CS4400/5400
Programming λanguages

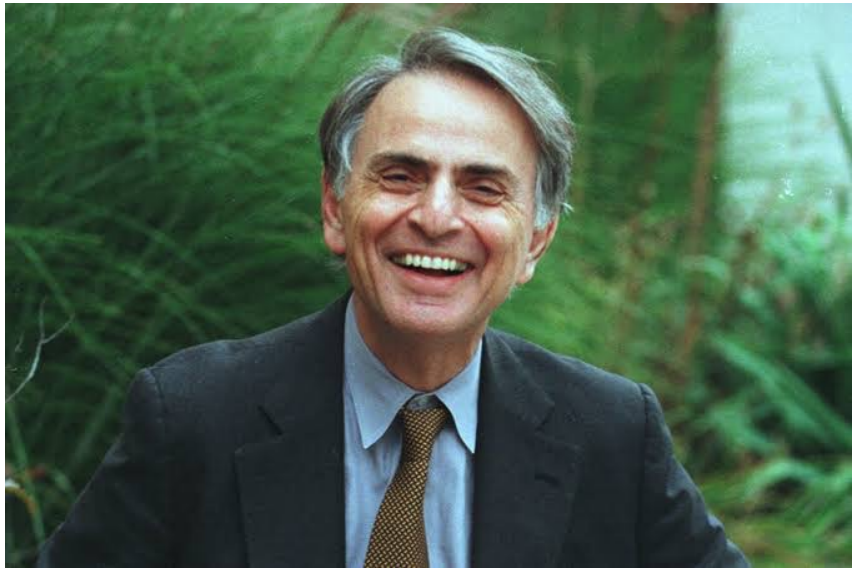
Spring 2024

Instructor: Steven Holtzen

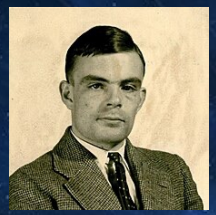
s.holtzen@northeastern.edu

Northeastern University
Khoury College of Computer
and Information Sciences

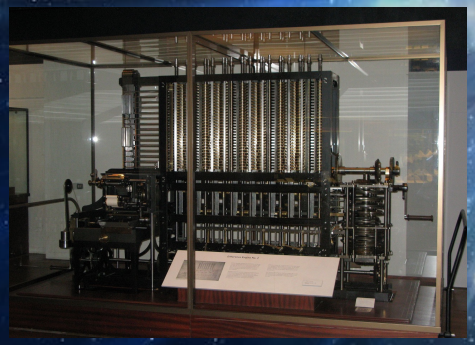
“If you wish to make an apple pie from scratch, you must first invent the universe.”



Carl Sagan
November 9, 1934 -- December 20, 1996



Alan Turing
1912 – 1954
Turing Machine (1936)



Babbage Difference Engine

Ada Lovelace
1815—1842
First Programmer

Alonzo Church
1903 – 1995
Lambda Calculus (1930s)

Tiny Computers, Tiny Languages

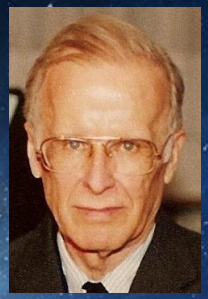
The ENIAC Computer
1945

First **programmable** digital computer

First program:
feasibility of nuclear weapons

AUTODOCDE

1952, Alick Glennie
First **compiled** programming language



FORTRAN
John Backus
1953

LISP
John McCarthy
1960

FLOW-MATIC/COBOL
Grace Hopper
1954

ALGOL

1958
Designed by committee

1950s: The Dawn of the Digital Era

Smalltalk

Object-oriented programming
Mid 1970s

C++

1980

Prolog

Logic programming
1972

C

1972







SQL

1978

ML

Statically typed functional language
1973

Growth of Digital Computers

Python
1990

Haskell
1990

Java
1995

Go
2009

Smalltalk

C++

Prolog

C

Julia
2012

ML

SQL

JavaScript
1995

TypeScript
2012

Rust
2015

Swift
2014

The Modern Era

Questions

Why are there so many programming languages?

Which language should I learn? Should I use?

Are some languages worse than others? Better?
How can I compare them?

What distinguishes one language from another?

Why are new languages being made today?

How are new languages made?

Why study programming languages?

- Be a *more effective programmer*
  - How to *choose languages* for your problems
  - How to *design and implement languages* when needed
- Become equipped to *learn new languages quickly*
- Be *prepared for an evolving world*
  - New languages are showing up all the time
- Enjoy an *aesthetic journey* through this elegant field (subjective)

What is a “programming language”?

“A programming language is a system of notation for writing computer programs.”

https://en.wikipedia.org/wiki/Programming_language
Accessed Friday, January 5

What is a “programming language”?

“Computer programming language, any of various languages for expressing a set of detailed instructions for a digital computer.”

https://www.britannica.com/technology/computer-programming-language
Accessed Friday, January 5

A programming language has two parts:

Syntax

What does a program look like?

Ancient Mesopotamia

Semantics

What does a program do?

"Your debt
is canceled"

A programming language has two parts:

Syntax

What does a program look like?

Python

```
x = 5
print(x)
```

Semantics

What does a program do?

- Create a variable called “x”
- Print the contents of that variable

A programming language has two parts:

Syntax

What does a program look like?

JavaScript

```
let x = 5;
console.log(x)
```

Semantics

What does a program do?

- Create a variable called “x”
- Print the contents of that variable

A programming language has two parts:

Syntax

What does a program look like?

OCaml

```
let x = 5 in
Format.printf “%s” x
```

Semantics

What does a program do?

- Create a variable called “x”
- Print the contents of that variable

This course is all about *precisely defining programming languages*

Syntax

What does a program look like?

