Course Introduction

Concepts of Programming Languages
Lecture 1

CAS CS 320

Outline

» Discuss the logistics of the course

» Give an overview of what PL is about

» Take a first look at OCaml

Course Logistics

Minutiae

Instructor: Nathan Mull

Teaching Fellows: Zachery Casey and Jared Pincus

Teaching Assistant: Vishesh Jain

Course Webpage: https://nmmull.github.io/CS320/landing/Spring-2025/index.html

Midterm Date: February 27

Grade Breakdown

30%	Assignments	(6 total, 1 dropped, 6% each)
30%	Mini-Projects	(3 total, no drops, 10% each)
20%	Midterm Exam	(February 27 during class)
20%	Final Exam	(Date TBD, Cumulative)

Assignments

Assignments

» Assignments include both written and programming components

Assignments

» Assignments include both written and programming components

» Assignments are due weekly by 8:00PM on Thursdays, one week after the release date. No late submissions

Assignments

- Assignments include both written and programming components
- Assignments are due weekly by 8:00PM on Thursdays, one week after the release date. No late submissions
- Assignments are submitted via **Gradescope.** Regrades are open for a week only

Assignments

» Assignments include both written and programming components

» Assignments are due weekly by 8:00PM on Thursdays, one week after the release date. No late submissions

» Assignments are submitted via **Gradescope**. Regrades are open for a week only

» Solutions must be your own. No group submissions and no AI assistants

Assignments

- Assignments include both written and programming components
- Assignments are due weekly by 8:00PM on Thursdays, one week after the release date. No late submissions
- Assignments are submitted via **Gradescope**. Regrades are open for a week only
- Solutions must be your own. No group submissions and no AI assistants
- Sources must be cited. Any violations will be considered academic misconduct

Assignments

» Assignments include both written and programming components

» Assignments are due weekly by 8:00PM on Thursdays, one week after the release date. No late submissions

» Assignments are submitted via **Gradescope**. Regrades are open for a week only

» Solutions must be your own. No group submissions and no AI assistants

» Sources must be cited. Any violations will be considered academic misconduct

» We will automatically drop your lowest assignment score

Lectures

Lectures

» We will not take attendance in lectures, but it is highly recommended that you attend

Lectures

- We will not take attendance in lectures, but it is highly recommended that you attend
- If something is said in lecture and not on Piazza there is no excuse for missing the information

Lectures

» We will not take attendance in lectures, but it is highly recommended that you attend

» If something is said in lecture and not on Piazza there is no excuse for missing the information

» Barring technical issues, lectures will be **recorded**

Labs

Labs

» Labs meet weekly on Wednesdays. See the registrar for meeting times and locations

Labs

» Labs meet weekly on Wednesdays. See the registrar for meeting times and locations

» Labs are an opportunity to practice the material

Labs

» Labs meet weekly on Wednesdays. See the registrar for meeting times and locations

» Labs are an opportunity to practice the material

» We will not take attendance, but it is highly recommended that you attend

Labs

» Labs meet weekly on Wednesdays. See the registrar for meeting times and locations

» Labs are an opportunity to practice the material

» We will not take attendance, but it is highly recommended that you attend

» All lab material is fair game for exams

Mini-Projects

Mini-Projects

» In the second half of the course, you will complete 3 mini-projects

Mini-Projects

» In the second half of the course, you will complete 3 mini-projects

» Each mini-project is an interpreter for a fragment of OCaml, each more complicated than the last

Mini-Projects

» In the second half of the course, you will complete 3 mini-projects

» Each mini-project is an interpreter for a fragment of OCaml, each more complicated than the last

» You cannot drop a mini-project

Grading

Grading

» Programming assignments and mini-projects will be graded with autograders

Grading

» Programming assignments and mini-projects will be graded with autograders

- **The score you see is the score you get**

Grading

» Programming assignments and mini-projects will be graded with autograders

- **The score you see is the score you get**
- If we can't build your code, you'll receive a zero

Grading

» Programming assignments and mini-projects will be graded with autograders

- **The score you see is the score you get**
- If we can't build your code, you'll receive a zero
- Please submit early enough to be able to see the autograder output

Grading

» Programming assignments and mini-projects will be graded with autograders

- **The score you see is the score you get**
- If we can't build your code, you'll receive a zero
- Please submit early enough to be able to see the autograder output

» Written assignments are graded by hand

Grading

» Programming assignments and mini-projects will be graded with autograders

- **The score you see is the score you get**
- If we can't build your code, you'll receive a zero
- Please submit early enough to be able to see the autograder output

» Written assignments are graded by hand

- They must be exceptionally neat

Grading

» Programming assignments and mini-projects will be graded with autograders

- **The score you see is the score you get**
- If we can't build your code, you'll receive a zero
- Please submit early enough to be able to see the autograder output

» Written assignments are graded by hand

- They must be exceptionally neat
- You must chose the correct pages on Gradescope

Grading

» Programming assignments and mini-projects will be graded with autograders

- **The score you see is the score you get**
- If we can't build your code, you'll receive a zero
- Please submit early enough to be able to see the autograder output

