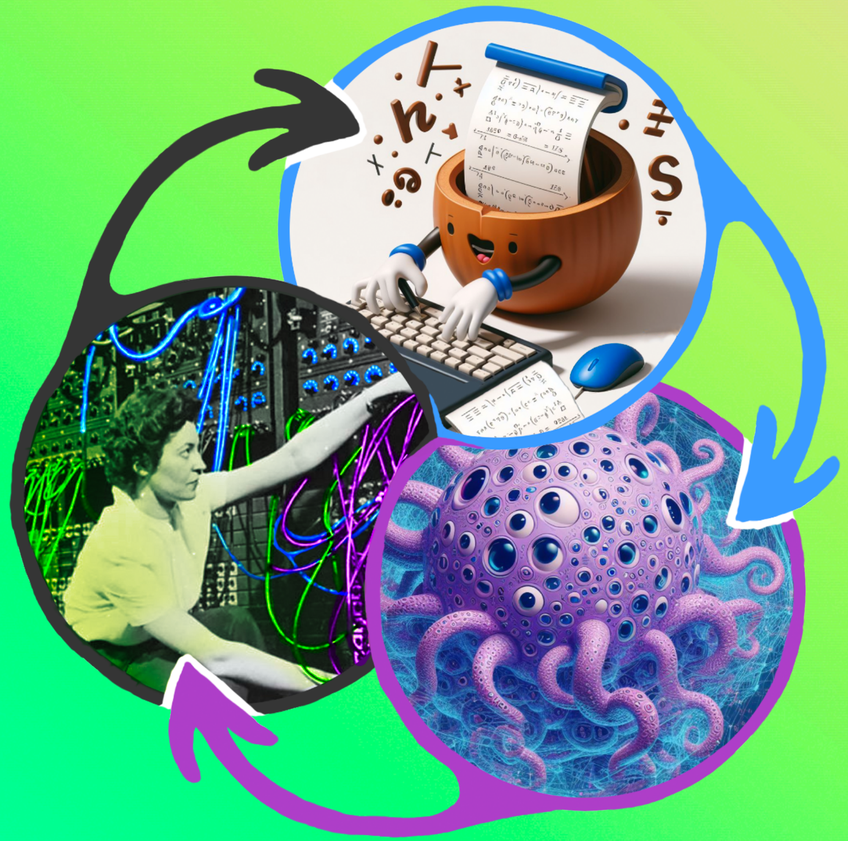
OOPSLA24

Statically Contextualizing Large Language Models with Typed Holes



Andrew Blinn · Xiang (Kevin) Li · June Hyung (Jacob) Kim · Cyrus Omar

Future of Programming Lab
University of Michigan

Problem:

LLMs often produce broken code, sometimes due to lack of appropriate context. How can we best proactively provide context to inform code completions?

Problem:

LLMs often produce broken code, sometimes due to lack of appropriate context.
How can we best proactively provide context to inform code completions?

Our Approach:

A conversation between **programmer**, **language server**, and **language model,** using typed holes to bridge **cursor-local** and **repository-wide** semantic information

Problem:

LLMs often produce broken code, sometimes due to lack of appropriate context.
How can we best proactively provide context to inform code completions?

Our Approach:

A conversation between **programmer**, **language server**, and **language model,** using typed holes to bridge **cursor-local** and **repository-wide** semantic information

Our Evaluation:

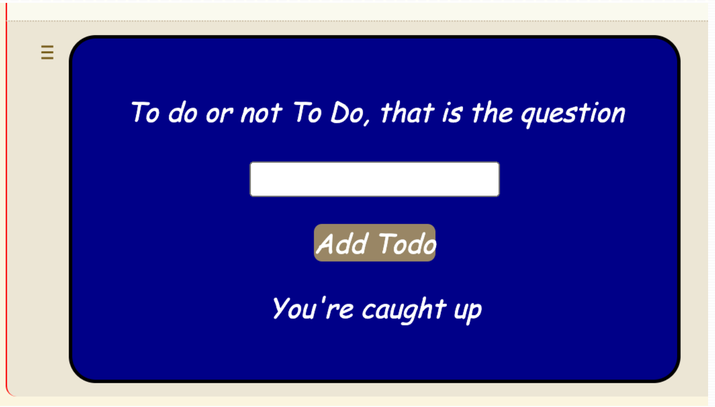
We implemented and evaluated our method in Hazel, our lab's typed functional programming language and live program sketching environment

