

# Introduction to Type Systems

Michael McThrow

San Jose State University  
Computer Science Department  
CS 152 – Programming Paradigms

October 12, 2020



# Table of Contents

- 1 Introduction to Types
- 2 Strong and Weak Typing
- 3 Static and Dynamic Typing
- 4 Preview of Wednesday's Lecture

# Table of Contents

- 1** Introduction to Types
- 2 Strong and Weak Typing
- 3 Static and Dynamic Typing
- 4 Preview of Wednesday's Lecture

Throughout your years programming, you've encountered and used various data types. Some of these are primitive (such as numbers and Boolean values), some are more abstract (such as data structures like arrays, linked lists, binary search trees, etc.), and some are composite (such as objects defined by a class definition).

In this lecture and the following lecture, we will be covering the workings of *type systems*, which describes how a programming language manages the collection of types it supports and how it handles interactions between different types of data.

# Table of Contents

- 1 Introduction to Types
- 2 Strong and Weak Typing**
- 3 Static and Dynamic Typing
- 4 Preview of Wednesday's Lecture

What should happen when you perform  $2 + 3$ ?

What should happen when you perform  $2 + 3$ ? It is reasonable to answer that the  $+$  operator should perform addition on the integers 2 and 3, which results in the answer 5.



What should happen when you perform  $2.1 + 3$ ?

What should happen when you perform  $2.1 + 3$ ? Depending on the type system of the programming language, there are many possibilities when adding a floating point number and an integer:

What should happen when you perform  $2.1 + 3$ ? Depending on the type system of the programming language, there are many possibilities when adding a floating point number and an integer:

- We could convert the integer into a floating-point value, or we could convert the floating-point value to an integer.

What should happen when you perform  $2.1 + 3$ ? Depending on the type system of the programming language, there are many possibilities when adding a floating point number and an integer:

- We could convert the integer into a floating-point value, or we could convert the floating-point value to an integer.
- We could perform the addition, which results in  $5.1$ . But what type should the result be? Should it be a floating point value (resulting in  $5.1$ ) or an integer (resulting in  $5$ )?

What should happen when you perform  $2.1 + 3$ ? Depending on the type system of the programming language, there are many possibilities when adding a floating point number and an integer:

- We could convert the integer into a floating-point value, or we could convert the floating-point value to an integer.
- We could perform the addition, which results in  $5.1$ . But what type should the result be? Should it be a floating point value (resulting in  $5.1$ ) or an integer (resulting in  $5$ )?
- We could inform the user of a type mismatch error, saying it cannot evaluate the following expression since  $2.1$  and  $3$  are of different types.

What should happen when you perform "2.1" + 1?

What should happen when you perform `"2.1" + 1`?

- We could inform the user of a type mismatch error, saying it cannot evaluate the following expression because `"2.1"` is a string and `1` is an integer.

What should happen when you perform `"2.1" + 1`?

- We could inform the user of a type mismatch error, saying it cannot evaluate the following expression because `"2.1"` is a string and `1` is an integer.
- We could convert `2.1` into a floating-point value and perform the addition `2.1 + 1`, which the output would be either a floating-point value, an integer, or a string.



What should happen when you perform `"2.1" + 1`?

- We could inform the user of a type mismatch error, saying it cannot evaluate the following expression because `"2.1"` is a string and `1` is an integer.
- We could convert `2.1` into a floating-point value and perform the addition `2.1 + 1`, which the output would be either a floating-point value, an integer, or a string.
- We could convert `1` into a string and concatenate the string `"2.1"` with `"1"`, giving the string `"2.11"`.

What should happen when you perform `"2.1" + 1`?