Formal descriptions as grammars

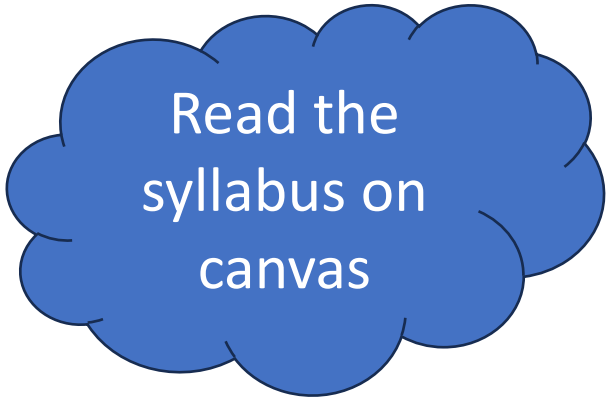
Semantics

What does a program do?

Programs that run programs

Interpreters!

- Grow big languages out of small ones
- Implement new languages



Course Logistics & Content

- Course resources, staff, and policies
- Course modules and overview
- Grading and evaluation

Course Staff

- Instructor: Steven Holtzen
  - Assistant Professor at Northeastern since 2021
  - This is my first time teaching this course

- Teaching Assistants

Minsung Cho
PhD. Student
minsung@ccs.neu.edu

Sam Stites
PhD. Student
stites.s@northeastern.edu

Jack Czenszak
PhD. Student
czenszak.j@northeastern.edu

You are at one of the best schools for PL in the world

https://prl.khoury.northeastern.edu/people.html

Many of the tools we use in this class were developed here!

Module 0: Plait

"Most programming languages — including the most widely used — have serious design defects, so that learning such languages is less a matter of mastering a style than of learning workarounds for the language designer's mistakes... I believe that the most reasonable approach to this problem is to first learn to program in a single well-designed programming language (or perhaps a small number of stylistically varied well-designed languages) that imposes a minimal number of obstacles to the programming task"

John C. Reynolds
1935 -- 2017

Some Thoughts on Teaching Programming and Programming Languages, 2012

Module 0: Plait Learning Objectives

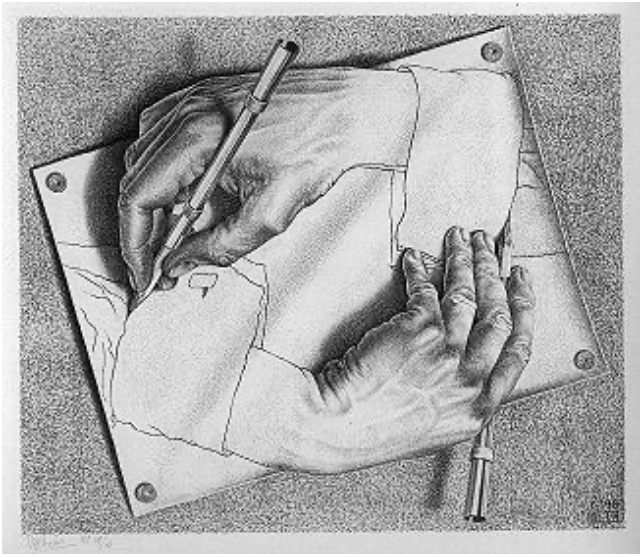
1. Become familiar with *functional programming* and how to solve problems in a *functional style*

2. Become familiar with a *typed language* and *programming with types*

3. Gain practice *learning a new language from scratch*

Module 1: Growing an interpreter

- We will grow the *syntax* and *semantics* of a tiny core language called SMoL (Standard Model of Languages)
  - We will *program interpreters* for this language in Plait
- Language features include:
  - Conditionals, scope and binding, first-class functions, macros, object, state

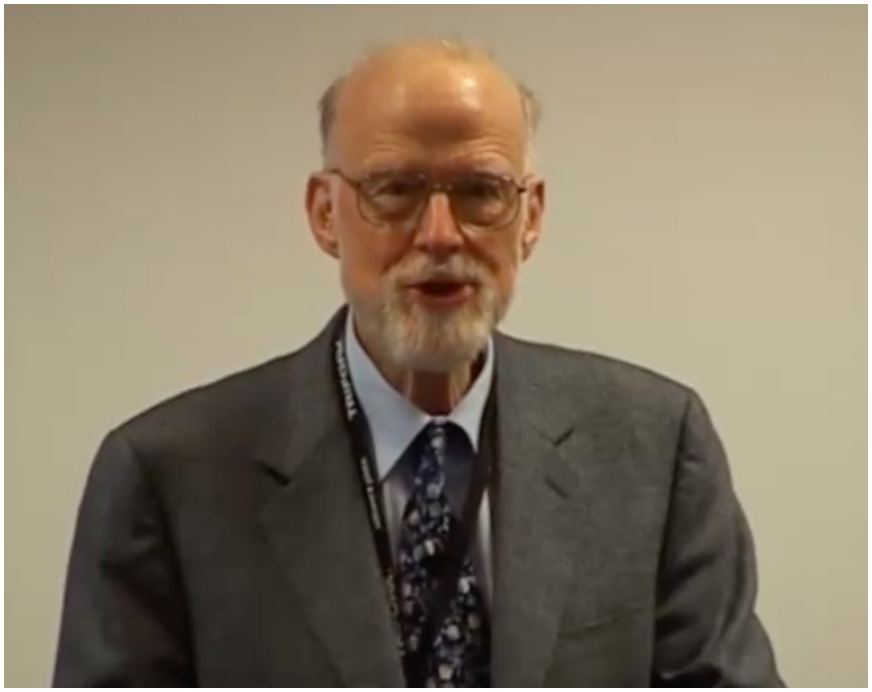


Drawing Hands
M. C. Escher 1968

Module 2: Types

- Types are form of ***checked specification*** for programs
- Example: Java

```
import java.util.Scanner;

public class HelloWorld {

    public static void main(String[] args) {

        // Creates a reader instance which takes
        // input from standard input - keyboard
        Scanner reader = new Scanner(System.in);
        System.out.print("Enter a number: ");

        // nextInt() reads the next integer from the keyboard
        int number = reader.nextInt();

        // println() prints the following line to the output
        System.out.println("You entered: " + number);
    }
}
```