» Written assignments are graded by hand

- They must be exceptionally neat
- You must chose the correct pages on Gradescope
- We reserve the right to dock points for not following these instructions

Course Communication

Course Communication

» We will be using **Piazza** for course communication

Course Communication

» We will be using **Piazza** for course communication

» Please check regularly. "I didn't see the piazza post" is not a valid excuse

Course Communication

» We will be using **Piazza** for course communication

» Please check regularly. "I didn't see the piazza post" is not a valid excuse

» We will not respond to any material-related questions by email

Course Communication

- We will be using **Piazza** for course communication
- Please check regularly. "I didn't see the piazza post" is not a valid excuse
- We will not respond to any material-related questions by email
- If you have logistical questions (e.g., about disability accommodations) send me an email directly

Course Webpage

https://nmmull.github.io/CS320/landing/Spring-2025/index.html

The webpage contains readings, assignments and labs.

Please check it frequently for updates

Course Repository

https://github.com/BU-CS320/cs320-spring-2025

The course repo contains starter code and lecture material

In **Lab 1**, you'll set up a mirror of this repository for assignment submission

Course Standard Library

https://nmmull.github.io/CS320/landing/Spring-2025/Specifications/Stdlib320/index.html

We'll assume a barebones standard library during the first half of the course

You'll need to familiarize yourself with what's there during this first half

You'll install it during Lab 1 (Tomorrow)

Questions?

If I missed anything, ask on Piazza

Make sure you're on Piazza and Gradescope, checking the course webpage, and pulling down the course repo frequently

By continuing in this course you're agreeing to all these conditions

One Last Thing

Remember to be kind. This is a difficult course. Don't take it out on other students

We care about your success in this course. We're not out to get you, we're here to help

Use your best judgment, you're adults

What is a PL?

Fair Question

How would you define a PL?

How would you explain it to your roommate?

How would you answer if you were asked during an interview?

Discuss this with the people around you for 1min

Programmer's view of a PL

```
42  def subtraction():
43      num_1 = randint(0,9)
44      num_2 = randint(0,9)
45
46      print(f"What is {num_1} - {num_2} ?\n")
47
48      choice = input("> ")
49
50      if int(choice) == num_1 - num_2:
51          print("Correct! Nice job! Keep on playing!\n")
52          start_game()
53      else:
54          print(f"Incorrect...the answer is {num_1-num_2}!\n")
55          start_game()
56
57  def multiplication():
58      num_1 = randint(0,9)
59      num_2 = randint(0,9)
60
61      print(f"What is {num_1} x {num_2} ?\n")
62
63      choice = input("> ")
64
65      if int(choice) == num_1*num_2:
66          print("Correct! Nice job! Keep on playing!\n")
67          start_game()
68      else:
69          print(f"Incorrect...the answer is {num_1*num_2}!\n")
70          start_game()
```

Programmer's view of a PL

» A tool for programming

```
42  def subtraction():
43      num_1 = randint(0,9)
44      num_2 = randint(0,9)
45
46      print(f"What is {num_1} - {num_2} ?\n")
47
48      choice = input("> ")
49
50      if int(choice) == num_1 - num_2:
51          print("Correct! Nice job! Keep on playing!\n")
52          start_game()
53      else:
54          print(f"Incorrect...the answer is {num_1-num_2}!\n")
55          start_game()
56
57  def multiplication():
58      num_1 = randint(0,9)
59      num_2 = randint(0,9)
60
61      print(f"What is {num_1} x {num_2} ?\n")
62
63      choice = input("> ")
64
65      if int(choice) == num_1*num_2:
66          print("Correct! Nice job! Keep on playing!\n")
67          start_game()
68      else:
69          print(f"Incorrect...the answer is {num_1*num_2}!\n")
70          start_game()
```

Programmer's view of a PL

» A tool for programming

» A text-based way of interacting with hardware/a computer

```
42  def subtraction():
43      num_1 = randint(0,9)
44      num_2 = randint(0,9)
45
46      print(f"What is {num_1} - {num_2} ?\n")
47
48      choice = input("> ")
49
50      if int(choice) == num_1 - num_2:
51          print("Correct! Nice job! Keep on playing!\n")
52          start_game()
53      else:
54          print(f"Incorrect...the answer is {num_1-num_2}!\n")
55          start_game()
56
57  def multiplication():
58      num_1 = randint(0,9)
59      num_2 = randint(0,9)
60
61      print(f"What is {num_1} x {num_2} ?\n")
62
63      choice = input("> ")
64
65      if int(choice) == num_1*num_2:
66          print("Correct! Nice job! Keep on playing!\n")
67          start_game()
68      else:
69          print(f"Incorrect...the answer is {num_1*num_2}!\n")
70          start_game()
```