- We could inform the user of a type mismatch error, saying it cannot evaluate the following expression because `"2.1"` is a string and `1` is an integer.
- We could convert `2.1` into a floating-point value and perform the addition `2.1 + 1`, which the output would be either a floating-point value, an integer, or a string.
- We could convert `1` into a string and concatenate the string `"2.1"` with `"1"`, giving the string `"2.11"`.
- We could treat `1` as an index of the beginning of the substring of `"2.1"`, resulting in the string `".1"`.

The answers to these questions depend on the semantics of the *type system* of the programming language.

# Strong Typing

## Definition (Strongly Typed)

A *strongly typed* programming language is one that does not rely on implicit type conversions whenever dealing with type mismatches.

Note that this is not a precise definition; some languages thought of as strongly typed do sometimes perform implicit type conversions, particularly when dealing with combinations of integers and floating-point numbers.

# Examples of Strongly-Typed Programming Languages

- Java
- Scheme (including Racket and Typed Racket)
- Python
- Rust
- Haskell

## Some Examples of "2.1" + 1 in Strongly-Typed Programming Languages

# Scheme/Racket

```
Welcome to DrRacket, version 7.8 [3m].  
Language: racket, with debugging; memory limit: 128 MB.
```

```
> (+ "2.1" 1)
```



```
+: contract violation
```

```
expected: number?
```

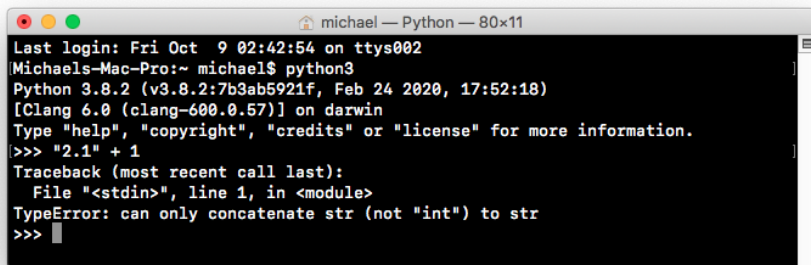
```
given: "2.1"
```

```
argument position: 1st
```

```
other arguments...:
```

```
> |
```

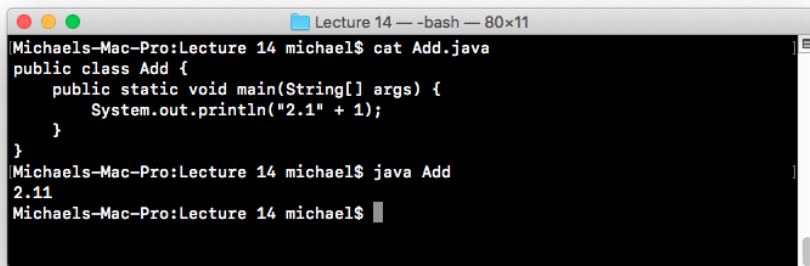
# Python

A terminal window titled "michael — Python — 80x11" with standard macOS window controls (red, yellow, green buttons). The terminal text is as follows:

```
Last login: Fri Oct 9 02:42:54 on ttys002
Michael's-Mac-Pro:~ michael$ python3
Python 3.8.2 (v3.8.2:7b3ab5921f, Feb 24 2020, 17:52:18)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> "2.1" + 1
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can only concatenate str (not "int") to str
>>> █
```

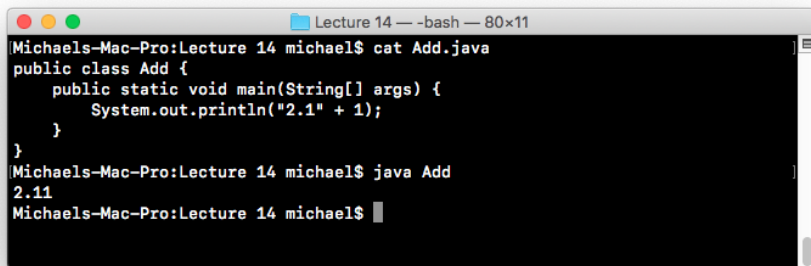


# Java



```
Lecture 14 — -bash — 80x11
Michaels-Mac-Pro:Lecture 14 michael$ cat Add.java
public class Add {
    public static void main(String[] args) {
        System.out.println("2.1" + 1);
    }
}
Michaels-Mac-Pro:Lecture 14 michael$ java Add
2.11
Michaels-Mac-Pro:Lecture 14 michael$
```