We (partially) reproduced our method and results in TypeScript

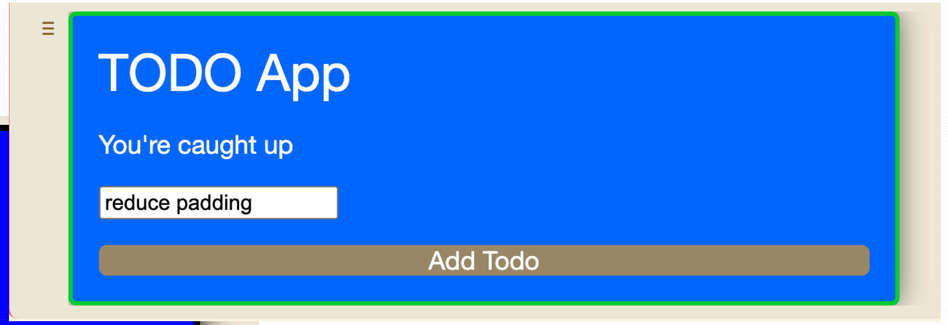
DEMO

DEMO


Add Todo
You're caught up
Let'sDo it!


To do or not To Do, that is the question
Add Todo
You're caught up


ToDo Zoo
Add Todo
todos:
yo Pending


TODO App
You're caught up
reduce padding
Add Todo

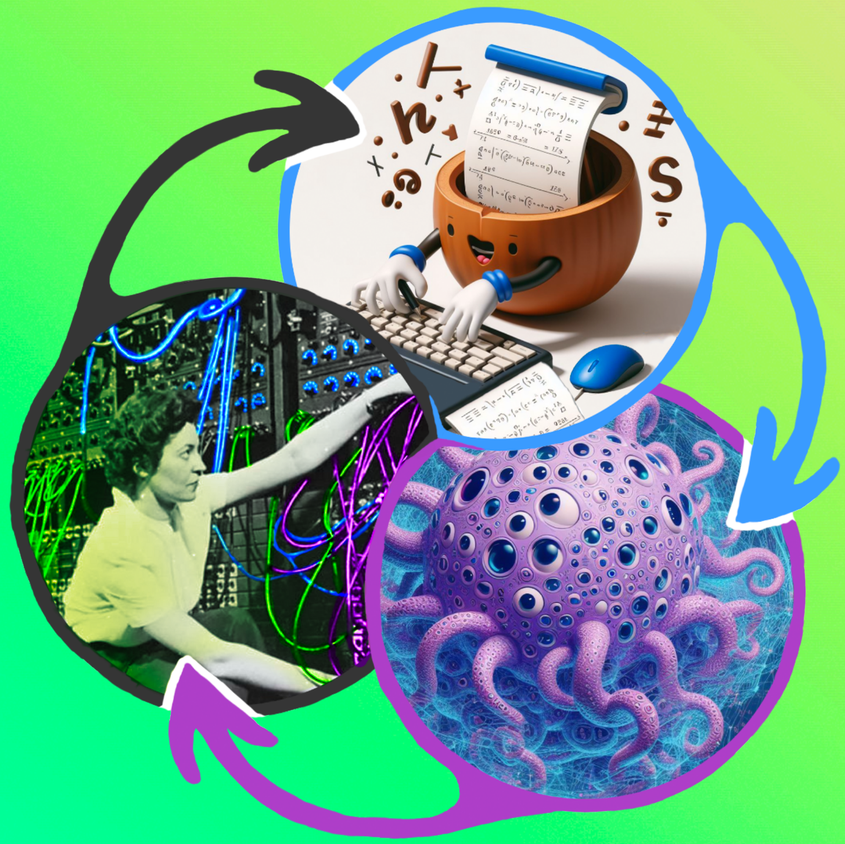

Todo App
Add Todo
You're caught up


TODO List
Add Todo
You're caught up


Todo App
Add Todo
You're caught up

MOTIVATION

Contextualizing Context



Complete this program sketch

```
# Update the EmojiPainter app #

let update: (Model, Action) -> Model =
  ??
in
```

