Evaluation Strategies
CS 6371: Advanced Programming Languages

Kevin W. Hamlen

April 13, 2023
Definition (first-class): A type is said to be *first-class* for a programming language if values of that type require no special syntax or encapsulation to be

- assigned to variables,
- passed as arguments,
- returned by functions,
- any other type-agnostic usages.

Which of the following languages have first-class functions?

- C
- C++
- SIMPL
- Java
- JavaScript
- Python
- \( \lambda \rightarrow \)
- System F
- OCaml
- Haskell
**Definition (first-class):** A type is said to be *first-class* for a programming language if values of that type require no special syntax or encapsulation to be

- assigned to variables,
- passed as arguments,
- returned by functions,
- any other type-agnostic usages.

Which of the following languages have first-class functions?

- C 🗿
- C++ 🗿
- SIMPL
- Java
- JavaScript
- Python
- λ →
- System F
- OCaml
- Haskell
**Definition (first-class):** A type is said to be *first-class* for a programming language if values of that type require no special syntax or encapsulation to be

- assigned to variables,
- passed as arguments,
- returned by functions,
- any other type-agnostic usages.

Which of the following languages have first-class functions?

- C  
- C++  
- SIMPL  
- Java  
- JavaScript  
- Python  
- λ$\to$  
- System F  
- OCaml  
- Haskell
**Definition (first-class):** A type is said to be *first-class* for a programming language if values of that type require no special syntax or encapsulation to be

- assigned to variables,
- passed as arguments,
- returned by functions,
- any other type-agnostic usages.

Which of the following languages have first-class functions?

- C
- C++
- SIMPL
- Java
- JavaScript
- Python
- λ→
- System F
- OCaml
- Haskell
Definition (first-class): A type is said to be *first-class* for a programming language if values of that type require no special syntax or encapsulation to be

- assigned to variables,
- passed as arguments,
- returned by functions,
- any other type-agnostic usages.

Which of the following languages have first-class functions?

- C: ✗
- C++: ✗
- SIMPL: ✓
- Java: ✗
- JavaScript: ✗
- Python: ✗
- λ →
- System F: ✓
- OCaml: ✓
- Haskell: ✓
Definition (first-class): A type is said to be first-class for a programming language if values of that type require no special syntax or encapsulation to be

- assigned to variables,
- passed as arguments,
- returned by functions,
- any other type-agnostic usages.

Which of the following languages have first-class functions?

- C  ❌
- C++  ❌
- SIMPL  ✔
- Java  ❌
- JavaScript  ✔
- Python
- λ →
- System F
- OCaml
- Haskell
**First-class**

**Definition (first-class):** A type is said to be *first-class* for a programming language if values of that type require no special syntax or encapsulation to be

- assigned to variables,
- passed as arguments,
- returned by functions,
- any other type-agnostic usages.

Which of the following languages have first-class functions?

- C
  - x
- C++
  - x
- SIMPL
  - ✓
- Java
  - x
- JavaScript
  - ✓
- Python
  - ✓
- \( \lambda \rightarrow \)
- System F
- OCaml
- Haskell
First-class

Definition (first-class): A type is said to be first-class for a programming language if values of that type require no special syntax or encapsulation to be

- assigned to variables,
- passed as arguments,
- returned by functions,
- any other type-agnostic usages.

Which of the following languages have first-class functions?

<table>
<thead>
<tr>
<th>Language</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>✗</td>
</tr>
<tr>
<td>C++</td>
<td>✗</td>
</tr>
<tr>
<td>SIMPL</td>
<td>✓</td>
</tr>
<tr>
<td>Java</td>
<td>✗</td>
</tr>
<tr>
<td>JavaScript</td>
<td>✓</td>
</tr>
<tr>
<td>Python</td>
<td>✓</td>
</tr>
<tr>
<td>λ</td>
<td>✓</td>
</tr>
<tr>
<td>System F</td>
<td></td>
</tr>
<tr>
<td>OCaml</td>
<td></td>
</tr>
<tr>
<td>Haskell</td>
<td></td>
</tr>
</tbody>
</table>
First-class

**Definition (first-class):** A type is said to be *first-class* for a programming language if values of that type require no special syntax or encapsulation to be

- assigned to variables,
- passed as arguments,
- returned by functions,
- any other type-agnostic usages.

Which of the following languages have first-class functions?

- C  X
- C++  X
- SIMPL  ✓
- Java  X
- JavaScript  ✓
- Python  ✓
- λ → ✓
- System F  ✓
- OCaml  ✓
- Haskell  ✓
**Definition (first-class):** A type is said to be *first-class* for a programming language if values of that type require no special syntax or encapsulation to be

- assigned to variables,
- passed as arguments,
- returned by functions,
- any other type-agnostic usages.

Which of the following languages have first-class functions?

- C
- C++
- SIMPL✓
- Java✗
- JavaScript✓
- Python✓
- $\lambda \rightarrow$✓
- System F✓
- OCaml✓
- Haskell✓
First-class

Definition (first-class): A type is said to be first-class for a programming language if values of that type require no special syntax or encapsulation to be

- assigned to variables,
- passed as arguments,
- returned by functions,
- any other type-agnostic usages.

Which of the following languages have first-class functions?

