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

Course Staff

Instructors: Nathan Mull & Ankush Das (me)

Teaching Fellows: Sam Silverman & Ugur Yavuz

Course Assistants: Miranda Quimbar & Shiyun Yang

Office Hours will be posted on website soon

Lectures

Lectures

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

Lectures

- We will not take attendance,
  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,
  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

» Weekly on Wednesdays. See the course webpage for meeting times and locations

Labs

- Weekly on Wednesdays. See the course webpage for meeting times and locations
- Labs are an opportunity to practice the material

Labs

- Weekly on Wednesdays. See the course webpage 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

- Weekly on Wednesdays. See the course webpage 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

Grade Breakdown

20%	Assignments	(6 total, 1 dropped, 4% each)
16%	In-Class Quiz	(Sep 30 & Nov 18, 1 dropped)
24%	Mini-Projects	(3 total, no drops, 8% each)
20%	Midterm Exam	(Oct 16, 2025 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
- Due weekly by 8:00PM on Thursdays, one week after release date.
  No late submissions

Assignments

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

Assignments

- Assignments include both written and programming components
- Due weekly by 8:00PM on Thursdays, one week after release date.
  No late submissions
- Submitted via **Gradescope.** Regrades open for a week
- Solutions must be your own. Sources must be cited.
  No group submissions.
  And please **no AI assistants**!

Assignments

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

Assignments

- Assignments include both written and programming components
- Due weekly by 8:00PM on Thursdays, one week after release date.
  No late submissions
- Submitted via **Gradescope.** Regrades open for a week
- Solutions must be your own. Sources must be cited.
  No group submissions.
  And please **no AI assistants**!
- Any violations will be considered academic misconduct
- We will automatically drop your lowest assignment score

In-Lecture Quizzes

In-Lecture Quizzes

- 2 quizzes: Sep 30 and Nov 18 ($\frac{1}{4}$th and $\frac{3}{4}$th point in the semester)

In-Lecture Quizzes

- 2 quizzes: Sep 30 and Nov 18 ($\frac{1}{4}$th and $\frac{3}{4}$th point in the semester)
- **Goal:** to make sure you are learning the concepts (particularly programming), and not just using AI assistants!

In-Lecture Quizzes

- 2 quizzes: Sep 30 and Nov 18 ($\frac{1}{4}$th and $\frac{3}{4}$th point in the semester)
- **Goal:** to make sure you are learning the concepts (particularly programming), and not just using AI assistants!
- We will automatically drop your lower quiz score

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 are graded with autograders

Grading

» Programming assignments and mini-projects are graded with autograders

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

Grading

» Programming assignments and mini-projects are 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 are 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 are 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 are 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 are 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 choose the correct pages on Gradescope

Grading

- Programming assignments and mini-projects are 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 choose the correct pages on Gradescope
  - We reserve the right to dock points for not following 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
- Ask any material-related questions on Piazza, **not 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
- Ask any material-related questions on Piazza, **not by email**
- If you have logistical questions (e.g., about disability accommodations) send us an email directly

Course Webpage

The webpage contains readings, assignments and labs. **Please check it frequently for updates**

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

Course Repository

The course repo contains starter code and lecture material

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

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

Course Standard Library

We'll assume a very basic standard library for 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 we 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 (or us)

We care about your success in this course, we’re here to help

Use your best judgment, you're adults

What is this Course About?

Theme of the Course

Theme of the Course

- This is not a course on “Programming”! We will not teach you programming. In fact, I am not a good programmer myself.

Theme of the Course

- This is not a course on “Programming”! We will not teach you programming. In fact, I am not a good programmer myself.
- This is a course on “Programming Languages” (PL).