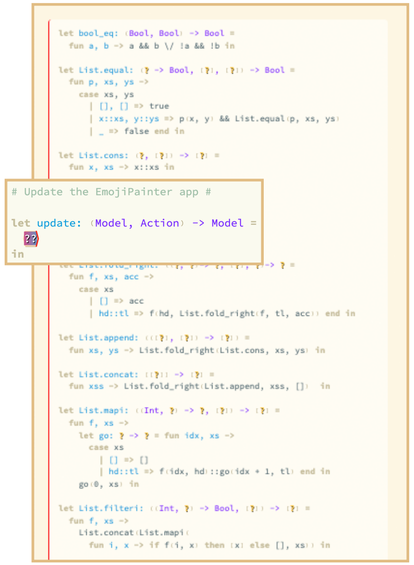
Prompting an LLM is like lightning onboarding

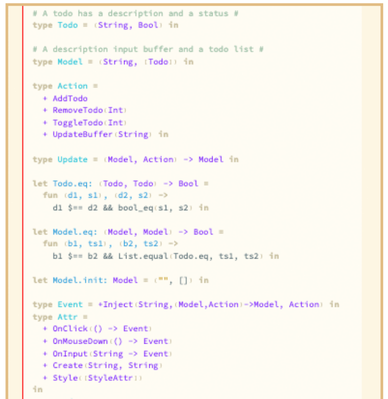
```
# Update the EmojiPainter app #

let update: (Model, Action) -> Model =
  ??
in
```