Programmer's view of a PL

» A tool for programming

» A text-based way of interacting with hardware/a computer

» A way of organizing and working with data

```
42  def subtraction():
43      num_1 = randint(0,9)
44      num_2 = randint(0,9)
45
46      print(f"What is {num_1} - {num_2} ?\n")
47
48      choice = input("> ")
49
50      if int(choice) == num_1 - num_2:
51          print("Correct! Nice job! Keep on playing!\n")
52          start_game()
53      else:
54          print(f"Incorrect...the answer is {num_1-num_2}!\n")
55          start_game()
56
57  def multiplication():
58      num_1 = randint(0,9)
59      num_2 = randint(0,9)
60
61      print(f"What is {num_1} x {num_2} ?\n")
62
63      choice = input("> ")
64
65      if int(choice) == num_1*num_2:
66          print("Correct! Nice job! Keep on playing!\n")
67          start_game()
68      else:
69          print(f"Incorrect...the answer is {num_1*num_2}!\n")
70          start_game()
```

Programmer's view of a PL

» A tool for programming

» A text-based way of interacting with hardware/a computer

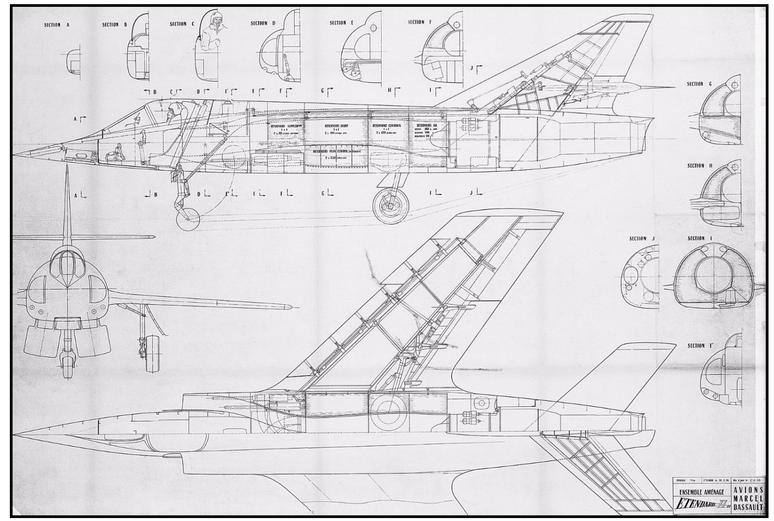
» A way of organizing and working with data

These are not what the course is about

```
42  def subtraction():
43      num_1 = randint(0,9)
44      num_2 = randint(0,9)
45
46      print(f"What is {num_1} - {num_2} ?\n")
47
48      choice = input("> ")
49
50      if int(choice) == num_1 - num_2:
51          print("Correct! Nice job! Keep on playing!\n")
52          start_game()
53      else:
54          print(f"Incorrect...the answer is {num_1-num_2}!\n")
55          start_game()
56
57  def multiplication():
58      num_1 = randint(0,9)
59      num_2 = randint(0,9)
60
61      print(f"What is {num_1} x {num_2} ?\n")
62
63      choice = input("> ")
64
65      if int(choice) == num_1*num_2:
66          print("Correct! Nice job! Keep on playing!\n")
67          start_game()
68      else:
69          print(f"Incorrect...the answer is {num_1*num_2}!\n")
70          start_game()
```