Theme of the Course

- This is not a course on “Programming”! We will not teach you programming. In fact, I am not a good programmer myself.
- This is a course on “Programming Languages” (PL).
- You’ll learn “**Programming Abstractions**”, i.e., high-level constructs to make programming easier (e.g., loops, functions, arithmetic)

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 1 min

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

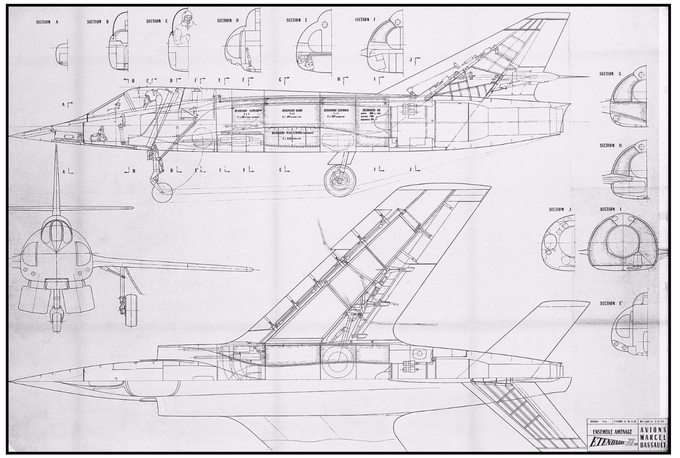
- A tool for programming
- A text-based way of interacting with hardware/a computer
- A way of organizing and working with data