Tony Hoare

Types can't prevent all bugs:
this is a valid Java program:

```
public class Example {

    public static void main(String[] args) {
        Object obj = null;
        obj.hashCode();
    }

}
```

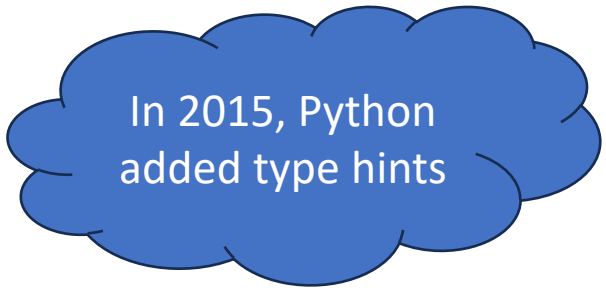
Recommended viewing:

https://www.infoq.com/presentations/Null-References-The-Billion-Dollar-Mistake-Tony-Hoare/

Module 2: Why study types?

- You ***will*** encounter types in your day-to-day programming

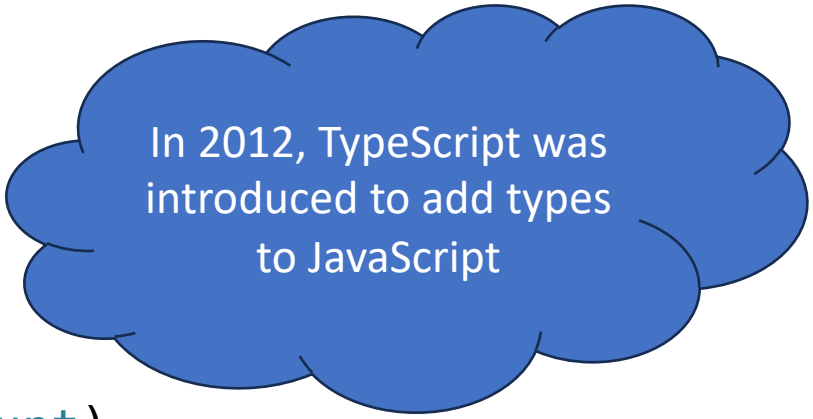
```
def greeting(name: str) -> str:
    return 'Hello ' + name
```



Module 2: Why study types?

- You ***will*** encounter types in your day-to-day programming

```
interface Account {
    id: number
    displayName: string
    version: 1
}

function welcome(user: Account)
{
    console.log(user.id)
}
```



Module 2: Types

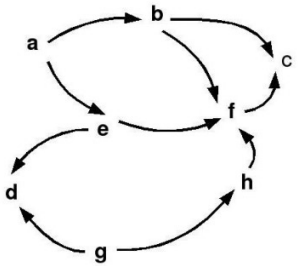
- We will build our own type system for SMoL
  - Learn the formal properties of type systems
- Study ***type inference*** and techniques to make programming with types easier
- Encounter the rich ***mathematical structure*** of types
  - What formal properties can a type system give you?

* Subject to change

Module 3: Beyond SMoL*

- We will study advanced languages and language features that go beyond SMoL
  - Implement and program some interesting languages

```
a(g,h).
a(g,d).
a(e,d).
a(h,f).
a(e,f).
a(a,e).
a(a,b).
a(b,f).
a(b,c).
a(f,c).
```



Logic programming

Laziness

Goal: Broaden your horizons on what programming languages can look like

Schedule overview

- Available here:
  https://docs.google.com/spreadsheets/d/e/2PACX-1vQNTDbNs-WnG7YU5iebhT9XuWfTNF2LBSPWzU1ctif8YrNuciQWZDtzU2hviaFt22asf1C2O27tdOoe/pubhtml?gid=0&single=true

CS4400 Spring'24 Schedule : Sheet1

Module	Date	Lecture title	Topics	Resources
Introduction to Plait and Functional Programming	Monday Jan 8	Introduction and course overview	- Why programming languages? - A first look at Racket/Plait	https://docs.racket-lang.org/plait/Tutorial.html Sections 1.1 -- 1.5
	Wednesday Jan 10	Programming in Racket and Plait	- Solving problems functionally and recursively - How to write tests in Plait - How Plait's type system works	https://docs.racket-lang.org/plait/Tutorial.html Sections 1.1 -- 1.8
	Monday Jan 15	**No Class --- Martin Luther King Jr. Day**		
	Wednesday Jan 17	**No Class --- Steven Traveling for POPL**	- Optional in-class workshop on Plait	
Building a SMoL Interpreter	Monday Jan 22	Abstract Syntax	- Abstract syntax trees - A simple calculator language - Interpreting a language by hand	PLAI pg. 17 -- 27 See https://www.plai.org/3/2/PLAI%20Version%203.2.2%20printing.pdf
	Wednesday Jan 24	Evaluation	- Implementing and testing the evaluator - Parsing and s-expressions	PLAI pg. 28 -- 37
	Monday Jan 29	Conditionals	- Extending the syntax with `if` - Design space of `if` - Adding Booleans, the `Value` type	PLAI pg. 37 -- 47
	Wednesday Jan 31	Local binding	- The `let` syntax - Scope - An evaluator for `let`	PLAI pg. 47 -- 57
	Monday Feb 5	First-class functions	- Syntax for functions - Adding functions to the `Value` type - Evaluating functions	PLAI pg. 58 -- 69
	Wednesday Feb 7	Growing SMoL: Macros	- Desugaring - An example: Strict If - define-syntax - Macro stepping in DrRacket	PLAI pg. 71 -- 84
	Monday Feb 12	Objects I	- The "standard model" of objects - State - Access control	PLAI pg. 85 -- 95