Contextualization

Strategies

- Text window
- Vector Retrieval
- Repository-level

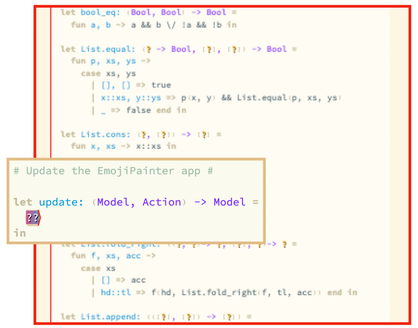




Contextualization

Strategies

- **Text window**
- Vector Retrieval
- Repository-level

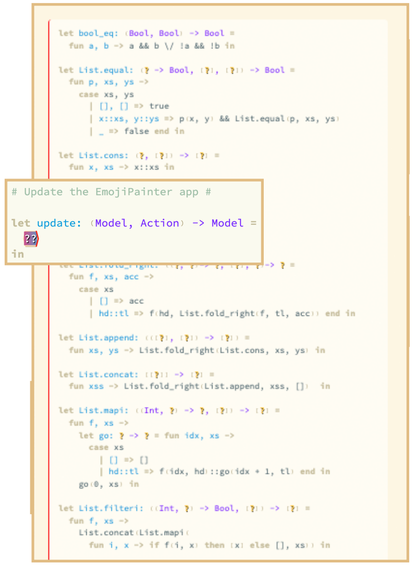


Contextualization

Strategies

- Text window
- **Vector Retrieval**
- Repository-level

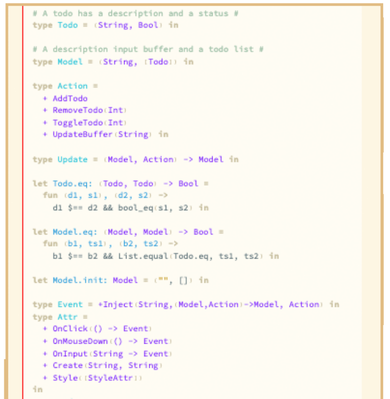
```
let todos_deck: (Action -> Event, (Todo)) -> Node =
  fun go, todos ->
    if (List.is_empty(todos))
    then Text("You're caught up")
    else Div(
      (Create("class", "todos"),
      (Text("todos:"))
      @ List.mapi(fun i,t ->todo_card(go,i,t), todos)) in


let add_button: (Action -> Event) -> Node =
  fun go -> Div((
    Style((
      Display:"flex",
      JustifyContent:"center",
      BackgroundColor:"#986",
      BorderRadius:"0.3em",
      Cursor:"pointer"})),
    OnClick(fun () -> go(AddTodo))),
    (Text("Add Todo"))) in

let buffer: (Action -> Event) -> Node =
  fun go -> Div(
    [],
    (TextInput((OnInput(fun s -> go(UpdateBuffer(s)))), []))) in
```



```
let bool_eq: (Bool, Bool) -> Bool =
  fun a, b -> a && b \/ !a && !b in

let List.equal: (? -> Bool, (?), (?)) -> Bool =
  fun p, xs, ys ->
    case xs, ys
      | [], [] => true
      | x::xs, y::ys => p(x, y) && List.equal(p, xs, ys)
      | _ => false end in

let List.cons: (?, (?)) -> (?) =
  fun x, xs -> x::xs in
```

```
# Update the EmojiPainter app #

let update: (Model, Action) -> Model =
  ??
in
```

```
  fun f, xs, acc ->
    case xs
      | [] => acc
      | hd::tl => f(hd, List.fold_right(f, tl, acc)) end in

let List.append: ((?), (?)) -> (?) =
  fun xs, ys -> List.fold_right(List.cons, xs, ys) in

let List.concat: ((?)) -> (?) =
  fun xss -> List.fold_right(List.append, xss, []) in

let List.mapi: ((Int, ?) -> ?, (?)) -> (?) =
  fun f, xs ->
    let go: ? -> ? = fun idx, xs ->
      case xs
        | [] => []
        | hd::tl => f(idx, hd::go(idx + 1, tl)) end in
    go(0, xs) in

let List.filteri: ((Int, ?) -> Bool, (?)) -> (?) =
  fun f, xs ->
    List.concat(List.mapi(
      fun i, x -> if f(i, x) then (x) else [], xs)) in
```



```
# A todo has a description and a status #
type Todo = (String, Bool) in

# A description input buffer and a todo list #
type Model = (String, (Todo)) in

type Action =
  + AddTodo
  + RemoveTodo(Int)
  + ToggleTodo(Int)
  + UpdateBuffer(String) in

type Update = (Model, Action) -> Model in

let Todo.eq: (Todo, Todo) -> Bool =
  fun (d1, s1), (d2, s2) ->
    d1 $== d2 && bool_eq(s1, s2) in

let Model.eq: (Model, Model) -> Bool =
  fun (b1, ts1), (b2, ts2) ->
    b1 $== b2 && List.equal(Todo.eq, ts1, ts2) in

let Model.init: Model = ("", []) in

type Event = +Inject(String,(Model,Action)->Model, Action) in
type Attr =
  + OnClick(() -> Event)
  + OnMouseDown(() -> Event)
  + OnInput(String -> Event)
  + Create(String, String)
  + Style(StyleAttr)
in
```

Contextualization

Strategies

- Text window
- **Vector Retrieval**
- Repository-level

Contextualization

Strategies

- Text window
- **Vector Retrieval**
- Repository-level

```
# Update the EmojiPainter app #

let update: (Model, Action) -> Model =
  ??
in
```

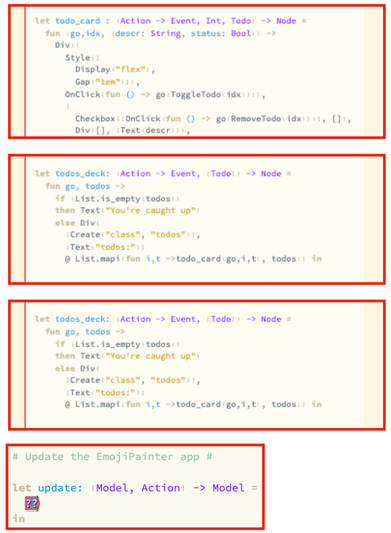
```
let todo_card : (Action -> Event, Int, Todo) -> Node =
  fun (go,idx, (descr: String, status: Bool)) ->
    Div(
      Style(
        Display("flex"),
        Gap("1em")),
      OnClick(fun () -> go(ToggleTodo(idx)))),
      [
        Checkbox(OnClick(fun () -> go(RemoveTodo(idx)))), [],
        Div([], [Text(descr)]),
```

```
let todos_deck: (Action -> Event, [Todo]) -> Node =
  fun go, todos ->
    if (List.is_empty(todos))
    then Text("You're caught up")
    else Div(
      [Create("class", "todos")],
      [Text("todos:")]
      @ List.map(fun i,t ->todo_card(go,i,t), todos)) in
```