This is 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()
```

Users vs. Designers

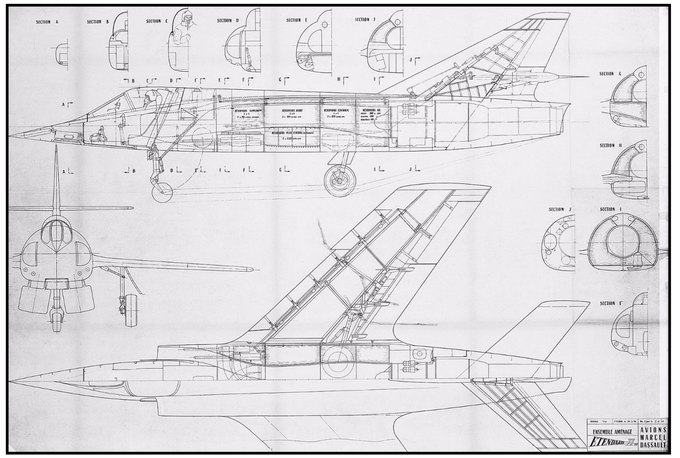
vs.



Users vs. Designers

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

vs.

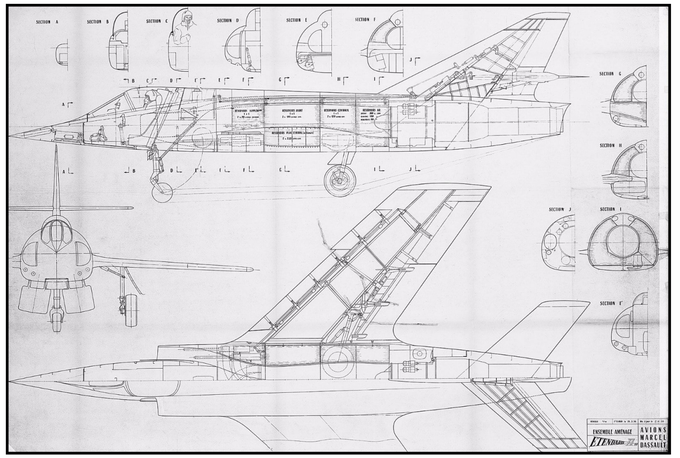


Users vs. Designers

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

Users are not necessarily the best designers; I can drive a car, but not make one ……
Who should design PLs?

vs.



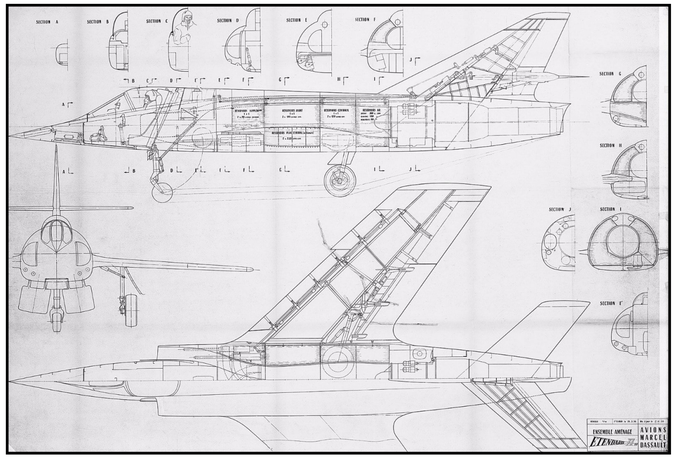
Users vs. Designers

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

Users are not necessarily the best designers; I can drive a car, but not make one ……
Who should design PLs?

Answer: Mathematicians

vs.



Users vs. Designers

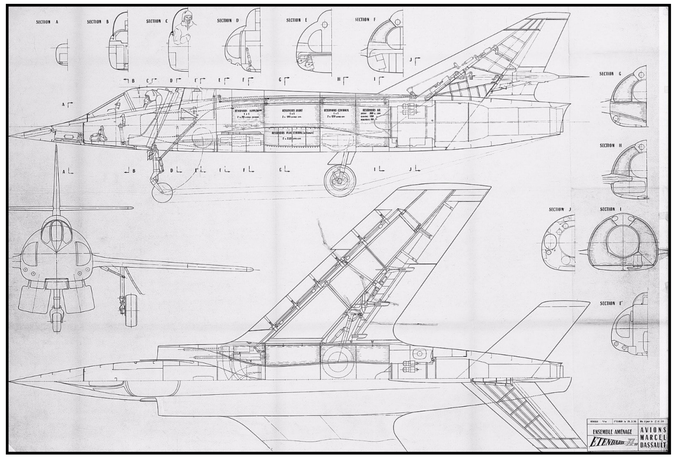
Programmers *use* PLs. We're interested in **designing** PLs

Users are not necessarily the best designers; I can drive a car, but not make one ……
Who should design PLs?

Answer: Mathematicians

(CS320 is secretly a math class, sorry)

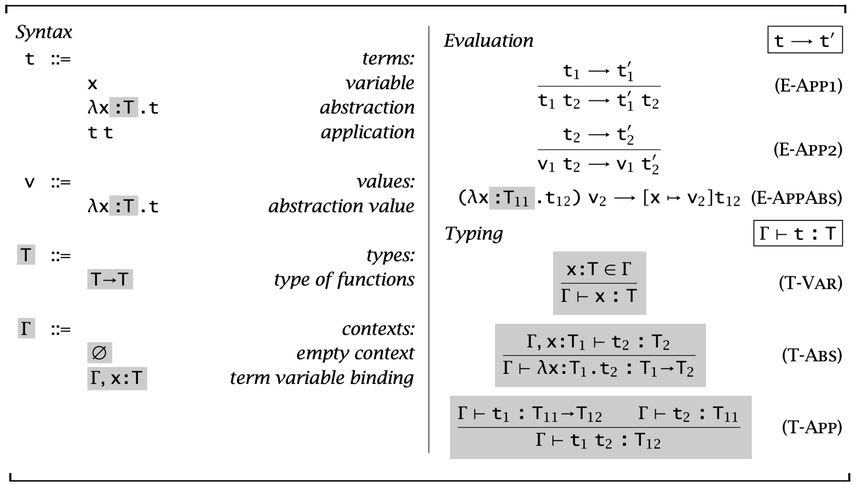