Schedule overview

Types	Wednesday Feb 14	Objects II	- Extending objects: mixins, traits	PLAI pg. 97 -- 106
	Monday Feb 19	Introduction to types	- What are types? - A simple type checker - How to read and write typing judgments	PLAI pg. 109 -- 122
	Wednesday Feb 21	Typing functions	- The typing rule for functions - Assume-guarantee reasoning - Making a typechecker - Handling recursion	PLAI pg. 123 -- 132
	Monday Feb 26	The Simply Typed Lambda Calculus	- Syntax and a type checker - The Omega term and normalization	
	Wednesday Feb 28	Safety and soundness	- What is type safety, why do you want it - Enforcing type safety - Type safety for simply-typed lambda calculus	PLAI pg. 133 -- 144
	Monday Mar 4	**No Class -- Spring Break**		
	Wednesday Mar 6	**No Class -- Spring Break**		
	Monday Mar 11	Type inference	- Basic goals of type inference - Hindley-Milner - Complexity of type inference	PLAI pg. 145 -- 149
	Wednesday Mar 13	Algebraic datatypes and pairs	- Typechecking algebraic datatypes and pairs - Proofs and programs: Curry-Howard	PLAI pg. 150 -- 153
	Monday Mar 18	Subtyping	- Adding subtyping to typing judgments - Applications: information flow analysis	PLAI pg. 165 -- 170
	Wednesday Mar 20	Gradual typing	- TypeScript, typed Python, Typed Racket	PLAI pg. 170 -- 176
Paradigms	Monday Mar 25	Logic Programming I	- Programming with relations - Unification - A simple type checker	PLAI pg. 178 -- 184
	Wednesday Mar 27	Logic Programming II		PLAI pg. 193 -- 202
	Monday Apr 1	Laziness I	- Evaluation schemes: eager, lazy, call-by-need, call by name - Consequences of evaluation schemes - A lazy evaluator - Programming in lazy languages	
	Wednesday Apr 3	Laziness II	- Modeling state and mutation - A taste of Haskell	
	Monday Apr 8	Effects I	- Effects in Racket - Effect handlers	
	Wednesday Apr 10	Effects II		
	Monday Apr 15	**No Class -- Patriot's day**		
	Wednesday Apr 17	Slack day		

Textbook

- *Programming Languages: Application and Interpretation*, Third Edition, by Shriram Krishnamurthy

- Open-access textbook, available at: https://www.plai.org/

- We will follow it closely: you are encouraged to read the textbook sections ahead or after lecture as a reference

Prerequisites

- Required: CS3500 (Object oriented design) or Equivalent

- Highly recommended:

  - Experience programming in at least one major programming language

  - This will be a programming-intensive course

Input/Output

- Important course announcements will be broadcast as Canvas Announcements
  - You should check to make sure that you receive these
- We will use Piazza
  - Please ask all course questions there
  - You can ask private questions
  - You should have received an email invitation
- Notes/slides will be posted on Canvas after lecture

Assignments and grading

Type	Frequency	Percent of Final Grade
Homework	About once a week	60%
Quizzes	About 4	40%

Range	> 93	[90,93)	[87, 90)	[83, 87)	[80, 83)	[77, 80)	[73, 77)	[70, 73)	[67, 70)	[60, 67)	<50
Grade	A	A-	B+	B	B-	C+	C	C-	D+	D	F

Note: Some assignments / problems will be marked as “CS5400”. These are only for students enrolled in CS5400.

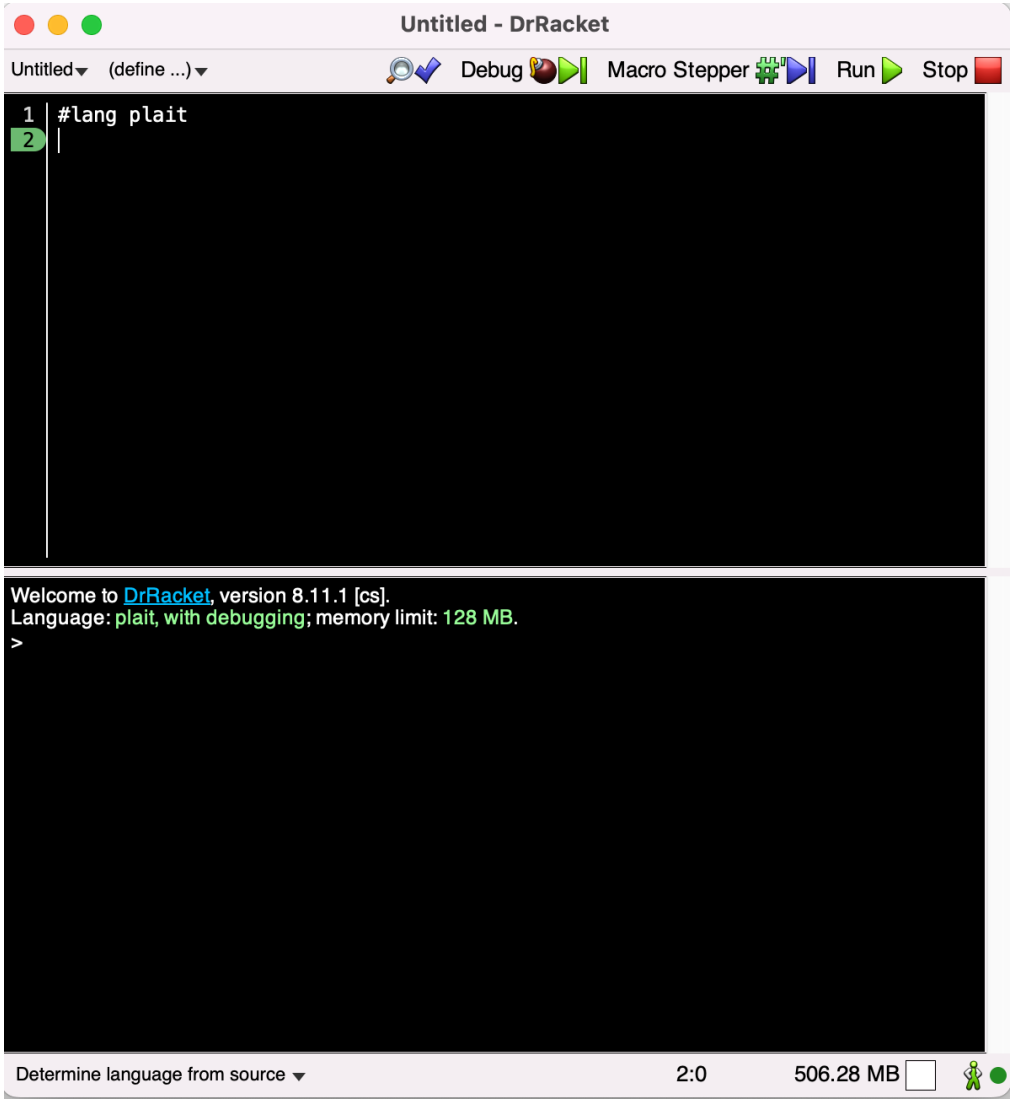
A taste of Plait

Goals:

- Set up and install Plait
- Understand how to write small Plait programs and call functions
- Know the key properties of Plait

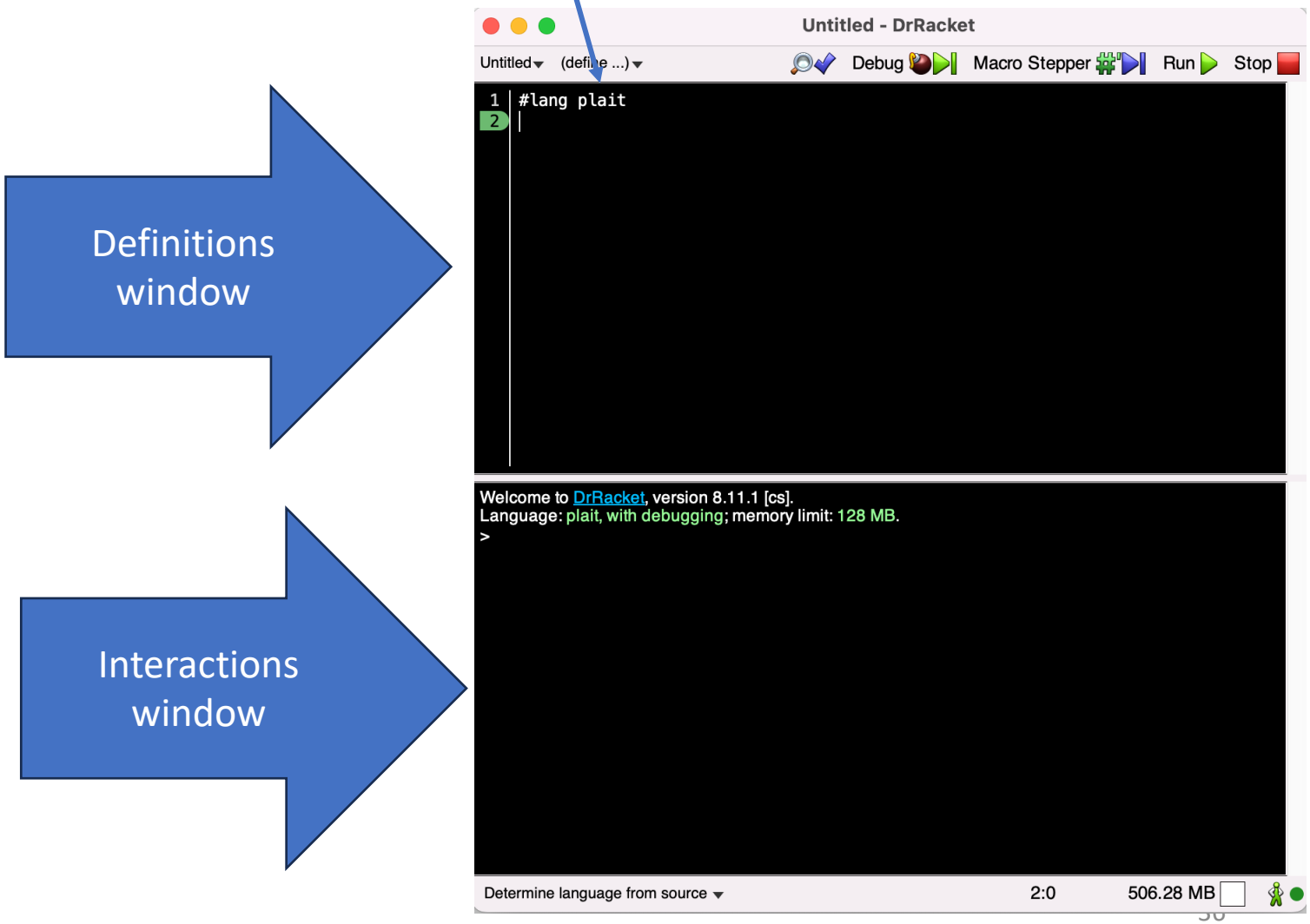
Setting up Plait

- Follow https://docs.racket-lang.org/plait/getting-started.html
- Download DrRacket (use version 8.11.1)
- You will need to install the Plait .plt file



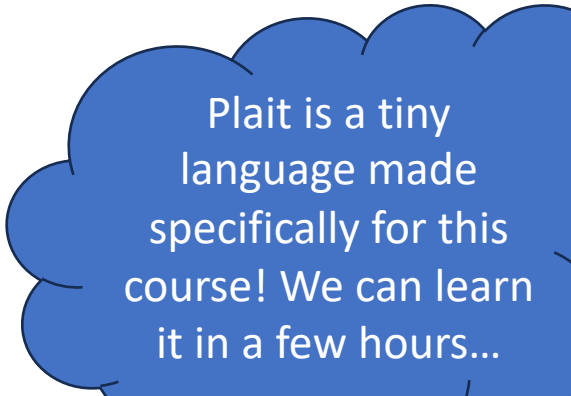
DrRacket

Tells DrRacket which programming language to use



Plait Resources

- Plait website: https://docs.racket-lang.org/plait/
- Plait tutorial: https://docs.racket-lang.org/plait/Tutorial.html
- YouTube video tutorial: https://www.youtube.com/playlist?list=PLbdXd8eufjyUT8rza1qDcS0RUnRTr9A1f
- Plait CheatSheet on Canvas
- Ask on Piazza if you get stuck :)