<table>
<thead>
<tr>
<th>Language</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>✗</td>
</tr>
<tr>
<td>C++</td>
<td>✗</td>
</tr>
<tr>
<td>SIMPL</td>
<td>✓</td>
</tr>
<tr>
<td>Java</td>
<td>✗</td>
</tr>
<tr>
<td>JavaScript</td>
<td>✓</td>
</tr>
<tr>
<td>Python</td>
<td>✓</td>
</tr>
<tr>
<td>$\lambda \to$</td>
<td>✓</td>
</tr>
<tr>
<td>System F</td>
<td>✓</td>
</tr>
<tr>
<td>OCaml</td>
<td>✓</td>
</tr>
<tr>
<td>Haskell</td>
<td>✓</td>
</tr>
</tbody>
</table>
Partial Evaluation

**Definition (Curried):** A multi-argument function is *curried* if it is expressed as a function from each individual argument to a function of the remaining arguments (i.e., has type $\tau_1 \to \cdots \to \tau_n$).

**Definition (Partial Evaluation):** A multi-argument function is *partially evaluated* when it is applied to fewer than its total number of arguments, yielding a function from the remaining arguments to the return value.

Which of the following languages support currying and partial evaluation?

- C
- C++
- SIMPL
- Java
- JavaScript
- Python
- $\lambda$
- System F
- OCaml
- Haskell
Definition (Curried): A multi-argument function is *curried* if it is expressed as a function from each individual argument to a function of the remaining arguments (i.e., has type $\tau_1 \to \cdots \to \tau_n$).

Definition (Partial Evaluation): A multi-argument function is *partially evaluated* when it is applied to fewer than its total number of arguments, yielding a function from the remaining arguments to the return value.

Which of the following languages support currying and partial evaluation?

- C
- C++
- SIMPL
- Java
- JavaScript
- Python
- $\lambda$
- System F
- OCaml
- Haskell

C

Partial Evaluation

**Definition (Curried):** A multi-argument function is *curried* if it is expressed as a function from each individual argument to a function of the remaining arguments (i.e., has type $\tau_1 \to \cdots \to \tau_n$).

**Definition (Partial Evaluation):** A multi-argument function is *partially evaluated* when it is applied to fewer than its total number of arguments, yielding a function from the remaining arguments to the return value.

Which of the following languages support currying and partial evaluation?

- C  
- C++  
- SIMPL  
- Java  
- JavaScript  
- Python  
- $\lambda$  
- System F  
- OCaml  
- Haskell
**Definition (Curried):** A multi-argument function is *curried* if it is expressed as a function from each individual argument to a function of the remaining arguments (i.e., has type $\tau_1 \to \cdots \to \tau_n$).

**Definition (Partial Evaluation):** A multi-argument function is *partially evaluated* when it is applied to fewer than its total number of arguments, yielding a function from the remaining arguments to the return value.

Which of the following languages support currying and partial evaluation?

- C  [x]
- C++  [x]
- SIMPL  [✓]
- Java  [✓]
- JavaScript  [✓]
- Python  [✓]
- λ [✓]
- System F  [✓]
- OCaml  [✓]
- Haskell  [✓]
Partial Evaluation

**Definition (Curried):** A multi-argument function is *curried* if it is expressed as a function from each individual argument to a function of the remaining arguments (i.e., has type $\tau_1 \rightarrow \cdots \rightarrow \tau_n$).

**Definition (Partial Evaluation):** A multi-argument function is *partially evaluated* when it is applied to fewer than its total number of arguments, yielding a function from the remaining arguments to the return value.

Which of the following languages support currying and partial evaluation?

- C: ×
- C++: ×
- SIMPL: ✓
- Java: ×
- JavaScript: ×
- Python
- $\lambda$: ×
- System F
- OCaml
- Haskell
Partial Evaluation

**Definition (Curried):** A multi-argument function is *curried* if it is expressed as a function from each individual argument to a function of the remaining arguments (i.e., has type $\tau_1 \to \cdots \to \tau_n$).

**Definition (Partial Evaluation):** A multi-argument function is *partially evaluated* when it is applied to fewer than its total number of arguments, yielding a function from the remaining arguments to the return value.

Which of the following languages support currying and partial evaluation?

- C  
- C++  
- SIMPL  
- Java  
- JavaScript  
- Python  
- $\lambda$  
- System F  
- OCaml  
- Haskell
Definition (Curried): A multi-argument function is *curried* if it is expressed as a function from each individual argument to a function of the remaining arguments (i.e., has type $\tau_1 \rightarrow \cdots \rightarrow \tau_n$).

Definition (Partial Evaluation): A multi-argument function is *partially evaluated* when it is applied to fewer than its total number of arguments, yielding a function from the remaining arguments to the return value.

Which of the following languages support currying and partial evaluation?

- C  
  - C++  
  - SIMPL  
  - Java  
  - JavaScript  
  - Python  
    - (with functools)  
  - $\lambda$  
  - System F  
  - OCaml  
  - Haskell
Partial Evaluation

**Definition (Curried):** A multi-argument function is *curried* if it is expressed as a function from each individual argument to a function of the remaining arguments (i.e., has type $\tau_1 \rightarrow \cdots \rightarrow \tau_n$).

**Definition (Partial Evaluation):** A multi-argument function is *partially evaluated* when it is applied to fewer than its total number of arguments, yielding a function from the remaining arguments to the return value.

Which of the following languages support currying and partial evaluation?

```
C   
C++  
SIMPL ✓ 
Java  
JavaScript ✓ 
Python ✓ (with functools) 
λ → ✓ 
System F 
OCaml 
Haskell 
```
Definition (Curried): A multi-argument function is *curried* if it is expressed as a function from each individual argument to a function of the remaining arguments (i.e., has type $\tau_1 \to \cdots \to \tau_n$).

Definition (Partial Evaluation): A multi-argument function is *partially evaluated* when it is applied to fewer than its total number of arguments, yielding a function from the remaining arguments to the return value.