Contextualization

Strategies

- Text window
- **Vector Retrieval**
- Repository-level



```
let todo_card : (Action -> Event, Int, Todo) -> Node =
  fun (go,idx, (descr: String, status: Bool)) ->
    Div(
      Style(
        Display("flex"),
        Gap("1em")),
      OnClick(fun () -> go(ToggleTodo(idx))),
      [
        Checkbox(OnClick(fun () -> go(RemoveTodo(idx)))), [],
        Div([], [Text(descr)]),
```

```
let todos_deck: (Action -> Event, [Todo]) -> Node =
  fun go, todos ->
    if (List.is_empty(todos))
    then Text("You're caught up")
    else Div(
      [Create("class", "todos")],
      [Text("todos:")]
      @ List.map(fun i,t ->todo_card(go,i,t), todos)) in
```

```
let todos_deck: (Action -> Event, [Todo]) -> Node =
  fun go, todos ->
    if (List.is_empty(todos))
    then Text("You're caught up")
    else Div(
      [Create("class", "todos")],
      [Text("todos:")]
      @ List.map(fun i,t ->todo_card(go,i,t), todos)) in
```

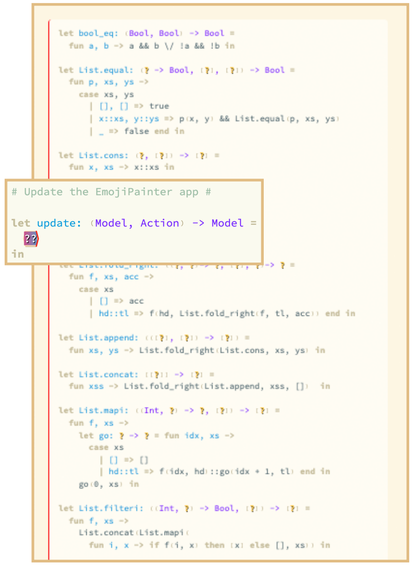
```
# Update the EmojiPainter app #

let update: (Model, Action) -> Model =
  ??
in
```