Simple data

- Enter code into the Interaction window and press “enter” to run your code
- Basic data types:
  - Numbers `1, 1.2, 1/3`
  - Strings “`quoted`”
  - Symbols `‘a, ‘b, ‘sym, …`
  - Booleans `#t #f`
- Comments begin with `;`

Input Plait program called a **term** or **expression**

Type of entered term; every term has a type

Value program evaluates to

An unusual type; we will see more of it later

```
> 4
- Number
4
> "hello"
- String
"hello"
> 'a
- Symbol
'a
> #t
- Boolean
#t
> #f
- Boolean
#f
```

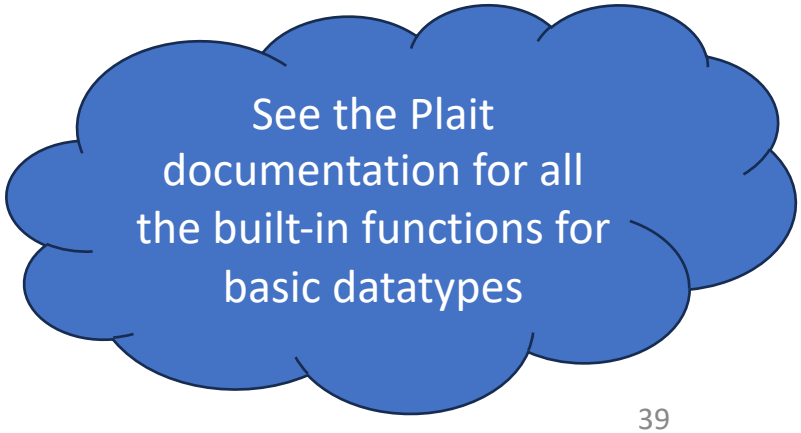
Calling functions

- Adding two numbers:
  - Unusual syntax! We will come to appreciate it.

```
> (+ 1 2)
- Number
3
```

- Some built-in functions for integers:

```
> (max 1 2)
- Number
2

> (- 2 3)
- Number
-1

> (< 10 20)
- Boolean
#t
```



Calling functions

- Chaining together functions:

```
> (+ 1 (max 3 4))
– Number
5
```

- A note on floating point:

```
> (eq? (/ 1.0 3.0) 1/3)
– Boolean
#f
```

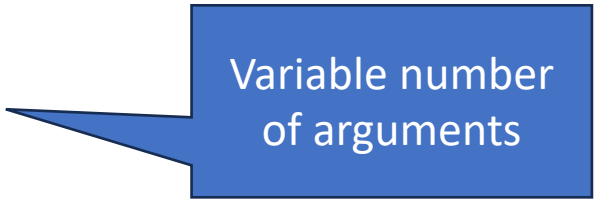
- Be careful! Plait represents these two values differently

```
> 1/3
– Number
1/3
```

```
> (/ 1.0 3.0)
– Number
0.3333333333333333
```

Some functions on Booleans

```
> (and #t #f)
- Boolean
#f

> (and #t #t #f)
- Boolean
#f

> (or #f #t)
- Boolean
#t

> (not #t)
- Boolean
#f

> (eq? #t #f)
- Boolean
#f
```



Some functions on Strings

```
> (equal? "hello" "world")
- Boolean
#f

> (string-append "hello"
"world")
- String
"helloworld”

> (substring "hello" 1 3)
- String
"el"
>
```

Get substring between index 1 and 3, inclusive lower-bound, exclusive upper-bound

Types

- Plait functions expect their arguments to have certain types

Type signature says the max function takes two Number as input and returns a Number

```
> max
- (Number Number -> Number)
#<procedure:max>
```

- If you call a function with the wrong type of arguments, Plait will help you by complaining

```
> (max 1 "oops")
typecheck failed: Number vs. String in:
max
"oops"
```

`if` expressions

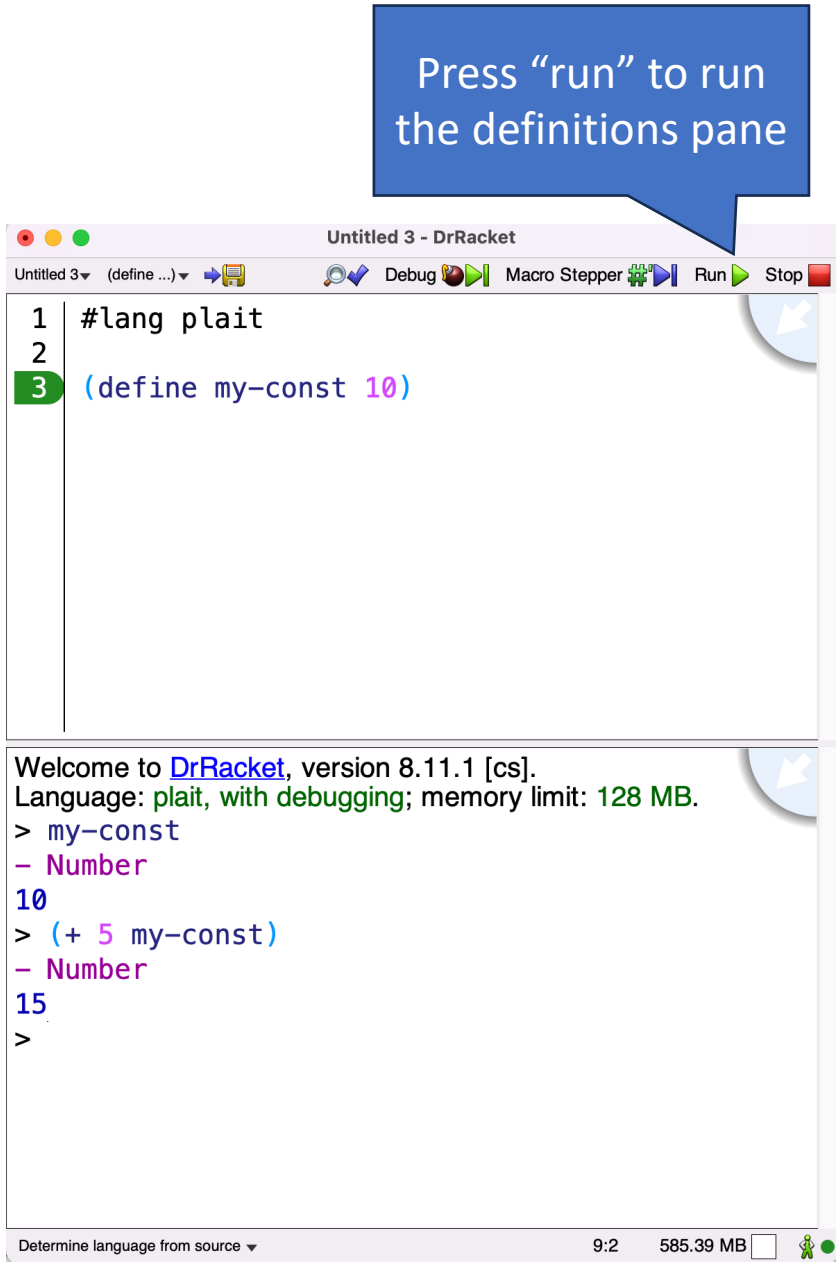
`(if guard thn els)`
If the guard is true, evaluate thn; otherwise, evaluate els

```
> (if #t "woohoo" "ohno")
- String
"woohoo"

> (if #t "what" 10)
typecheck failed: String vs. Number in:
  "what"
  10

> (if (< 10 20) 'ok 'oops)
- Symbol
'ok
```