Which of the following languages support currying and partial evaluation?

- C: ✗
- C++: ✗
- SIMPL: ✓
- Java: ✗
- JavaScript: ✓
- Python: ✓ (with functools)
- $\lambda$: ✓
- System F: ✓
- OCaml: ✓
- Haskell: ✓
**Partial Evaluation**

**Definition (Curried):** A multi-argument function is *curried* if it is expressed as a function from each individual argument to a function of the remaining arguments (i.e., has type $\tau_1 \rightarrow \cdots \rightarrow \tau_n$).

**Definition (Partial Evaluation):** A multi-argument function is *partially evaluated* when it is applied to fewer than its total number of arguments, yielding a function from the remaining arguments to the return value.

Which of the following languages support currying and partial evaluation?

- C: ×
- C++: ×
- SIMPL: ✓
- Java: ×
- JavaScript: ✓
- Python: ✓ (with functools)
- λ: ✓
- System F: ✓
- OCaml: ✓
- Haskell: ✓
Partial Evaluation

**Definition (Curried):** A multi-argument function is *curried* if it is expressed as a function from each individual argument to a function of the remaining arguments (i.e., has type $\tau_1 \to \cdots \to \tau_n$).

**Definition (Partial Evaluation):** A multi-argument function is *partially evaluated* when it is applied to fewer than its total number of arguments, yielding a function from the remaining arguments to the return value.

Which of the following languages support currying and partial evaluation?

- C: ✗
- C++: ✗
- SIMPL: ✓
- Java: ✗
- JavaScript: ✓
- Python: ✓ (with functools)
- $\lambda$: ✓
- System F: ✓
- OCaml: ✓
- Haskell: ✓
**Eager Evaluation**

**Definition (Eager Semantics or Call-by-value):** An *eager* or *call-by-value* language evaluates all function arguments before passing them as parameters.

Operational semantics look like this:

\[
\forall i \in [1, n], \langle e_i, \sigma \rangle \Downarrow u_i \quad \sigma(f)(u_1, \ldots, u_n) \Downarrow u'
\]

\[
\langle f(e_1, \ldots, e_n), \sigma \rangle \Downarrow u'
\]
Lazy Evaluation

Definition (Lazy Semantics): A lazy language evaluates function arguments after the function body has started evaluating. There are two main varieties:

- **Call-by-name** languages (re)evaluate each argument expression each time the function uses it.
  - Can be formalized via capture-avoiding substitution
  - Disadvantage: usually inefficient
  - Advantage: sometimes highly efficient (e.g., unused arguments, highly parallelizable languages)

- **Call-by-need** languages evaluate each argument at first use, then memoize and reuse those values at subsequent uses.
  - Advantage: highest efficiency (usually)
  - Disadvantage: sometimes unintuitive!

Optional Exercises: Devise call-by-value and call-by-need operational semantics for $\lambda$-calculus
**Definition (call-by-reference):** Languages supporting *call-by-reference* allow callees to destructively modify the values of variables passed as arguments.

Example: Most object-oriented languages pass objects by reference, allowing callees to globally modify the object’s fields instead of receiving a local copy of the object.

Note: Call-by-reference does not make sense for immutable variables.
Which evaluation strategies are supported by the following languages?

- C
- C++
- SIMPL
- Java
- JavaScript
- Python
- $\lambda \rightarrow$
- System F
- OCaml
- Haskell
Which evaluation strategies are supported by the following languages?

C                call-by-value
C++              call-by-value, call-by-reference
SIMPL            call-by-value
Java             call-by-value, call-by-reference (objects)
JavaScript       call-by-value, call-by-reference (objects)
Python           call-by-value, call-by-reference (everything mutable!)
λ →              call-by-name
System F         call-by-name
OCaml            call-by-value, call-by-reference (objects)
Haskell          call-by-need
Which evaluation strategies are supported by the following languages?

- C: call-by-value
- C++: call-by-value, call-by-reference
- SIMPL: call-by-value
- Java: call-by-reference
- JavaScript: call-by-value, call-by-reference
- Python: call-by-value, call-by-reference
- λ →: call-by-name
- System F: call-by-need
- OCaml: call-by-value, call-by-reference
- Haskell: call-by-need
Which evaluation strategies are supported by the following languages?

- C: call-by-value
- C++: call-by-value, call-by-reference
- SIMPL: call-by-value
- Java
- JavaScript
- Python
- λ →
- System F
- OCaml
- Haskell
Which evaluation strategies are supported by the following languages?

- C call-by-value
- C++ call-by-value, call-by-reference
- SIMPL call-by-value
- Java call-by-value, call-by-reference (objects)
- JavaScript call-by-value, call-by-reference (objects)
- Python call-by-value, call-by-reference (everything mutable!)
- $\lambda \mapsto$ call-by-name
- System F
- OCaml
- Haskell
Which evaluation strategies are supported by the following languages?

<table>
<thead>
<tr>
<th>Language</th>
<th>Strategy</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>call-by-value</td>
</tr>
<tr>
<td>C++</td>
<td>call-by-value, call-by-reference</td>
</tr>
<tr>
<td>SIMPL</td>
<td>call-by-value</td>
</tr>
<tr>
<td>Java</td>
<td>call-by-value, call-by-reference (objects)</td>
</tr>
<tr>
<td>JavaScript</td>
<td>call-by-value, call-by-reference (objects)</td>
</tr>
<tr>
<td>Python</td>
<td>call-by-value</td>
</tr>
<tr>
<td>λ →</td>
<td></td>
</tr>
<tr>
<td>System F</td>
<td></td>
</tr>
<tr>
<td>OCaml</td>
<td></td>
</tr>
<tr>
<td>Haskell</td>
<td></td>
</tr>
</tbody>
</table>
Which evaluation strategies are supported by the following languages?

