

Programming Languages and Paradigms

COMP 302

Instructor: Jacob Errington
School of Computer Science
McGill University

Lesson 1: Welcome to COMP 302

Why so many programming
languages?

So many *ways to write programs!*

So many *ways to write programs!*

And these different ways, categorized, give rise to **paradigms**.

The landscape of programming languages in a nutshell

What is a programming paradigm?

“A style of building the structure and elements of computer programs.” – Wikipedia

The landscape of programming languages in a nutshell

What is a programming paradigm?

“A style of building the structure and elements of computer programs.” – Wikipedia

Two major paradigms

- ▶ *Imperative programming* expresses a computation as a sequence of explicit steps
- ▶ *Declarative programming* expresses the *logic* of a computation without describing its control flow

Background: imperative programming

- ▶ Associated with *stateful* computations.
- ▶ A program is a series of *statements* that affect the program state.

Background: imperative programming

- ▶ Associated with *stateful* computations.
- ▶ A program is a series of *statements* that affect the program state.

Most languages you've used until now belong this paradigm.

Python. *Imperative.* Object-oriented. Functional.

C. Procedural. *Imperative.*

Java. Object-oriented. *Imperative.*

Background: imperative programming

- ▶ Associated with *stateful* computations.
- ▶ A program is a series of *statements* that affect the program state.

Most languages you've used until now belong this paradigm.

Python. *Imperative.* Object-oriented. Functional.

C. Procedural. *Imperative.*

Java. Object-oriented. *Imperative.*

And many, many more.

The overarching theme of this course

Introduction to various
programming paradigms through the
lens of *functional programming*

Functional programming is a sub-paradigm of declarative programming.

Goals of this course

Goals of this course

- ▶ Thoroughly introduce you to fundamental concepts in programming languages.

Goals of this course

- ▶ Thoroughly introduce you to fundamental concepts in programming languages.

Control flow, exceptions, stateful vs stateless computation, objects vs closures, lazy evaluation, polymorphism / generics, higher-order functions, compositionality

Goals of this course

- ▶ Thoroughly introduce you to fundamental concepts in programming languages.

Control flow, exceptions, stateful vs stateless computation, objects vs closures, lazy evaluation, polymorphism / generics, higher-order functions, compositionality

- ▶ Show different ways of reasoning about programs.

Goals of this course

- ▶ Thoroughly introduce you to fundamental concepts in programming languages.

Control flow, exceptions, stateful vs stateless computation, objects vs closures, lazy evaluation, polymorphism / generics, higher-order functions, compositionality

- ▶ Show different ways of reasoning about programs.

Type-checking, tracing, testing, induction

Goals of this course

- ▶ Thoroughly introduce you to fundamental concepts in programming languages.

Control flow, exceptions, stateful vs stateless computation, objects vs closures, lazy evaluation, polymorphism / generics, higher-order functions, compositionality

- ▶ Show different ways of reasoning about programs.

Type-checking, tracing, testing, induction

- ▶ Introduce you to fundamental ideas in programming language design

Goals of this course

- ▶ Thoroughly introduce you to fundamental concepts in programming languages.

Control flow, exceptions, stateful vs stateless computation, objects vs closures, lazy evaluation, polymorphism / generics, higher-order functions, compositionality

- ▶ Show different ways of reasoning about programs.

Type-checking, tracing, testing, induction

- ▶ Introduce you to fundamental ideas in programming language design

Formal grammar, parsing, static and dynamic semantics, type-checking, polymorphism, subtyping, interpretation

Goals of this course

- ▶ Thoroughly introduce you to fundamental concepts in programming languages.
Control flow, exceptions, stateful vs stateless computation, objects vs closures, lazy evaluation, polymorphism / generics, higher-order functions, compositionality
- ▶ Show different ways of reasoning about programs.
Type-checking, tracing, testing, induction
- ▶ Introduce you to fundamental ideas in programming language design
Formal grammar, parsing, static and dynamic semantics, type-checking, polymorphism, subtyping, interpretation
- ▶ Expose you to a new way of thinking about problems.

