🏳️‍⚧️

Why Sophie Wilson?

- Much of Brewin' is inspired by early languages (BBC BASIC) and assembly code (ARM)!
- You don't often hear about trans engineers :(



Sophie Wilson with a photograph of the first ARM processor, and an equivalently-powerful processor now.

Chris Monk / Wikipedia, 2013.

```
90 REPEAT

100 ON D GOSUB 160,260,350,430

110 IF RND(1000)<10 THEN R=D-1

120 GCOL R,(D*1.7)

130 DRAW X,Y

140 UNTIL FALSE
```

BBC Basic Code.

What control flow is happening here?

```
90 REPEAT
100 ON D GOSUB 160,260,350,430
110 IF RND(1000)<10 THEN R=D-1
120 GCOL R,(D*1.7)
130 DRAW X,Y
140 UNTIL FALSE
```

BBC Basic Code.

This is a while loop!

(well, an infinite do-while)

Acorn BASIC *did not have these!* You had to manually make them with GOTO statements.

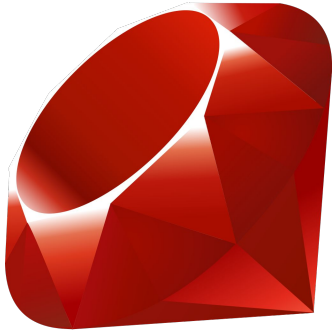
In Sophie's BBC BASIC, she added while loops, named functions, "if-then-else", and integer math!

~ break ~

discussion will resume at 11:00

language of the week

Sorbet (and **a bit of Ruby**)



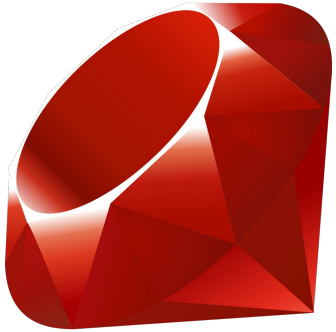
A little about Ruby (we'll come back to this later!):

- interpreted, **dynamically typed** (but strong!)
- heavy support for introspection / metaprogramming

... sounds like Python :o

(and in fact, Python and Ruby often inspire each other – including their extensions!!)

Sorbet (and a bit of Ruby)



More on **Sorbet** (and why now?):

- **static + runtime type system for Ruby**
- **implements gradual typing**
- autogenerates types for library files with introspection!!!
    - *no expectation for you to know what this means lol*
- implements IDE-level type support!!

Sorbet is mainly developed at **Stripe** and **Shopify**.
Many Ruby companies (ex GitHub, Coinbase, Flexport) use it!

- But, Matt did some OSS Sorbet work at CZI :’)))))

Sorbet catches type errors statically - in your editor!

```
sig {params(name: String).returns(Integer)}
def main(name)
  puts "Hello, #{name}!"
  name.length
end

main("Sorbet") # ok!
main()   # error: Not enough arguments provided
man("")  # error: Method `man` does not exist
```

Sorbet catches type errors statically - in your editor!

```
sig {params(name: String).returns(Integer)}
def main(name)
  puts "Hello, #{name}!"
  name.length
end

main("Sorbet") # ok!
main()   # error: Not enough arguments provided
man("")  # error: Method `man` does not exist
```

Developers write this annotation

the secret sauce?

Sorbet guesses types for **100k+ LoC libraries** with *only* code examples, **without manual intervention!**

but, the “how” is out-of-scope :/

project tips & tricks

(mostly things from OH)

if you're just starting...

First, it's okay! **Don't panic!**

Some overall tips:

1. start with getting a simple print function to work
2. figure out how to test locally
3. create easy/trivial versions of **tokenization**, **main interpreter loop**
4. then, slowly add features from the spec
   a. suggestion: make assign work, then expressions, then main function, ...
5. **partial submission is better than no submission at all!**
   a. related: **inefficient but works >>> overthinking**

if you're not yet at 10/10 on provided test cases...

First, it's okay! **Don't panic!**

Some overall tips:

1. each test case contains only a handful of concepts: **do you know which ones?**
2. for test cases that don't work:
    a. Can you isolate the line(s) that's causing the problem?
    b. Can you create a different test case with the same problem?
3. still stuck?
    a. **Ask for help!**
    b. Bandaid fix and move on!

if you are at 10/10...

First, nice job :) that's the hard part!

Now, you need to:

- make sure you implement the rest of the spec
  - **please read the spec!! in full!!**
- figure out edge cases

coming up with edge cases...

Here are some good (generalizable) strategies! **Not all always apply now.**

- many operations/functions have “special values”
  - `0, "", [], {}, ()`, **empty return**, **no return**, ...
- many “special values” still need to follow some general behavior
  - **main function**, NaN, void/undefined/null/nullptr, ...
- concepts that are typically prone to errors in PL
  - **recursion, control flow,** stack management, heap alloc, scoping, ...
- things that behave differently than what you’re used to
  - **global variables, no type coercion,** ...

other things I have discussed in OH

Things that are overkill (that you **do not need to do**):

- stack frames
- ASTs
- “retokenization”
  - turning `['"Hello', 'World"']` “back” into a string
- manually keeping track of recursion count
- using `eval()`
  - you should … not do this
- implementing things not required in the spec!!

other things I have discussed in OH

Random pieces of advice:

- the indentation rule is there to **help you**
  - in particular – good for matching if/endif, func/endfunc, …
  - we won't give you "broken" indentation, and we give you a `validate_program` func
- **stacks** (the data structure) **are your friend**
  - hint: they come up *multiple times* in this project!
- you need to do ***some work*** before you interpret line-by-line
- **avoid overthinking & overcomplicating!**
  - in particular: you are writing an *interpreter*, ***not a compiler!***

~ post-discussion survey ~

always appreciate the feedback!

- how is pacing?
- how was JS gone wrong?
- am I helping you do the project?

see you next week :3

https://forms.gle/33gPkKDfajrrQrZ88

project q&a // help session

(feel free to leave!)

appendix

Useful Project Links:

- Spec
- Gradescope
- Template + Autograder (optional)
- Tips (from Ashwin!)

Fun reading on Python & Tail Recursion

(by Guido van Rossum, the *creator of Python!*)

- **Tail Recursion Elimination**
- **Final Words on Tail Calls**