vs.



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.



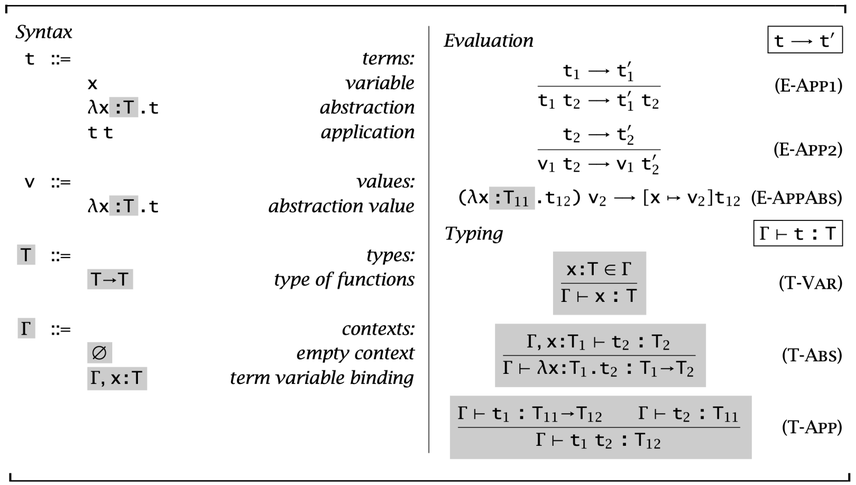
(from T&PL by Pierce)

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)

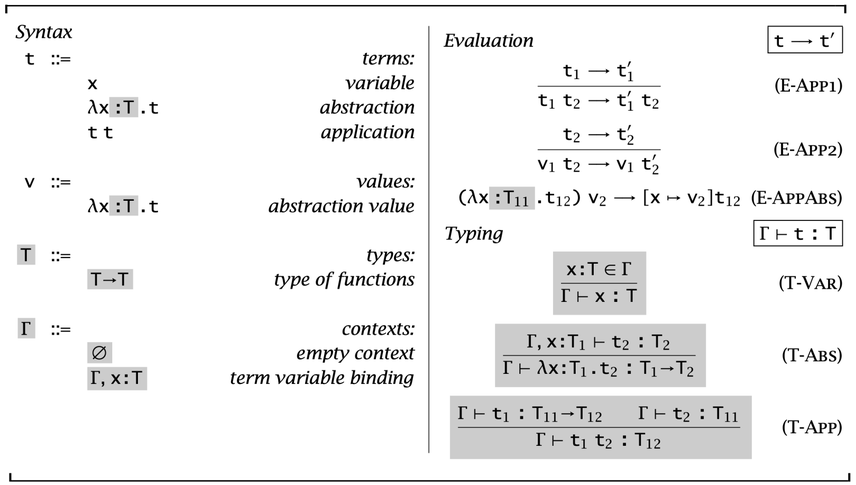
Mathematician's View of PL

» 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

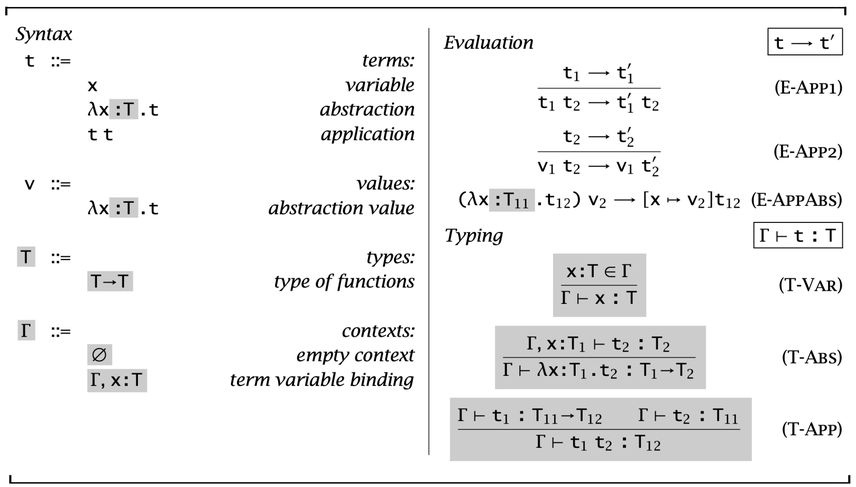
» a formal specification

» 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**

Gist: ***a good PL has a mathematically well-defined syntax, type system, and semantics***

Definition of a PL

The Three Components

The Three Components

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

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

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

The Three Components

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

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

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

Type System: 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
```