Aside: Users vs. Designers

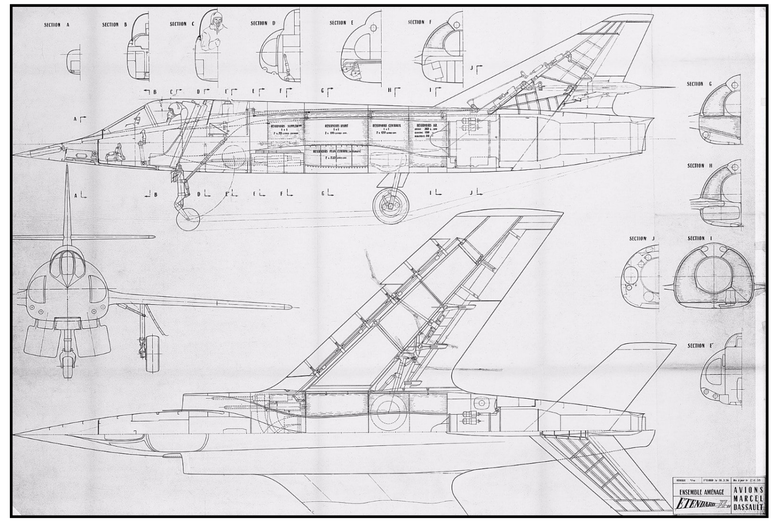
VS



Aside: Users vs. Designers

Programmers *use* PLs. We're interested in **designing** PLs

vs

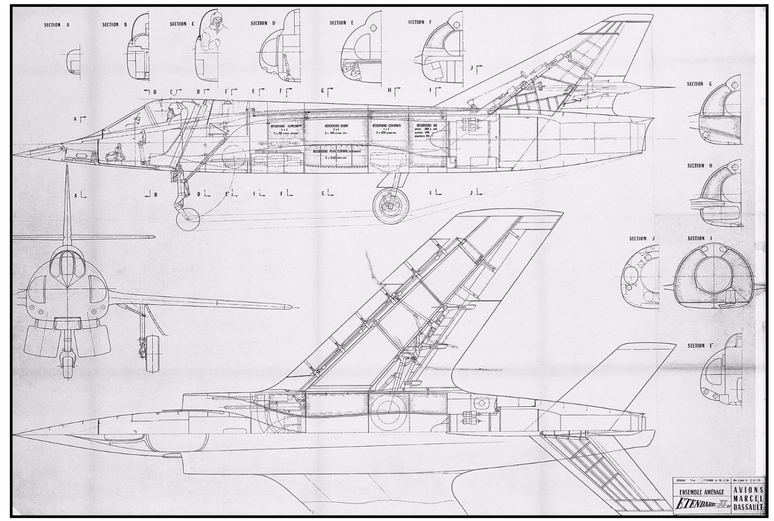


Aside: Users vs. Designers

Programmers *use* PLs. We're interested in **designing** PLs

Users are not necessarily the best designers...Who should design PLs?

vs



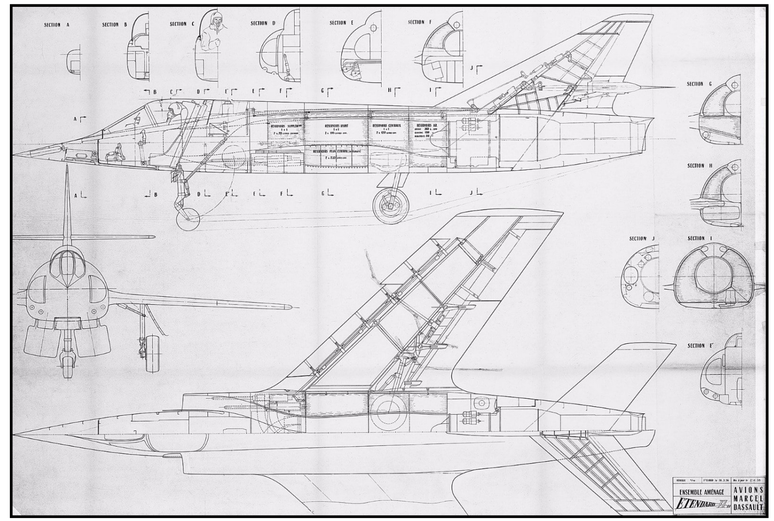
Aside: Users vs. Designers

Programmers *use* PLs. We're interested in **designing** PLs

Users are not necessarily the best designers...Who should design PLs?

Answer: Mathematicians

vs



Aside: Users vs. Designers

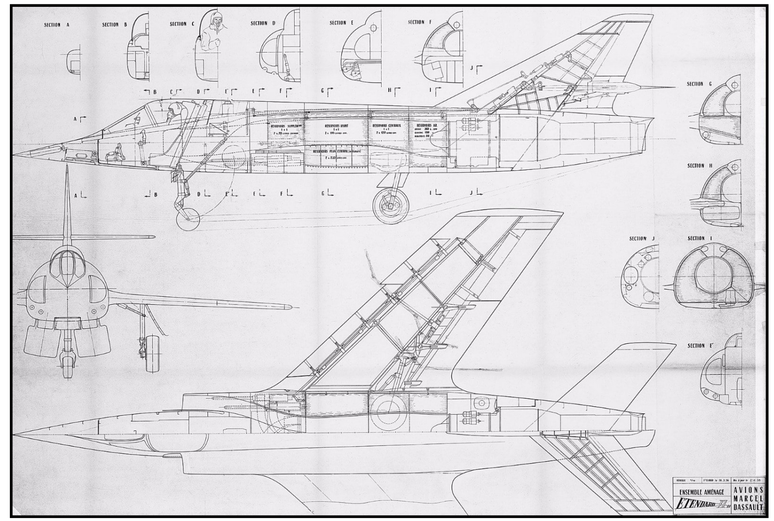
Programmers *use* PLs. We're interested in **designing** PLs

Users are not necessarily the best designers...Who should design PLs?

Answer: Mathematicians

(CS320 is secretly a math class)