- C: call-by-value
- C++: call-by-value, call-by-reference
- SIMPL: call-by-value
- Java: call-by-value, call-by-reference (objects)
- JavaScript: call-by-value, call-by-reference (objects)
- Python: call-by-value, call-by-reference (everything mutable!)
- λ→: call-by-name
- System F
- OCaml
- Haskell
Which evaluation strategies are supported by the following languages?

<table>
<thead>
<tr>
<th>Language</th>
<th>Evaluation Strategy</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>call-by-value</td>
</tr>
<tr>
<td>C++</td>
<td>call-by-value, call-by-reference</td>
</tr>
<tr>
<td>SIMPL</td>
<td>call-by-value</td>
</tr>
<tr>
<td>Java</td>
<td>call-by-value, call-by-reference (objects)</td>
</tr>
<tr>
<td>JavaScript</td>
<td>call-by-value, call-by-reference (objects)</td>
</tr>
<tr>
<td>Python</td>
<td>call-by-value, call-by-reference (everything mutable!)</td>
</tr>
<tr>
<td>λ →</td>
<td>call-by-name</td>
</tr>
<tr>
<td>System F</td>
<td></td>
</tr>
<tr>
<td>OCaml</td>
<td></td>
</tr>
<tr>
<td>Haskell</td>
<td></td>
</tr>
</tbody>
</table>
Which evaluation strategies are supported by the following languages?

<table>
<thead>
<tr>
<th>Language</th>
<th>Evaluation Strategy</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>call-by-value</td>
</tr>
<tr>
<td>C++</td>
<td>call-by-value, call-by-reference</td>
</tr>
<tr>
<td>SIMPL</td>
<td>call-by-value</td>
</tr>
<tr>
<td>Java</td>
<td>call-by-value, call-by-reference (objects)</td>
</tr>
<tr>
<td>JavaScript</td>
<td>call-by-value, call-by-reference (objects)</td>
</tr>
<tr>
<td>Python</td>
<td>call-by-value, call-by-reference (everything mutable!)</td>
</tr>
<tr>
<td>λ →</td>
<td>call-by-name</td>
</tr>
<tr>
<td>System F</td>
<td>call-by-name</td>
</tr>
<tr>
<td>OCaml</td>
<td>call-by-name</td>
</tr>
<tr>
<td>Haskell</td>
<td></td>
</tr>
</tbody>
</table>
Which evaluation strategies are supported by the following languages?

- C: call-by-value
- C++: call-by-value, call-by-reference
- SIMPL: call-by-value
- Java: call-by-value, call-by-reference (objects)
- JavaScript: call-by-value, call-by-reference (objects)
- Python: call-by-value, call-by-reference (everything mutable!)
- λ→: call-by-name
- System F: call-by-name
- OCaml: call-by-value, call-by-reference (objects)
- Haskell: call-by-value, call-by-reference (objects)
Which evaluation strategies are supported by the following languages?

<table>
<thead>
<tr>
<th>Language</th>
<th>Evaluation Strategies</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>call-by-value</td>
</tr>
<tr>
<td>C++</td>
<td>call-by-value, call-by-reference</td>
</tr>
<tr>
<td>SIMPL</td>
<td>call-by-value</td>
</tr>
<tr>
<td>Java</td>
<td>call-by-value, call-by-reference (objects)</td>
</tr>
<tr>
<td>JavaScript</td>
<td>call-by-value, call-by-reference (objects)</td>
</tr>
<tr>
<td>Python</td>
<td>call-by-value, call-by-reference (everything mutable!)</td>
</tr>
<tr>
<td>λ →</td>
<td>call-by-name</td>
</tr>
<tr>
<td>System F</td>
<td>call-by-name</td>
</tr>
<tr>
<td>OCaml</td>
<td>call-by-value, call-by-reference (objects)</td>
</tr>
<tr>
<td>Haskell</td>
<td>call-by-need</td>
</tr>
</tbody>
</table>
Definition (Church-Rosser): Languages with the *Church-Rosser Property* are those in which the order of evaluation has no impact on the observable result. More technically, they are those languages whose small-step operational semantics are *confluent*.

Church-Rosser languages typically...

- have strictly immutable variables,
- are *pure* (i.e., free of side-effects).

Languages that are Church-Rosser can have unknown evaluation strategies (unobservable to the user), and offer compilers many optimization opportunities.
Definition (**static/dynamic typing**): A language is *(strictly) statically typed* if all types are erased during compilation. In contrast, a language is *dynamically typed* if types are available at runtime (usually attached to runtime values).

Advantages of strict static typing:
- space- and time-efficiency (no runtime storage or tracking of types)
- types facilitate static debugging
- types facilitate compile-time static code optimization
- types can be more universal (e.g., characterizing all possible executions)

Advantages of dynamic typing:
- type-tag values available at runtime (whether you need them or not)
- sometimes easier patching and bug mitigation
- opportunities for extra security sanity-checking
Strict Static Typing

Which of the following languages are strictly statically typed?

- C
- C++
- SIMPL
- Java
- JavaScript
- Python
- λ →
- System F
- OCaml
- Haskell
Which of the following languages are strictly statically typed?

- C
- C++
- SIMPL
- Java
- JavaScript
- Python
- λ→
- System F
- OCaml
- Haskell
Strict Static Typing

Which of the following languages are strictly statically typed?