Goals of this course

- ▶ Thoroughly introduce you to fundamental concepts in programming languages.
Control flow, exceptions, stateful vs stateless computation, objects vs closures, lazy evaluation, polymorphism / generics, higher-order functions, compositionality
- ▶ Show different ways of reasoning about programs.
Type-checking, tracing, testing, induction
- ▶ Introduce you to fundamental ideas in programming language design
Formal grammar, parsing, static and dynamic semantics, type-checking, polymorphism, subtyping, interpretation
- ▶ Expose you to a new way of thinking about problems.
It's like exercise; it's good for you!

Why should you care to learn this stuff?

- ▶ You'll be exposed to new ideas.

Why should you care to learn this stuff?

- ▶ You'll be exposed to new ideas.
- ▶ More of the science, craft, and **art** of programming.

Why should you care to learn this stuff?

- ▶ You'll be exposed to new ideas.
- ▶ More of the science, craft, and **art** of programming.
- ▶ Skills you learn will help you become a better programmer.
Yes, even in other languages. Yes, even Java!
More productive; easier to read and maintain code.

Why should you care to learn this stuff?

- ▶ You'll be exposed to new ideas.
- ▶ More of the science, craft, and **art** of programming.
- ▶ Skills you learn will help you become a better programmer.
Yes, even in other languages. Yes, even Java!
More productive; easier to read and maintain code.
- ▶ It's a prerequisite for some upper-level courses ;)
If you like this stuff, try COMP 523 or COMP 527!

Why should you care to learn this stuff?

- ▶ You'll be exposed to new ideas.
- ▶ More of the science, craft, and **art** of programming.
- ▶ Skills you learn will help you become a better programmer.
Yes, even in other languages. Yes, even Java!
More productive; easier to read and maintain code.
- ▶ It's a prerequisite for some upper-level courses ;)
If you like this stuff, try COMP 523 or COMP 527!
- ▶ It's really, really cool!
Fun, even!

Why should you care to learn this stuff?

- ▶ You'll be exposed to new ideas.
- ▶ More of the science, craft, and **art** of programming.
- ▶ Skills you learn will help you become a better programmer.
Yes, even in other languages. Yes, even Java!
More productive; easier to read and maintain code.
- ▶ It's a prerequisite for some upper-level courses ;)
If you like this stuff, try COMP 523 or COMP 527!
- ▶ It's really, really cool!
Fun, even!
- ▶ You might get a job out of it.
Facebook, Morgan Stanley, Jane Street Capital, O(1) Labs,
Tezos, ... are all companies that use FP.

So how will we achieve our goals?

Three course modules

1. Introduction to a (hopefully) new *programming paradigm*:
functional programming – the first month of classes.

So how will we achieve our goals?

Three course modules

1. Introduction to a (hopefully) new *programming paradigm*: functional programming – the first month of classes.
2. Survey of important language features across languages – the second month of classes.

So how will we achieve our goals?

Three course modules

1. Introduction to a (hopefully) new *programming paradigm*: functional programming – the first month of classes.
2. Survey of important language features across languages – the second month of classes.
3. Writing an interpreter for our own small programming language (evaluator, typechecker) and introducing you to programming language formal semantics – the last month classes.

How will I check that you achieved these goals?

Assessments

Homework. $10 \times 2\% = 20\%$. These are *learning activities*.
Practice problems. Autograded.

How will I check that you achieved these goals?

Assessments

Homework. $10 \times 2\% = 20\%$. These are *learning activities*.
Practice problems. Autograded.

Midterms. $3 \times 10\% = 30\%$. One for each course module.

How will I check that you achieved these goals?

Assessments

Homework. $10 \times 2\% = 20\%$. These are *learning activities*.
Practice problems. Autograded.

Midterms. $3 \times 10\% = 30\%$. One for each course module.