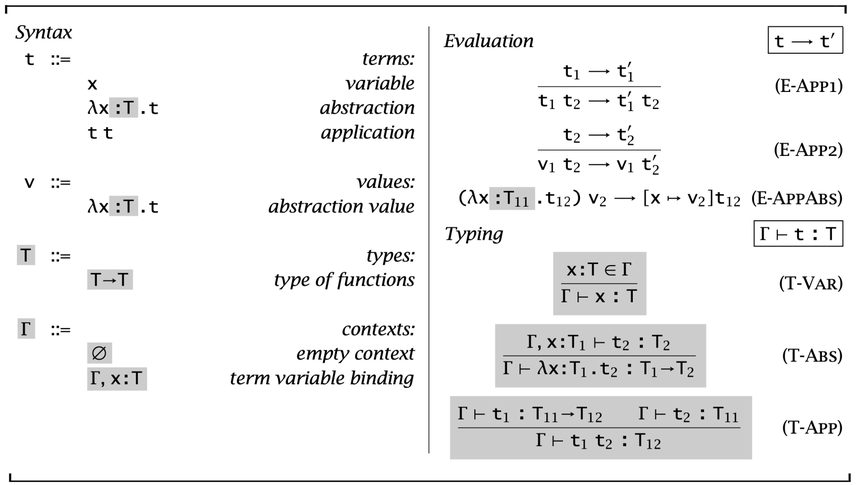
vs



Mathematician's View of PL

```python
def subtraction():
    num_1 = randint(0,9)
    num_2 = randint(0,9)

    print(f"What is {num_1} - {num_2} ?\n")

    choice = input("> ")

    if int(choice) == num_1 - num_2:
        print("Correct! Nice job! Keep on playing!\n")
        start_game()
    else:
        print(f"Incorrect...the answer is {num_1-num_2}!\n")
        start_game()

def multiplication():
    num_1 = randint(0,9)
    num_2 = randint(0,9)

    print(f"What is {num_1} x {num_2} ?\n")

    choice = input("> ")

    if int(choice) == num_1*num_2:
        print("Correct! Nice job! Keep on playing!\n")
        start_game()
    else:
        print(f"Incorrect...the answer is {num_1*num_2}!\n")
        start_game()
```

VS



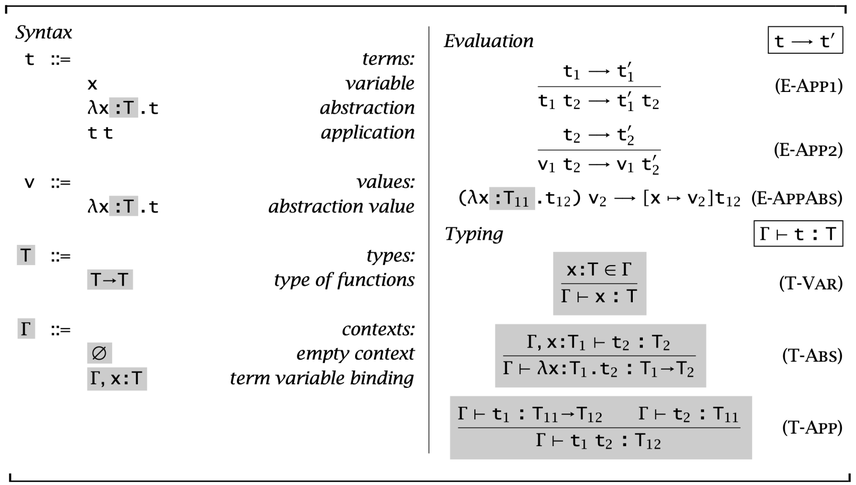
(from T&PL by Pierce)

Mathematician's View of PL

» a mathematical object, like a polynomial or a vector

```python
def subtraction():
    num_1 = randint(0,9)
    num_2 = randint(0,9)

    print(f"What is {num_1} - {num_2} ?\n")

    choice = input("> ")

    if int(choice) == num_1 - num_2:
        print("Correct! Nice job! Keep on playing!\n")
        start_game()
    else:
        print(f"Incorrect...the answer is {num_1-num_2}!\n")
        start_game()

def multiplication():
    num_1 = randint(0,9)
    num_2 = randint(0,9)

    print(f"What is {num_1} x {num_2} ?\n")

    choice = input("> ")

    if int(choice) == num_1*num_2:
        print("Correct! Nice job! Keep on playing!\n")
        start_game()
    else:
        print(f"Incorrect...the answer is {num_1*num_2}!\n")
        start_game()
```

VS



(from T&PL by Pierce)