`if` requires that both `thn` and `els` terms be the same type

Definitions

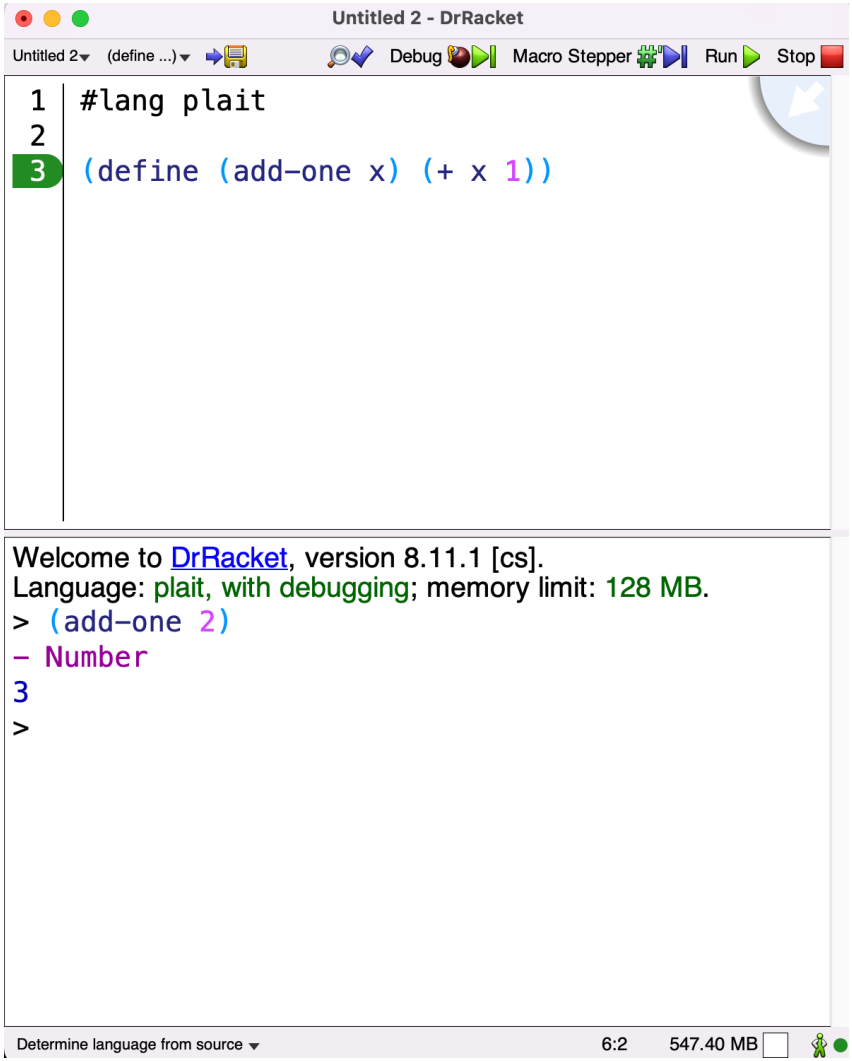
- Syntax:
  `(define id e)`
- `id` is an identifier, `e` is a Plait expression
- Creates a globally accessible constant called `id`