Final. 50%. Consists of 3 evenly-weighted sections, one for each course module. Allows you to retake **up to two** of the midterms.

You are in control of your learning

You are in control of your learning

No homework due dates.

You are in control of your learning

No homework due dates.

- ▶ Less stress for you.

You are in control of your learning

No homework due dates.

- ▶ Less stress for you.
- ▶ Do these the evening after class or the day after to **solidify your learning**.

You are in control of your learning

No homework due dates.

- ▶ Less stress for you.
- ▶ Do these the evening after class or the day after to **solidify your learning**.
- ▶ Hard deadline is the day of the final exam.

You are in control of your learning

No homework due dates.

- ▶ Less stress for you.
- ▶ Do these the evening after class or the day after to **solidify your learning**.
- ▶ Hard deadline is the day of the final exam.

Bidirectional midterm-final weight shifting.

You are in control of your learning

No homework due dates.

- ▶ Less stress for you.
- ▶ Do these the evening after class or the day after to **solidify your learning**.
- ▶ Hard deadline is the day of the final exam.

Bidirectional midterm-final weight shifting.

- ▶ Half the weight of each midterm can shift to the corresponding section of the final.

You are in control of your learning

No homework due dates.

- ▶ Less stress for you.
- ▶ Do these the evening after class or the day after to **solidify your learning**.
- ▶ Hard deadline is the day of the final exam.

Bidirectional midterm-final weight shifting.

- ▶ Half the weight of each midterm can shift to the corresponding section of the final.
- ▶ **The full weight** of each final exam section can shift to the corresponding midterm.

You are in control of your learning

No homework due dates.

- ▶ Less stress for you.
- ▶ Do these the evening after class or the day after to **solidify your learning**.
- ▶ Hard deadline is the day of the final exam.

Bidirectional midterm-final weight shifting.

- ▶ Half the weight of each midterm can shift to the corresponding section of the final.
- ▶ **The full weight** of each final exam section can shift to the corresponding midterm.
- ▶ If you do well on all midterms, **you do not need to write the final**.

You are in control of your learning

No homework due dates.

- ▶ Less stress for you.
- ▶ Do these the evening after class or the day after to **solidify your learning**.
- ▶ Hard deadline is the day of the final exam.

Bidirectional midterm-final weight shifting.

- ▶ Half the weight of each midterm can shift to the corresponding section of the final.
- ▶ **The full weight** of each final exam section can shift to the corresponding midterm.
- ▶ If you do well on all midterms, **you do not need to write the final**.

One no-questions-asked midterm deferral.

You are in control of your learning

No homework due dates.

- ▶ Less stress for you.
- ▶ Do these the evening after class or the day after to **solidify your learning**.
- ▶ Hard deadline is the day of the final exam.

Bidirectional midterm-final weight shifting.

- ▶ Half the weight of each midterm can shift to the corresponding section of the final.
- ▶ **The full weight** of each final exam section can shift to the corresponding midterm.
- ▶ If you do well on all midterms, **you do not need to write the final**.

One no-questions-asked midterm deferral.

- ▶ **On request** you may choose to skip a midterm, fully shifting its weight to the final.

Instructor – Jacob Thomas Errington



- ▶ Born & raised in Montreal ♡
- ▶ B.Sc. Math & CS @ McGill
- ▶ M.Sc. CS @ McGill - proof theory
- ▶ Worked in fintech; didn't work for me
- ▶ Faculty Lecturer @ McGill since 2022

Fun. Baking, biking, bodybuilding, writing, coding.

Quote. From the man, the myth, the legend, Edsger Dijkstra:

- ▶ “Simplicity is prerequisite for reliability.”

Communication outside of class

We will be using **Discord** for online discussion:

<https://discord.gg/XSVsbPQzc6>



TAs on Discord are assigned a “TA” role and their names appear in **red**.

Teaching assistants

- ▶ We have ~ 9 fantastic teaching assistants this year.
- ▶ Their details will be made available in Discord.
- ▶ Office hours will begin next week.