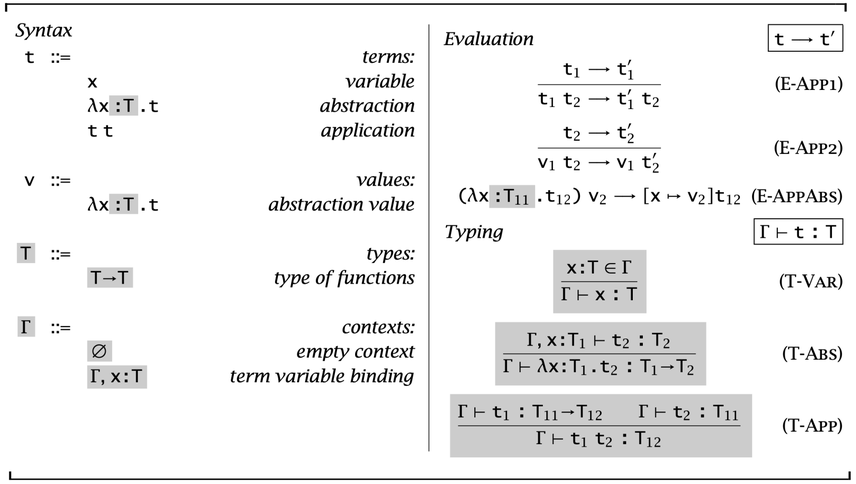
Mathematician's View of PL

» a mathematical object, like a polynomial or a vector

» a formal specification

```python
def subtraction():
    num_1 = randint(0,9)
    num_2 = randint(0,9)

    print(f"What is {num_1} - {num_2} ?\n")

    choice = input("> ")

    if int(choice) == num_1 - num_2:
        print("Correct! Nice job! Keep on playing!\n")
        start_game()
    else:
        print(f"Incorrect...the answer is {num_1-num_2}!\n")
        start_game()

def multiplication():
    num_1 = randint(0,9)
    num_2 = randint(0,9)

    print(f"What is {num_1} x {num_2} ?\n")

    choice = input("> ")

    if int(choice) == num_1*num_2:
        print("Correct! Nice job! Keep on playing!\n")
        start_game()
    else:
        print(f"Incorrect...the answer is {num_1*num_2}!\n")
        start_game()
```

VS



(from T&PL by Pierce)

Mathematician's View of PL

» a mathematical object, like a polynomial or a vector

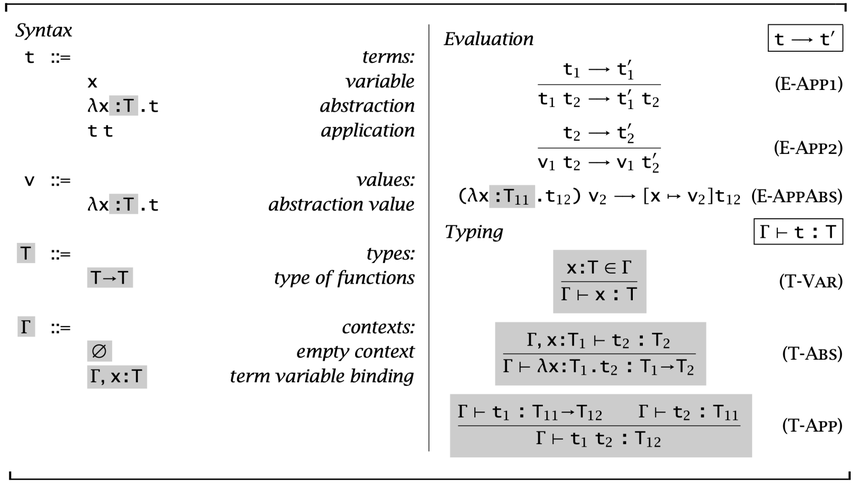
» a formal specification

» composed of exactly three things:

- Syntax
- Type System
- Semantics

```python
def subtraction():
    num_1 = randint(0,9)
    num_2 = randint(0,9)

    print(f"What is {num_1} - {num_2} ?\n")

    choice = input("> ")

    if int(choice) == num_1 - num_2:
        print("Correct! Nice job! Keep on playing!\n")
        start_game()
    else:
        print(f"Incorrect...the answer is {num_1-num_2}!\n")
        start_game()

def multiplication():
    num_1 = randint(0,9)
    num_2 = randint(0,9)

    print(f"What is {num_1} x {num_2} ?\n")

    choice = input("> ")

    if int(choice) == num_1*num_2:
        print("Correct! Nice job! Keep on playing!\n")
        start_game()
    else:
        print(f"Incorrect...the answer is {num_1*num_2}!\n")
        start_game()
```

VS



(from T&PL by Pierce)

Why does this matter?

Why does this matter?

There are a lot of *bad* PLs out there

Why does this matter?

There are a lot of *bad* PLs out there

We want ***good*** PLs

Why does this matter?

There are a lot of *bad* PLs out there

We want ***good*** PLs

This course is about finding out **what makes a PL good**

Why does this matter?

There are a lot of *bad* PLs out there

We want ***good*** PLs

This course is about finding out **what makes a PL good**

(correction: what I think makes a PL good)

A Couple Notes to the Skeptical

A Couple Notes to the Skeptical

» Knowing how your car works is still valuable, even if you're not a mechanic...

A Couple Notes to the Skeptical

» Knowing how your car works is still valuable, even if you're not a mechanic...

» There are jobs you can only get by knowing a functional PL, and there are more jobs in PL right now than you might expect (DSLs, compilers, verification, security)

A Couple Notes to the Skeptical

» Knowing how your car works is still valuable, even if you're not a mechanic...

» There are jobs you can only get by knowing a functional PL, and there are more jobs in PL right now than you might expect (DSLs, compilers, verification, security)

» If you think PL is useless, at least learn the language to tell people why you think so...