- C ✓
- C++ ✓
- SIMPL ✓
- Java
- JavaScript
- Python
- \( \lambda \rightarrow \)
- System F
- OCaml
- Haskell
Which of the following languages are strictly statically typed?

- C  ✔
- C++  ✔
- SIMPL  ✔
- Java
- JavaScript
- Python
- $\lambda\rightarrow$
- System F
- OCaml
- Haskell
Which of the following languages are strictly statically typed?

- C ✔
- C++ ✔
- SIMPL ✔
- Java ✗
- JavaScript
- Python
- λ→
- System F
- OCaml
- Haskell
Which of the following languages are strictly statically typed?

- C  ✓
- C++  ✓
- SIMPL  ✓
- Java  ✗
- JavaScript  ✗
- Python
- λ→
- System F
- OCaml
- Haskell
Which of the following languages are strictly statically typed?

- C ✓
- C++ ✓
- SIMPL ✓
- Java ✗
- JavaScript ✗
- Python ✗
- $\lambda \rightarrow$
- System F
- OCaml
- Haskell
### Strict Static Typing

Which of the following languages are strictly statically typed?

<table>
<thead>
<tr>
<th>Language</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>✓</td>
</tr>
<tr>
<td>C++</td>
<td>✓</td>
</tr>
<tr>
<td>SIMPL</td>
<td>✓</td>
</tr>
<tr>
<td>Java</td>
<td>✗</td>
</tr>
<tr>
<td>JavaScript</td>
<td>✗</td>
</tr>
<tr>
<td>Python</td>
<td>✗</td>
</tr>
<tr>
<td>System F</td>
<td>✓</td>
</tr>
<tr>
<td>OCaml</td>
<td>✓</td>
</tr>
<tr>
<td>Haskell</td>
<td>✓</td>
</tr>
</tbody>
</table>
Which of the following languages are strictly statically typed?

C
C++
SIMPL
Java
JavaScript
Python
λ→
System F
OCaml
Haskell
Strict Static Typing

Which of the following languages are strictly statically typed?

- C ✓
- C++ ✓
- SIMPL ✓
- Java ✗
- JavaScript ✗
- Python ✗
- λ → ✓
- System F ✓
- OCaml ✓ (except for objects)
- Haskell ✓

Advanced Programming Languages
Typing Features
Which of the following languages are strictly statically typed?

- C ✓
- C++ ✓
- SIMPL ✓
- Java ✗
- JavaScript ✗
- Python ✗
- $\lambda \to$ ✓
- System F ✓
- OCaml ✓ (except for objects)
- Haskell ✓
**Definition (type-safety):** A language is *type-safe* if its static semantics preclude all stuck states in its operational semantics.

Sometimes difficult to tell whether a language is type-safe because:

- Some languages have no formal semantics(?!?!).
- Some languages have an operational semantics that formalizes states most of us would consider stuck states.

Which of the following languages are type-safe?

- C
- C++
- SIMPL
- Java
- JavaScript
- Python
- λ→
- System F
- OCaml
- Haskell
Definition (type-safety): A language is type-safe if its static semantics preclude all stuck states in its operational semantics.

Sometimes difficult to tell whether a language is type-safe because:

- Some languages have no formal semantics.?!?.
- Some languages have an operational semantics that formalizes states most of us would consider stuck states.

Which of the following languages are type-safe?

C  
C++
SIMPL
Java
JavaScript
Python
λ→
System F
OCaml
Haskell
Definition (type-safety): A language is *type-safe* if its static semantics preclude all stuck states in its operational semantics.

Sometimes difficult to tell whether a language is type-safe because:

- Some languages have no formal semantics (?!?!).
- Some languages have an operational semantics that formalizes states most of us would consider stuck states.

Which of the following languages are type-safe?

- C  
- C++  
- SIMPL  
- Java  
- JavaScript  
- Python  
- $\lambda$  
- System F  
- OCaml  
- Haskell
Definition (type-safety): A language is type-safe if its static semantics preclude all stuck states in its operational semantics.

Sometimes difficult to tell whether a language is type-safe because:

- Some languages have no formal semantics. (!?(!?!
- Some languages have an operational semantics that formalizes states most of us would consider stuck states.

Which of the following languages are type-safe?

- C
- C++
- SIMPL   
- Java
- JavaScript
- Python
- λ
- System F
- OCaml
- Haskell
**Type-safety**

**Definition (type-safety):** A language is *type-safe* if its static semantics preclude all stuck states in its operational semantics.

Sometimes difficult to tell whether a language is type-safe because:

- Some languages have no formal semantics (?!?!).
- Some languages have an operational semantics that formalizes states most of us would consider stuck states.

Which of the following languages are type-safe?

C     ✗
C++   ✗
SIMPL ✓
Java  ✌️ (stuckness formalized as exception)
JavaScript
Python
λ →
System F
OCaml
Haskell
Definition (type-safety): A language is type-safe if its static semantics preclude all stuck states in its operational semantics.

Sometimes difficult to tell whether a language is type-safe because:

- Some languages have no formal semantics (?!?!).
- Some languages have an operational semantics that formalizes states most of us would consider stuck states.

Which of the following languages are type-safe?

- C
- C++
- SIMPL
- Java
- JavaScript
- Python
- λ
- System F
- OCaml
- Haskell

C
C++
SIMPL

Java
JavaScript

Python

λ
System F
OCaml
Haskell
Definition (type-safety): A language is type-safe if its static semantics preclude all stuck states in its operational semantics.