What does a TA do?

TAs:

- ▶ Prepare homework assignments
- ▶ Prepare the exams
- ▶ Grade the exams

What does a TA do?

TAs:

- ▶ Prepare homework assignments
- ▶ Prepare the exams
- ▶ Grade the exams
- ▶ Hold **office hours** – go to them! It's free tutoring!

What does a TA do?

TAs:

- ▶ Prepare homework assignments
- ▶ Prepare the exams
- ▶ Grade the exams
- ▶ Hold office hours – go to them! It's free tutoring!
- ▶ Answer **questions online** – ask questions!

Ambitious goals

This is not an easy course...
Expectations are high!

How can you succeed in this course?

How can you succeed in this course?

- ▶ Come to class. (Eat beforehand!)

How can you succeed in this course?

- ▶ Come to class. (Eat beforehand!)
- ▶ Participate in class: coding problems, discussions, ask questions

How can you succeed in this course?

- ▶ Come to class. (Eat beforehand!)
- ▶ Participate in class: coding problems, discussions, ask questions
- ▶ Participate outside class: TA office hours, Discord, my office hours
1-on-1 is way better than 1-on-350!

How can you succeed in this course?

- ▶ Come to class. (Eat beforehand!)
- ▶ Participate in class: coding problems, discussions, ask questions
- ▶ Participate outside class: TA office hours, Discord, my office hours
1-on-1 is way better than 1-on-350!
- ▶ Do the homework.

How can you succeed in this course?

- ▶ Come to class. (Eat beforehand!)
- ▶ Participate in class: coding problems, discussions, ask questions
- ▶ Participate outside class: TA office hours, Discord, my office hours
1-on-1 is way better than 1-on-350!
- ▶ Do the homework.
- ▶ (Sadly) practice coding on paper.

How can we help you to succeed?

How can we help you to succeed?

- ▶ Interactive lessons.

How can we help you to succeed?

- ▶ Interactive lessons.
- ▶ Prompt feedback.

How can we help you to succeed?

- ▶ Interactive lessons.
- ▶ Prompt feedback.
- ▶ Learning names.

How can we help you to succeed?

- ▶ Interactive lessons.
- ▶ Prompt feedback.
- ▶ Learning names.
- ▶ **Patience.**

A recipe for success

1. Come to class energized and ready to learn.

A recipe for success

1. Come to class energized and ready to learn.
2. Take a 1h shot at the homework after class.

A library is *right here!*

A recipe for success

1. Come to class energized and ready to learn.
2. Take a 1h shot at the homework after class.
A library is *right here!*
3. Make friends in the class.
You already have one thing in common!

A recipe for success

1. Come to class energized and ready to learn.
2. Take a 1h shot at the homework after class.
A library is *right here!*
3. Make friends in the class.
You already have one thing in common!
4. Participate on Discord.
Ask questions. Answer questions. Don't be shy!

A recipe for success

1. Come to class energized and ready to learn.
2. Take a 1h shot at the homework after class.
A library is *right here!*
3. Make friends in the class.
You already have one thing in common!
4. Participate on Discord.
Ask questions. Answer questions. Don't be shy!
5. Study hard for the midterms.
This way you can skip the final!

This course: OCaml



OCaml: strongly- and statically-typed, functional

Strongly-typed

- ▶ Values are associated with types that specify their interpretation.
- ▶ Cannot confuse, e.g., an integer with a float.

OCaml: strongly- and statically-typed, functional

Statically-typed

- ▶ OCaml programs are typechecked by the compiler.
- ▶ This prevents a large class of bugs:
 - ▶ Passing the wrong number of arguments to a function.
 - ▶ Passing the wrong kinds of arguments to a function.
- ▶ Downside (?) you can't run the program until it typechecks.

OCaml: strongly- and statically-typed, functional

Functional

- ▶ Functions are **first-class**. They are genuine values, like integers or strings.
- ▶ Functions are the way to create abstractions.

demo