Formal PL

The Three Components

The Three Components

Syntax: What a *well-formed* program in your PL?

```
def f():
    return 3
```

```
define f():
    3 return
```

The Three Components

Syntax: What a *well-formed* program in your PL?

```
def f():
    return 3
```

```
define f():
    3 return
```

Type System (Static Semantics): What is a *valid* program in your PL?

```
x = 2 + 2
```

```
x = 2 + "two"
```

The Three Components

Syntax: What a *well-formed* program in your PL?

```
def f():
    return 3
```

```
define f():
    3 return
```

Type System (Static Semantics): What is a *valid* program in your PL?

```
x = 2 + 2
```

```
x = 2 + "two"
```

Semantics (Dynamic Semantics): What is the *output* of a (valid) program?

```
>>> 2 + 2
4
```

```
>>> 2 + 2
False
```

For everything we do from now on, we'll define the **syntax rules**, the **typing rules**, and the **semantic rules**

Syntax Rules

Syntax Rules

Syntax rules describe what are well-formed expression or programs in a PL. They are independent of meaning

Syntax Rules

Syntax rules describe what are well-formed expression or programs in a PL. They are independent of meaning

A syntax rule will almost always be of the form:

Syntax Rules

Syntax rules describe what are well-formed expression or programs in a PL. They are independent of meaning

A syntax rule will almost always be of the form:

If ***<such-and-such>*** *is a well-formed expression and* ***<some-other-things>*** *are a well-formed expression, then* ***<some-combination-of-such-and-such-and-some-other-things>*** *is a well-formed expression*

Example: Integer Addition Syntax

```
<expr> ::= <expr> + <expr>
```

If e_1 is a well-formed expression and e_2 is a well-formed expression, then e_1 + e_2 is a well-formed expression

Typing Rules

Typing Rules

Typing rules describe the types of programs or expressions in a PL

Typing Rules

Typing rules describe the types of programs or expressions in a PL

They will almost always be of the form:

Typing Rules

Typing rules describe the types of programs or expressions in a PL

They will almost always be of the form:

If **<such-and-such>** *is of* **<such-and-such-type>** *and* **<some-other-things>** *are of* **<some-other-types>**, *then* **<some-combination-of-such-and-such-and-some-other-things>** *is of* **<some-different-type>**

Example: Integer Addition Typing

$$\frac{\Gamma \vdash e_1 : \texttt{int} \qquad \Gamma \vdash e_2 : \texttt{int}}{\Gamma \vdash e_1 \texttt{ + } e_2 : \texttt{int}} \text{ (addInt)}$$

If e_1 is an `int` (in any context Γ) and e_2 is an `int` then (in any context Γ) e_1 `+` e_2 is an `int` (in any context Γ)

What is a type?

What is a type?

And what the heck is a *context*?

What is a type?

And what the heck is a *context*?

Good questions, we'll ignore them for now and use our intuitions...(we'll get to them very soon)

What is a type?

And what the heck is a *context*?

Good questions, we'll ignore them for now and use our intuitions...(we'll get to them very soon)

Think of a context in the English sense like the "computational setting" or the "environment"

Semantic Rules

Semantic Rules

Semantic rules describe the output of a program or *value* of an expression

Semantic Rules

Semantic rules describe the output of a program or *value* of an expression

They will almost always be of the form:

Semantic Rules

Semantic rules describe the output of a program or *value* of an expression

They will almost always be of the form:

If **<such-and-such>** evaluates to **<such-and-such-value>** and **<some-other-things>** evaluate to **<some-other-values>** then **<some-combination-of-such-and-such-and-some-other-things>** evaluates to **<some-other-value-computed-based-on-such-and-such-value-and-some-other-values>**

Example: Integer Addition Semantics

$$\frac{e_1 \Downarrow v_1 \qquad e_2 \Downarrow v_2}{e_1 + e_2 \Downarrow v_1 + v_2}\ \text{(evalInt)}$$

If e_1 evaluates to the (integer) v_1 and e_2 evaluates to the (integer) v_2, then $e_1 + e_2$ evaluates to the (integer) $v_1 + v_2$

We'll come back to all this soon...

OCaml: First Look

Preamble

Preamble

A lot of people have a lot to say about OCaml...it's fun but also a bit difficult, it's a very different game then we're probably used to:

Preamble

A lot of people have a lot to say about OCaml...it's fun but also a bit difficult, it's a very different game then we're probably used to:

» **Minimality:** The language is simple, there's very little to it

Preamble

A lot of people have a lot to say about OCaml...it's fun but also a bit difficult, it's a very different game then we're probably used to:

» **Minimality:** The language is simple, there's very little to it

» **Functional:** A completely different paradigm. We're not writing procedures via commands/statements, we're defining values via expressions

Preamble (Continued)

Preamble (Continued)

That said, this is not an introductory programming course

Preamble (Continued)

That said, this is not an introductory programming course

We're going to leave a lot of the learning syntax to you. We won't dwell on how comments work, or what a floating-point value is, these are things you have to pick up along the way

Preamble (Continued)

That said, this is not an introductory programming course

We're going to leave a lot of the learning syntax to you. We won't dwell on how comments work, or what a floating-point value is, these are things you have to pick up along the way

Okay, let's get started...