Defining functions

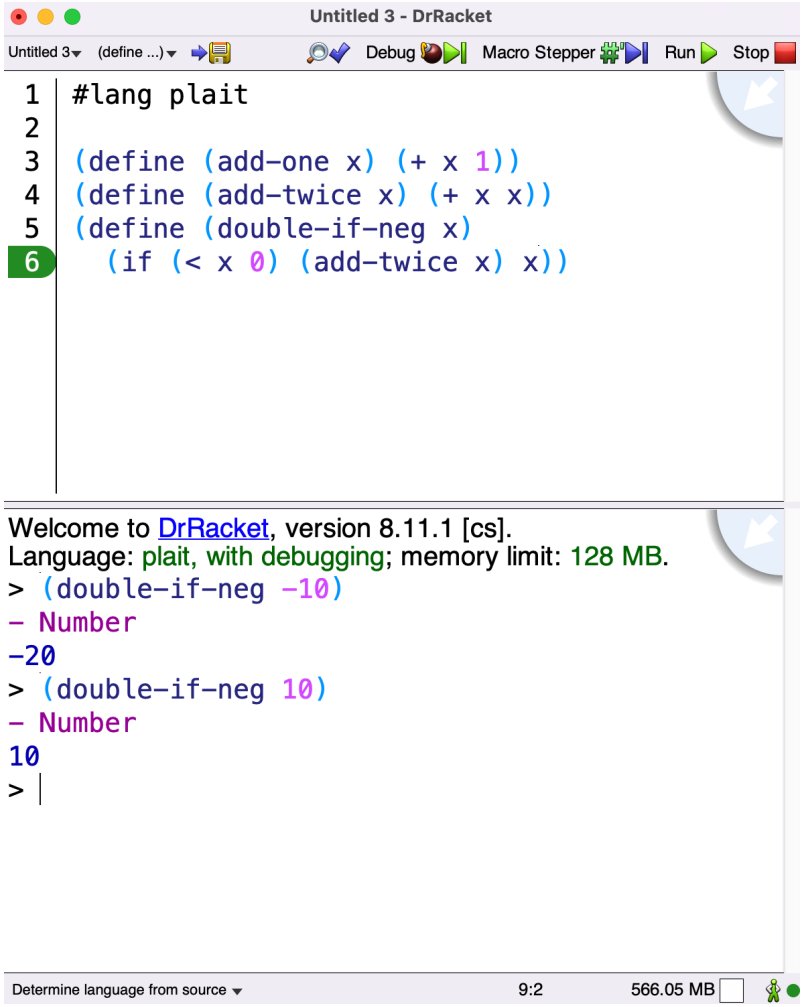
- Syntax:
  `(define (id arg1 arg2 …) e)`

- Creates a globally accessible function called `id` with arguments `arg1, arg2, …` and body `e`



Defining functions

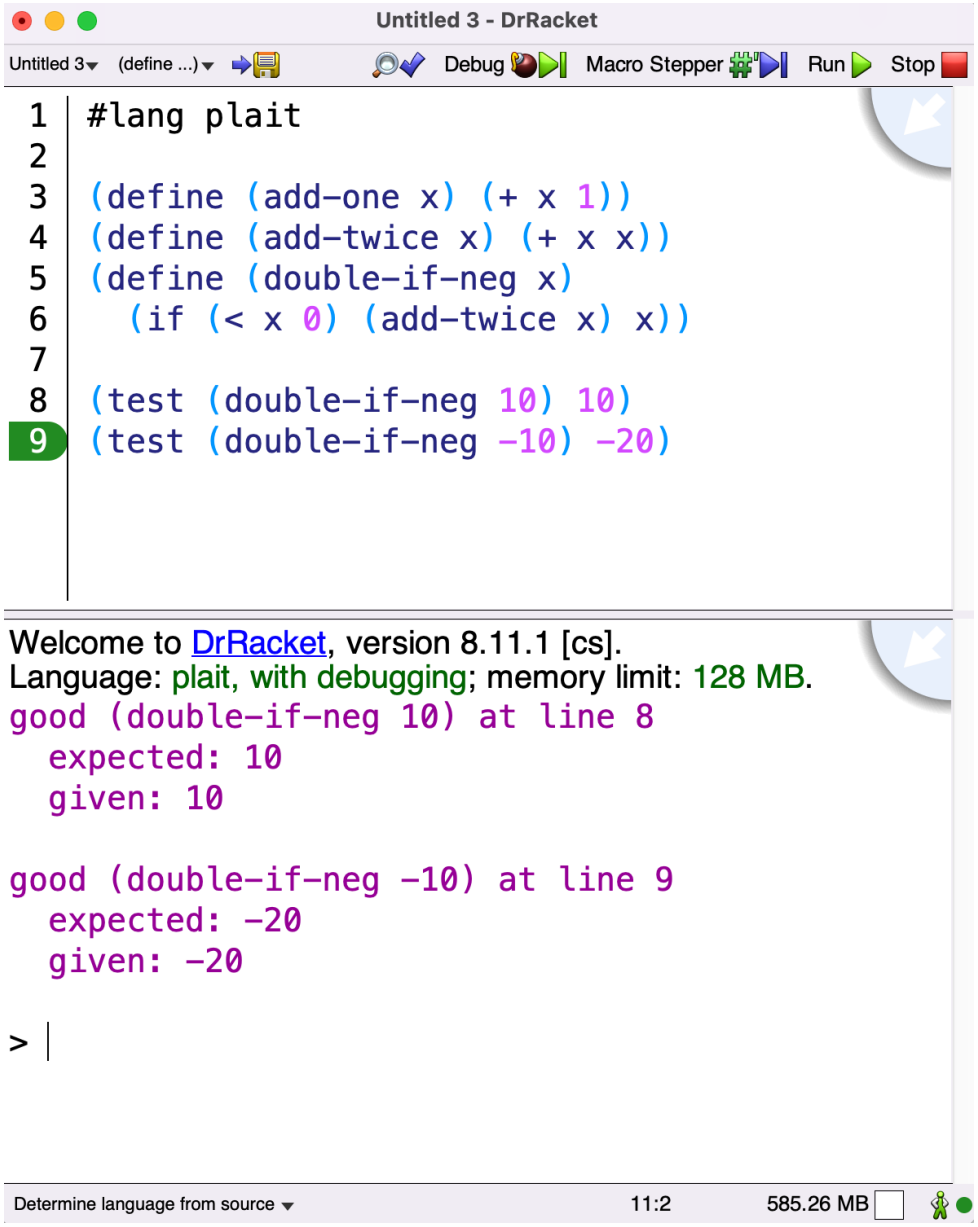
- Write a function “double-if-neg” that takes a number as an argument, and returns double that number if it is negative, otherwise return the input number



Testing

`(test e1 e2)`

tests that e1 and e2 evaluate to the same value



Conclusion

- Homework is due this Friday (Jan 12)

- Before next lecture:
  - Download DrRacket and install Plait
  - Try the first 2 homework problems

- Next time:
  - Bring a computer that is setup with Plait ready to do some in-class activities
  - We will see more Plait and practice solving more interesting problems