Sometimes difficult to tell whether a language is type-safe because:

- Some languages have no formal semantics.
- Some languages have an operational semantics that formalizes states most of us would consider stuck states.

Which of the following languages are type-safe?

C  
C++  
SIMPL  
Java  (stuckness formalized as exception)  
JavaScript  (stuckness formalized as exception)  
Python  (stuckness formalized as exception)  
λ→ System F  
OCaml  
Haskell
**Type-safety**

**Definition (type-safety):** A language is *type-safe* if its static semantics preclude all stuck states in its operational semantics.

Sometimes difficult to tell whether a language is type-safe because:

- Some languages have no formal semantics (?!?!).
- Some languages have an operational semantics that formalizes states most of us would consider stuck states.

Which of the following languages are type-safe?

<table>
<thead>
<tr>
<th>Language</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>✗</td>
</tr>
<tr>
<td>C++</td>
<td>✗</td>
</tr>
<tr>
<td>SIMPL</td>
<td>✓</td>
</tr>
<tr>
<td>Java</td>
<td>☺️ (stuckness formalized as exception)</td>
</tr>
<tr>
<td>JavaScript</td>
<td>☺️ (stuckness formalized as exception)</td>
</tr>
<tr>
<td>Python</td>
<td>☺️ (stuckness formalized as exception)</td>
</tr>
<tr>
<td>λ→</td>
<td>✓</td>
</tr>
<tr>
<td>System F</td>
<td></td>
</tr>
<tr>
<td>OCaml</td>
<td></td>
</tr>
<tr>
<td>Haskell</td>
<td></td>
</tr>
</tbody>
</table>
Type-safety

Definition (type-safety): A language is type-safe if its static semantics preclude all stuck states in its operational semantics.

Sometimes difficult to tell whether a language is type-safe because:

- Some languages have no formal semantics.
- Some languages have an operational semantics that formalizes states most of us would consider stuck states.

Which of the following languages are type-safe?

- C:  
- C++:  
- SIMPL: ✓ 
- Java:  \ (stuckness formalized as exception) 
- JavaScript:  \ (stuckness formalized as exception) 
- Python:  \ (stuckness formalized as exception) 
- λ: ✓ 
- System F: ✓ 
- OCaml: ✓ 
- Haskell: ✓
Type-safety

**Definition (type-safety):** A language is *type-safe* if its static semantics preclude all stuck states in its operational semantics.

Sometimes difficult to tell whether a language is type-safe because:

- Some languages have no formal semantics(?!?!).
- Some languages have an operational semantics that formalizes states most of us would consider stuck states.

Which of the following languages are type-safe?

- C  
  - ![Cross](representing-exclusion)
- C++  
  - ![Cross](representing-exclusion)
- SIMPL  
  - ![Check](representing-acceptance)
- Java  
  - ![Smiley](representing-acceptance) (stuckness formalized as exception)
- JavaScript  
  - ![Smiley](representing-acceptance) (stuckness formalized as exception)
- Python  
  - ![Smiley](representing-acceptance) (stuckness formalized as exception)
- λ  
  - ![Check](representing-acceptance)
- System F  
  - ![Check](representing-acceptance)
- OCaml  
  - ![Check](representing-acceptance)
- Haskell  
  - ![Check](representing-acceptance)
**Type-safety**

**Definition (type-safety):** A language is *type-safe* if its static semantics preclude all stuck states in its operational semantics.

Sometimes difficult to tell whether a language is type-safe because:

- Some languages have no formal semantics.
- Some languages have an operational semantics that formalizes states most of us would consider stuck states.

Which of the following languages are type-safe?

<table>
<thead>
<tr>
<th>Language</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>✗</td>
</tr>
<tr>
<td>C++</td>
<td>✗</td>
</tr>
<tr>
<td>SIMPL</td>
<td>✓</td>
</tr>
<tr>
<td>Java</td>
<td>✗</td>
</tr>
<tr>
<td>JavaScript</td>
<td>✗</td>
</tr>
<tr>
<td>Python</td>
<td>✗</td>
</tr>
<tr>
<td>$\lambda$</td>
<td>✓</td>
</tr>
<tr>
<td>System F</td>
<td>✓</td>
</tr>
<tr>
<td>OCaml</td>
<td>✓</td>
</tr>
<tr>
<td>Haskell</td>
<td>✓</td>
</tr>
</tbody>
</table>
**Polymorphism**

**Definition (polymorphism):** A language is *polymorphic* if interfaces (e.g., functions) can accommodate entities (e.g., arguments) of multiple different types.

Three main varieties:

1. **Parametric Polymorphism:** type system has type-variables $\alpha$
   - facilitates machine-checked code-reuse idioms
   - compatible with strictly static type-safety

2. **Subtyping Polymorphism:** object types arranged in a hierarchy
   - hallmark of object-oriented programming
   - static semantics usually characterized by a weakening rule:
     \[
     \frac{\Gamma \vdash e : \tau \quad \tau \leq \tau'}{\Gamma \vdash e : \tau'}
     \]
   - Warning: makes structural induction proofs much harder (Why?)

3. **Ad hoc Polymorphism:** conditionals can test types at runtime
   - opens the door for arbitrarily heterogeneous code blocks per type
   - antithesis of code-reuse (much harder to maintain and debug)
Which forms of polymorphism are supported by the following languages?

- C
- C++
- SIMPL
- Java
- JavaScript
- Python
- λ
- System F
- OCaml
- Haskell
Which forms of polymorphism are supported by the following languages?