Overview

Overview

» OCaml is a statically-typed "industrial-strength functional programming language" with powerful type-inference

Overview

» OCaml is a statically-typed "industrial-strength functional programming language" with powerful type-inference

» It was developed at Inria (smart researchers in France) in the 90s

Overview

» OCaml is a statically-typed "industrial-strength functional programming language" with powerful type-inference

» It was developed at Inria (smart researchers in France) in the 90s

» It won the ACM SIGPLAN Programming Languages Software Award in 2023

Overview

- OCaml is a statically-typed "industrial-strength functional programming language" with powerful type-inference
- It was developed at Inria (smart researchers in France) in the 90s
- It won the ACM SIGPLAN Programming Languages Software Award in 2023
- It's used/developed heavily by Jane Street (and too a lesser degree by facebook, Microsoft, docker, Wolfram)

Functional vs. Imperative

OCaml is a functional language. This means a couple things:

- No state (which means no loops!)
- We don't think of a program as describing a procedure, but as defining a value

Anatomy of an OCaml Program

```
let x = 3

let y = "string"

(* function definition *)
let square x = x * x

(* recursive function definition *)
let rec f x = if x = 0 then 0 else x + f (x - 1)

(* We can't just print , we assign to wildcard *)
let _ = print_endline("Hello world")
```

Anatomy of an OCaml Program

```
let x = 3

let y = "string"

(* function definition *)
let square x = x * x

(* recursive function definition *)
let rec f x = if x = 0 then 0 else x + f (x - 1)

(* We can't just print , we assign to wildcard *)
let _ = print_endline("Hello world")
```

An OCaml Program consists of top-level let-expressions

Anatomy of an OCaml Program

```
let x = 3

let y = "string"

(* function definition *)
let square x = x * x

(* recursive function definition *)
let rec f x = if x = 0 then 0 else x + f (x - 1)

(* We can't just print , we assign to wildcard *)
let _ = print_endline("Hello world")
```

An OCaml Program consists of top-level let-expressions

These lines assign values to names we'll talk about variables a lot more later, but remember, no state, these are not "global variables")

Anatomy of an OCaml Program

```
let x = 3

let y = "string"

(* function definition *)
let square x = x * x

(* recursive function definition *)
let rec f x = if x = 0 then 0 else x + f (x - 1)

(* We can't just print , we assign to wildcard *)
let _ = print_endline("Hello world")
```

An OCaml Program consists of top-level let-expressions

These lines assign values to names we'll talk about variables a lot more later, but remember, no state, these are not "global variables")

The values we assign are gotten by *evaluating* the expressions on the RHS of the '=' sign

Expressions

Expressions are syntactic objects which describe values in a program

Mneumonic: ***E****xpressions are* ***EV****aluated to* ***V****alues*

They appear in both functional and imperative PLs, but in functional PLs we *only* have expressions

$$2 + (2 * 3)$$

```
if x = 3 then 3 else 4
```

$$H(f(f(f(x, y), 2), g(z)))$$

A Note on State

```python
def fact(n):
    acc = 1
    for i in range(1, n + 1): # i is "stateful"
        acc *= i
    return acc
```

In Python, we can define variables that change throughout the evaluation of the program

We can't do this in OCaml. Instead we use **recursion**(!)

If you can write recursive functions in Python, then you can program in OCaml

Free yourself from Pythonic thinking...

demo

(learning by doing)

The Point

Imperative programs define how to update state by a sequence of commands

Function programs define what the output is for a given input

Every imperative program can be made functional by "simulating" loops using recursion

One Last Point: Building Interpreters

One Last Point: Building Interpreters

Okay, so *PL is math*, but also, we still like to *use* PLs. The three components of a PL correspond to the three things we need to *implement* in an **interpreter** of a PL.

One Last Point: Building Interpreters

Okay, so *PL is math*, but also, we still like to *use* PLs. The three components of a PL correspond to the three things we need to *implement* in an **interpreter** of a PL.

» **Syntax** is implemented by a **parser**

```
parse : string -> expr
```

One Last Point: Building Interpreters

Okay, so *PL is math*, but also, we still like to *use* PLs. The three components of a PL correspond to the three things we need to *implement* in an **interpreter** of a PL.

» **Syntax** is implemented by a **parser**

```
parse : string -> expr
```

» **Type system** is implemented by a **type checker**

```
type_check : expr -> bool (* valid or not *)
```

One Last Point: Building Interpreters

Okay, so *PL is math*, but also, we still like to *use* PLs. The three components of a PL correspond to the three things we need to *implement* in an **interpreter** of a PL.

» **Syntax** is implemented by a **parser**

```
parse : string -> expr
```

» **Type system** is implemented by a **type checker**

```
type_check : expr -> bool (* valid or not *)
```

» **(Dynamic) semantics** is implemented by an **evaluator**

```
eval : expr -> value
```

Next Steps

- Make sure you're on Piazza and Gradescope, keep an eye on announcements
- Bookmark the course webpage and course repo
- Install opam, VSCode, the course standard library, etc. *(go to lab tomorrow)*
- **Do the reading listed on the course webpage**

Summary

A PL is a mathematical object given by its **syntax**, **type system** and **semantics**

There is **no state** in functional programming. Programs define the output for a given input

Practice, practice, practice. Functional programming takes time to learn, but once you get it, it's as easy as programming in any other PL