# Java

A terminal window titled "Lecture 14 -- -bash -- 80x11" is shown. The prompt is "Michaels-Mac-Pro:Lecture 14 michael\$". The user enters "cat Add.java", and the terminal displays the contents of the file: "public class Add {", " public static void main(String[] args) {", " System.out.println(\"2.1\" + 1);", " }", "}". The user then enters "java Add", and the terminal outputs "2.11". The prompt returns to "Michaels-Mac-Pro:Lecture 14 michael\$".

```
Michaels-Mac-Pro:Lecture 14 michael$ cat Add.java
public class Add {
    public static void main(String[] args) {
        System.out.println("2.1" + 1);
    }
}
Michaels-Mac-Pro:Lecture 14 michael$ java Add
2.11
Michaels-Mac-Pro:Lecture 14 michael$
```

Huh? That worked? But how?



# Java String Concatenation

The reason why `"2.1" + 1` didn't return a type error, but instead resulted in the string `"2.11"` is because in Java the `String` class overloaded the `+` method to handle string concatenation. It allows a string to be concatenated with an integer. Note that the opposite `1 + "2.1"` also works, resulting in `"12.1"`.

# Weak Typing

## Definition (Weakly Typed)

A *weakly typed* programming language is one that relies on implicit type conversions whenever dealing with type mismatches.

Just like the definition of strong typing, this definition is also imprecise.

# Examples of Weakly-Typed Programming Languages

- C
- C++
- JavaScript

## Some Examples of "2.1" + 1 in Weakly-Typed Programming Languages

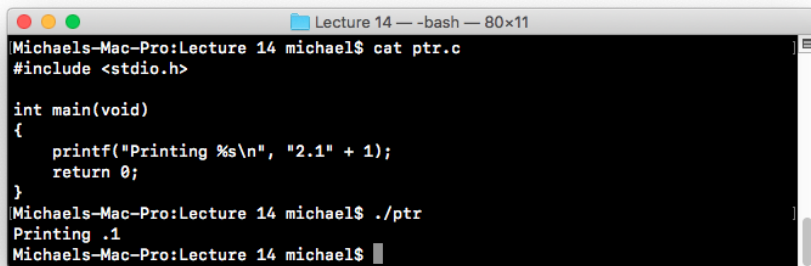
# JavaScript

```
> "2.1" + 1
```

```
< "2.11" = $1
```

```
> |
```

## C



```
Lecture 14 -- -bash -- 80x11
Michaels-Mac-Pro:Lecture 14 michael$ cat ptr.c
#include <stdio.h>

int main(void)
{
    printf("Printing %s\n", "2.1" + 1);
    return 0;
}
Michaels-Mac-Pro:Lecture 14 michael$ ./ptr
Printing .1
Michaels-Mac-Pro:Lecture 14 michael$
```



# Explanation of C

- In C, strings are considered arrays of characters.
- In C, arrays can be indexed in one of two ways:
  - By standard array indexing syntax (e.g., `greeting[1]`)
  - By using pointer arithmetic (e.g., `*(greeting + 1)`)
- An array in C is a pointer to a contiguous block of memory.
- `"2.1" + 1` is an example of pointer arithmetic. What this does results in a new pointer that starts at where `."` is located in memory. Since strings in C are null-terminated, the string starting at the location of `."` is `".1"`.

# Conclusions Regarding Strong and Weak Typing

What precisely defines a strongly- and weakly-typed language is a bit nebulous since some strongly typed languages do perform implicit type conversions on some types, but keep in mind that the definitions of strong and weak typing are general and not exact.