- C: none
- C++: subtyping
- SIMPL: none
- Java: parametric (generics), subtyping, ad hoc
- JavaScript: subtyping, ad hoc
- Python: parametric (generics), subtyping, ad hoc
- λ→: none
- System F: parametric
- OCaml: parametric, subtyping
- Haskell: parametric
Polymorphism Examples

Which forms of polymorphism are supported by the following languages?

- C: none
- C++: subtyping
- SIMPL: none
- Java: parametric (generics), subtyping, ad hoc
- JavaScript: subtyping, ad hoc
- Python: parametric (generics), subtyping, ad hoc
- $\lambda \rightarrow$: none
- System F: parametric
- OCaml: parametric, subtyping
- Haskell: parametric
Which forms of polymorphism are supported by the following languages?

<table>
<thead>
<tr>
<th>Language</th>
<th>Polymorphism</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>none</td>
</tr>
<tr>
<td>C++</td>
<td>subtyping</td>
</tr>
<tr>
<td>SIMPL</td>
<td>none</td>
</tr>
<tr>
<td>Java</td>
<td>none</td>
</tr>
<tr>
<td>JavaScript</td>
<td>none</td>
</tr>
<tr>
<td>Python</td>
<td>subtyping, ad hoc</td>
</tr>
<tr>
<td>λ</td>
<td>parametric (generics)</td>
</tr>
<tr>
<td>System F</td>
<td>parametric</td>
</tr>
<tr>
<td>OCaml</td>
<td>parametric, subtyping</td>
</tr>
<tr>
<td>Haskell</td>
<td>parametric</td>
</tr>
</tbody>
</table>

Polymorphism Examples
Which forms of polymorphism are supported by the following languages?

<table>
<thead>
<tr>
<th>Language</th>
<th>Polymorphism Forms Supported</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>none</td>
</tr>
<tr>
<td>C++</td>
<td>subtyping</td>
</tr>
<tr>
<td>SIMPL</td>
<td>none</td>
</tr>
<tr>
<td>Java</td>
<td>parametric (generics), subtyping, ad hoc</td>
</tr>
<tr>
<td>JavaScript</td>
<td>parametric (generics), subtyping, ad hoc</td>
</tr>
<tr>
<td>Python</td>
<td>parametric (generics), subtyping, ad hoc</td>
</tr>
<tr>
<td>λ →</td>
<td>none</td>
</tr>
<tr>
<td>System F</td>
<td>parametric (generics), subtyping, ad hoc</td>
</tr>
<tr>
<td>OCaml</td>
<td>parametric (generics), subtyping, ad hoc</td>
</tr>
<tr>
<td>Haskell</td>
<td>parametric (generics), subtyping, ad hoc</td>
</tr>
</tbody>
</table>
Polymorphism Examples

Which forms of polymorphism are supported by the following languages?

<table>
<thead>
<tr>
<th>Language</th>
<th>Forms of Polymorphism</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>none</td>
</tr>
<tr>
<td>C++</td>
<td>subtyping</td>
</tr>
<tr>
<td>SIMPL</td>
<td>none</td>
</tr>
<tr>
<td>Java</td>
<td>parametric (generics), subtyping, ad hoc</td>
</tr>
<tr>
<td>JavaScript</td>
<td>subtyping, ad hoc</td>
</tr>
<tr>
<td>Python</td>
<td>subtyping, ad hoc</td>
</tr>
<tr>
<td>λ            →</td>
<td>none</td>
</tr>
<tr>
<td>System F</td>
<td>parametric</td>
</tr>
<tr>
<td>OCaml</td>
<td>subtyping, ad hoc</td>
</tr>
<tr>
<td>Haskell</td>
<td>parametric</td>
</tr>
</tbody>
</table>
Which forms of polymorphism are supported by the following languages?

- C: none
- C++: subtyping
- SIMPL: none
- Java: parametric (generics), subtyping, ad hoc
- JavaScript: subtyping, ad hoc
- Python: parametric (generics), subtyping, ad hoc
- System F: parametric
- OCaml: parametric, subtyping
- Haskell: parametric
## Polymorphism Examples

Which forms of polymorphism are supported by the following languages?

<table>
<thead>
<tr>
<th>Language</th>
<th>Forms of Polymorphism</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>none</td>
</tr>
<tr>
<td>C++</td>
<td>subtyping</td>
</tr>
<tr>
<td>SIMPL</td>
<td>none</td>
</tr>
<tr>
<td>Java</td>
<td>parametric (generics), subtyping, ad hoc</td>
</tr>
<tr>
<td>JavaScript</td>
<td>subtyping, ad hoc</td>
</tr>
<tr>
<td>Python</td>
<td>parametric (generics), subtyping, ad hoc</td>
</tr>
<tr>
<td>λ →</td>
<td>none</td>
</tr>
<tr>
<td>System F</td>
<td></td>
</tr>
<tr>
<td>OCaml</td>
<td></td>
</tr>
<tr>
<td>Haskell</td>
<td></td>
</tr>
</tbody>
</table>
### Polymorphism Examples

Which forms of polymorphism are supported by the following languages?

<table>
<thead>
<tr>
<th>Language</th>
<th>Forms of Polymorphism</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>none</td>
</tr>
<tr>
<td>C++</td>
<td>subtyping</td>
</tr>
<tr>
<td>SIMPL</td>
<td>none</td>
</tr>
<tr>
<td>Java</td>
<td>parametric (generics), subtyping, ad hoc</td>
</tr>
<tr>
<td>JavaScript</td>
<td>subtyping, ad hoc</td>
</tr>
<tr>
<td>Python</td>
<td>parametric (generics), subtyping, ad hoc</td>
</tr>
<tr>
<td>λ→</td>
<td>none</td>
</tr>
<tr>
<td>System F</td>
<td>parametric</td>
</tr>
<tr>
<td>OCaml</td>
<td>parametric</td>
</tr>
<tr>
<td>Haskell</td>
<td>parametric</td>
</tr>
</tbody>
</table>
Polymorphism Examples

Which forms of polymorphism are supported by the following languages?