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

Type System: What is a *valid* program in your PL?

```
x = 2 + 2
```

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

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 expressions or programs in a PL. They are independent of meaning

Syntax Rules

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

A syntax rule will almost always be of the form:

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

Example: Integer Addition Syntax

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

In formal notation:

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

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

Example: Integer Addition Typing

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*)*

In formal notation:

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

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

Example: Integer Addition Semantics

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)* $v_1 + v_2$

In mathy notation:

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

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

What Language to Use for the Course?

What Language to Use for the Course?

What is OCaml?

What is OCaml?

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

What is OCaml?

- **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

Overview

Overview

- OCaml is a statically-typed functional programming language

Overview

- OCaml is a statically-typed functional programming language
- It is industrial-strength: lots of documentation, tutorials, stack overflow posts

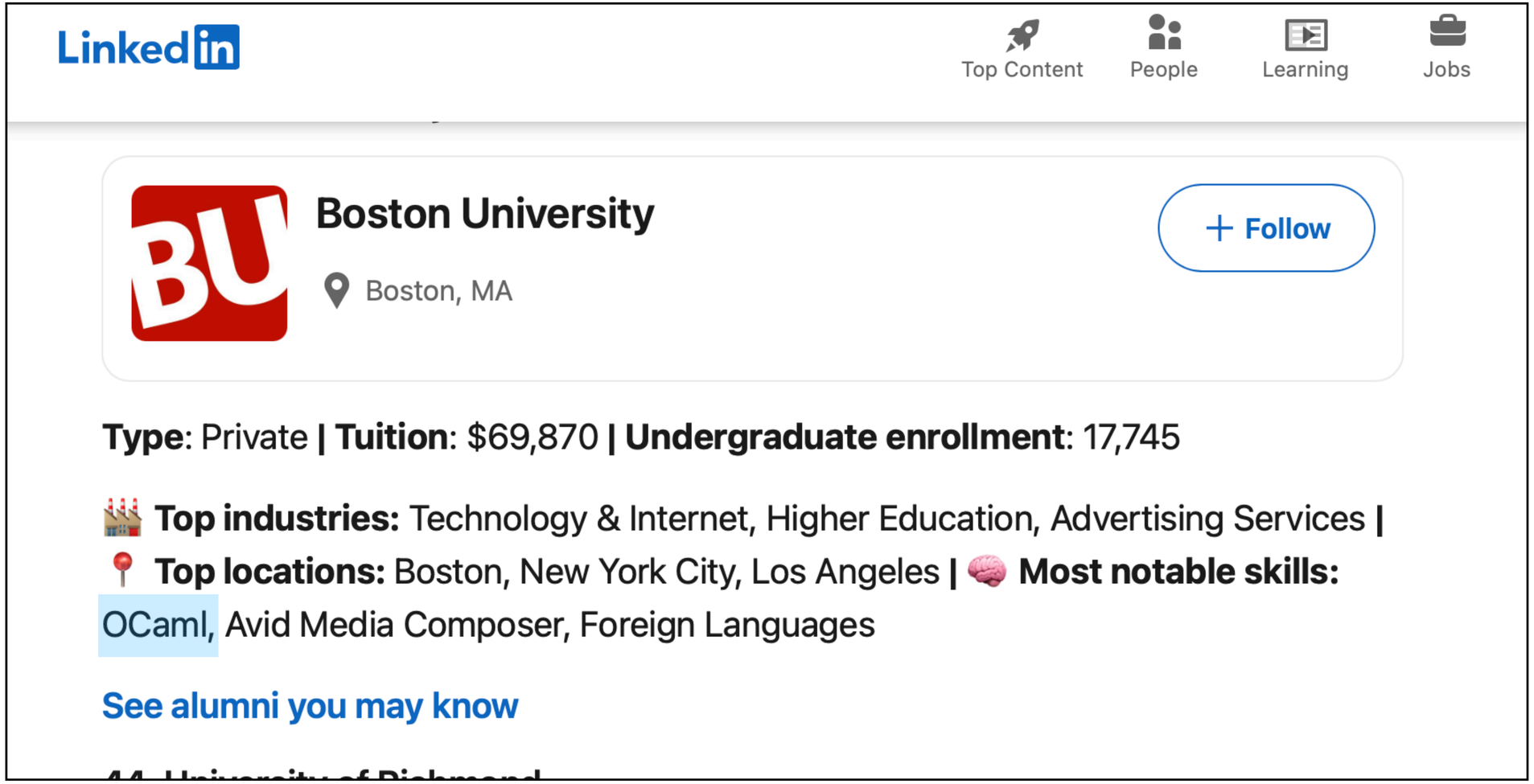
Overview

- OCaml is a statically-typed functional programming language
- It is industrial-strength: lots of documentation, tutorials, stack overflow posts
- It won the ACM SIGPLAN Programming Languages Software Award in 2023

Overview

- OCaml is a statically-typed functional programming language
- It is industrial-strength: lots of documentation, tutorials, stack overflow posts
- It won the ACM SIGPLAN Programming Languages Software Award in 2023
- It's used/developed heavily by Jane Street (and to a lesser degree by Bloomberg, Meta, Docker, Wolfram)

Aside: OCaml is a "notable skill" we provide!



LinkedIn Top Colleges 2025: The 50 best colleges for long-term career success in the U.S.

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 a function taking an input and producing an output

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, then you can program in OCaml

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 *)
```

» **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