# Table of Contents

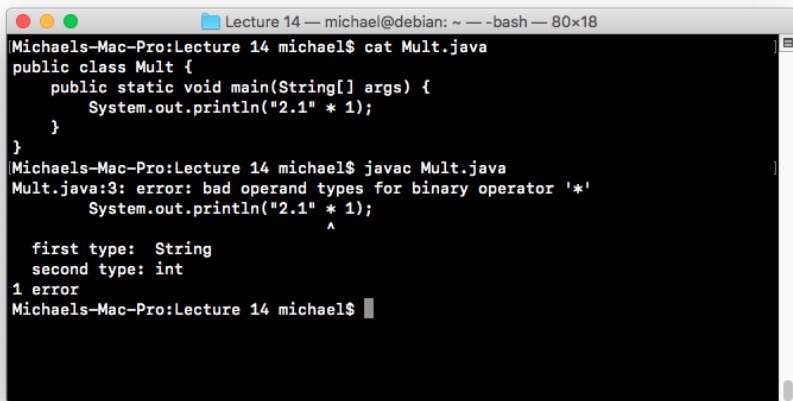
- 1 Introduction to Types
- 2 Strong and Weak Typing
- 3 Static and Dynamic Typing**
- 4 Preview of Wednesday's Lecture

# Static Typing

## Definition (Static Typing)

A *statically-typed* language is one where its type interactions can be checked without executing the program.

# Example of Static Type Checking in Java



```
Lecture 14 — michael@debian: ~ -- -bash -- 80x18
[Michaels-Mac-Pro:Lecture 14 michael$ cat Mult.java
public class Mult {
    public static void main(String[] args) {
        System.out.println("2.1" * 1);
    }
}
[Michaels-Mac-Pro:Lecture 14 michael$ javac Mult.java
Mult.java:3: error: bad operand types for binary operator '*'
    System.out.println("2.1" * 1);
                          ^
    first type: String
    second type: int
1 error
Michaels-Mac-Pro:Lecture 14 michael$
```

Static checkers such as compilers and linters can detect incorrect code usages like this:

```
int x = "5";
```

# Dynamic Typing

## Definition (Dynamic Typing)

A *dynamically-typed* language is one where its type interactions are checked at run-time.

# Classification of Languages

	Static	Dynamic
Strong	Java, Rust, Haskell, Typed Racket	Python, Scheme/Racket
Weak	C, C++	JavaScript

Note that the concepts of strong and weak typing are orthogonal to the concepts of static and dynamic typing.



# Pros and Cons of Static Typing

- Pro: Static typing allows programmers to check type compatibility problems without having to run the program (a very big deal for long-running programs).
- Pro: Static typing makes it easier for programmers to read code, since they know the parameter and return types of functions just by looking at function definitions.
- Con: Static typing requires programmers to think about and commit to types when designing their programs.

# Pros and Cons of Dynamic Typing

- Pro: Dynamically-typed programming languages tend to be oriented toward rapid application development by not having to pre-commit to data types in advance.
- Con: Because types are not checked until runtime, programmers need to be more diligent about testing to make sure programs don't crash due to type errors.
- Con: Dynamically-typed code tends to be harder to reason about (this is subjective, however).

# Table of Contents

- 1 Introduction to Types
- 2 Strong and Weak Typing
- 3 Static and Dynamic Typing
- 4 Preview of Wednesday's Lecture**

# Typed Racket

Typed Racket is a variant of Racket that adds static types checks. Typed Racket is a *gradually-typed* programming language; its substrate is still dynamically-typed, but programmers are free to add static type information to their variable and function definitions as they like, which means that users could combine the advantages of both static and dynamic typing.

# Nuts and Bolts of Type Systems

How would we implement a type system for a programming language? How would we write a static type checker? The next lecture will cover these topics.