- C: none
- C++: subtyping
- SIMPL: none
- Java: parametric (generics), subtyping, ad hoc
- JavaScript: subtyping, ad hoc
- Python: parametric (generics), subtyping, ad hoc
- \( \lambda \rightarrow \): none
- System F: parametric
- OCaml: parametric, subtyping
- Haskell: parametric, subtyping
Which forms of polymorphism are supported by the following languages?

- C: none
- C++: subtyping
- SIMPL: none
- Java: parametric (generics), subtyping, ad hoc
- JavaScript: subtyping, ad hoc
- Python: parametric (generics), subtyping, ad hoc
- λ→: none
- System F: parametric
- OCaml: parametric, subtyping
- Haskell: parametric
Non-shallow Types

**Definition (shallow types):** A shallowly-typed language is one whose type system only supports type quantifiers at the top level of types (not nested within non-quantifiers).

Which of the following languages support non-shallow types:

- C
- C++
- SIMPL
- Java
- JavaScript
- Python
- $\lambda \rightarrow$
- System F
- OCaml
- Haskell
Non-shallow Types

**Definition (shallow types):** A shallowly-typed language is one whose type system only supports type quantifiers at the top level of types (not nested within non-quantifiers).

Which of the following languages support non-shallow types:

- C
- C++
- SIMPL
- Java
- JavaScript
- Python
- \(\lambda\rightarrow\)
- System F
- OCaml
- Haskell
Non-shallow Types

**Definition (shallow types):** A shallowly-typed language is one whose type system only supports type quantifiers at the top level of types (not nested within non-quantifiers).

Which of the following languages support non-shallow types:

- C
- C++
- SIMPL
- Java
- JavaScript
- Python
- $\lambda \rightarrow$
- System F
- OCaml
- Haskell

*Note: Crosses indicate languages that do not support non-shallow types.*
**Definition (shallow types):** A shallowly-typed language is one whose type system only supports type quantifiers at the top level of types (not nested within non-quantifiers).

Which of the following languages support non-shallow types:

- C
- C++
- SIMPL
- Java
- JavaScript
- Python
- $\lambda \rightarrow$
- System F
- OCaml
- Haskell
Definition (shallow types): A shallowly-typed language is one whose type system only supports type quantifiers at the top level of types (not nested within non-quantifiers).

Which of the following languages support non-shallow types:

- C
- C++
- SIMPL
- Java
- JavaScript
- Python
- λ→
- System F
- OCaml
- Haskell

X
Non-shallow Types

Definition (shallow types): A shallowly-typed language is one whose type system only supports type quantifiers at the top level of types (not nested within non-quantifiers).

Which of the following languages support non-shallow types:

- C
- C++
- SIMPL
- Java
- JavaScript
- Python
- $\lambda \rightarrow$
- System F
- OCaml
- Haskell
Non-shallow Types

**Definition (shallow types):** A shallowly-typed language is one whose type system only supports type quantifiers at the top level of types (not nested within non-quantifiers).

Which of the following languages support non-shallow types:

- C  
- C++  
- SIMPL  
- Java  
- JavaScript  
- Python  
- \(\lambda\rightarrow\)  
- System F  
- OCaml  
- Haskell
Definition (shallow types): A shallowly-typed language is one whose type system only supports type quantifiers at the top level of types (not nested within non-quantifiers).

Which of the following languages support non-shallow types:

- C
- C++
- SIMPL
- Java
- JavaScript
- Python
- λ→
- System F
- OCaml
- Haskell
Definition (shallow types): A shallowly-typed language is one whose type system only supports type quantifiers at the top level of types (not nested within non-quantifiers).

Which of the following languages support non-shallow types:

- C
- C++
- SIMPL
- Java
- JavaScript
- Python
- $\lambda \rightarrow$
- System F
- OCaml
- Haskell
Non-shallow Types

Definition (shallow types): A shallowly-typed language is one whose type system only supports type quantifiers at the top level of types (not nested within non-quantifiers).

Which of the following languages support non-shallow types:

- C  
- C++  
- SIMPL  
- Java  
- JavaScript  
- Python  
- $\lambda\to$  
- System F  
- OCaml (with --rectypes)  
- Haskell
Non-shallow Types

**Definition (shallow types):** A shallowly-typed language is one whose type system only supports type quantifiers at the top level of types (not nested within non-quantifiers).

Which of the following languages support non-shallow types:

- C
- C++
- SIMPL
- Java
- JavaScript
- Python
- \(\lambda\)
- System F
- OCaml (with --rectypes)
- Haskell
## Summary Table

<table>
<thead>
<tr>
<th>Language</th>
<th>first-class functions</th>
<th>currying &amp; partial evaluation</th>
<th>evaluation strategies</th>
<th>strict static typing</th>
<th>type safety</th>
<th>polymorphism</th>
<th>non-shallow types</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>✗</td>
<td>✗</td>
<td>√</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>C++</td>
<td>✗</td>
<td>✗</td>
<td>√</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>SIMPL</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Java</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>JavaScript</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>Python</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>λ→</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>☹</td>
</tr>
<tr>
<td>System F</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>OCaml</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Haskell</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>