Contextualization

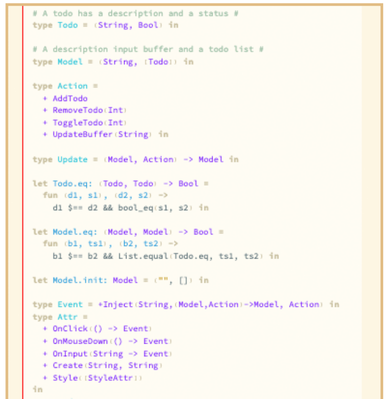
Strategies

- Text window
- Vector Retrieval
- **Repository-level**

```
fun (go,idx, (descr: String, status: Bool)) ->
  Div((
    Style((
      Display("flex"),
      Gap("1em")
    )),
    OnClick(fun () -> go(ToggleTodo(idx)))
  ),
  (
    Checkbox((OnClick(fun () -> go(RemoveTodo(idx)))), []),
    Div([], (Text(descr))),
    Text(if status then "Completed" else "Pending")
  )) in

let todos_deck: (Action -> Event, (Todo)) -> Node =
  fun go, todos ->
    if (List.is_empty(todos))
    then Text("You're caught up")
    else Div(
      (Create("class", "todos")),
      (Text("todos:"))
      @ List.mapi(fun i,t ->todo_card(go,i,t), todos)) in


let add_button: (Action -> Event) -> Node =
  fun go -> Div((
    Style((
      Display("flex"),
      JustifyContent("center"),
      BackgroundColor("#9886"),
      BorderRadius("0.3em"),
      Cursor("pointer")
    )),
    OnClick(fun () -> go(AddTodo))),
    (Text("Add Todo"))) in

let buffer: (Action -> Event) -> Node =
  fun go -> Div(
    [],
    (TextInput((OnInput(fun s -> go(UpdateBuffer(s)))), []))) in
```



```
let bool_eq: (Bool, Bool) -> Bool =
  fun a, b -> a && b \/ !a && !b in

let List.equal: (? -> Bool, (?), (?)) -> Bool =
  fun p, xs, ys ->
    case xs, ys
      | [], [] => true
      | x::xs, y::ys => p(x, y) && List.equal(p, xs, ys)
      | _ => false end in

let List.cons: (?, (?)) -> (?) =
  fun x, xs -> x::xs in

# Update the EmojiPainter app #

let update: (Model, Action) -> Model =
  ??
in

  fun f, xs, acc ->
    case xs
      | [] => acc
      | hd::tl => f(hd, List.fold_right(f, tl, acc)) end in

let List.append: ((?), (?)) -> (?) =
  fun xs, ys -> List.fold_right(List.cons, xs, ys) in

let List.concat: ((?)) -> (?) =
  fun xss -> List.fold_right(List.append, xss, []) in

let List.mapi: ((Int, ?) -> ?, (?)) -> (?) =
  fun f, xs ->
    let go: ? -> ? = fun idx, xs ->
      case xs
        | [] => []
        | hd::tl => f(idx, hd::go(idx + 1, tl)) end in
    go(0, xs) in

let List.filteri: ((Int, ?) -> Bool, (?)) -> (?) =
  fun f, xs ->
    List.concat(List.mapi(
      fun i, x -> if f(i, x) then (x) else [], xs)) in
```

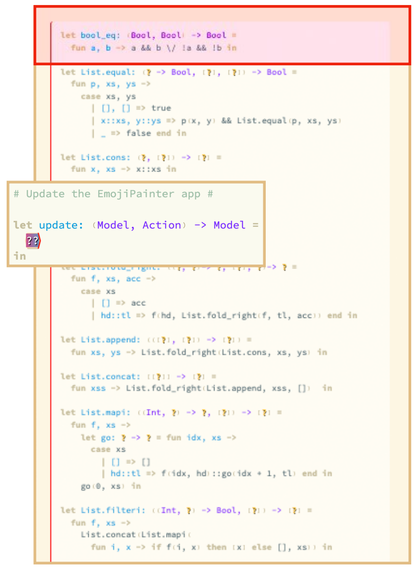


```
# A todo has a description and a status #
type Todo = (String, Bool) in

# A description input buffer and a todo list #
type Model = (String, (Todo)) in

type Action =
  + AddTodo
  + RemoveTodo(Int)
  + ToggleTodo(Int)
  + UpdateBuffer(String) in

type Update = (Model, Action) -> Model in

let Todo.eq: (Todo, Todo) -> Bool =
  fun (d1, s1), (d2, s2) ->
    d1 $== d2 && bool_eq(s1, s2) in

let Model.eq: (Model, Model) -> Bool =
  fun (b1, ts1), (b2, ts2) ->
    b1 $== b2 && List.equal(Todo.eq, ts1, ts2) in

let Model.init: Model = ("", []) in

type Event = +Inject(String,(Model,Action)->Model, Action) in
type Attr =
  + OnClick(() -> Event)
  + OnMouseDown(() -> Event)
  + OnInput(String -> Event)
  + Create(String, String)
  + Style((StyleAttr)) in
```

Contextualization

Strategies

- Text window
- Vector Retrieval
- **Repository-level**



Contextualization

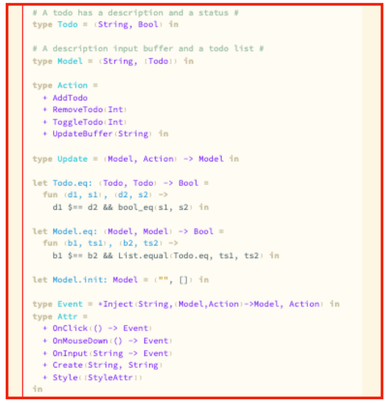
Strategies

- Text window
- Vector Retrieval
- **Repository-level**

Contextualization

Strategies

- Text window
- Vector Retrieval
- **Repository-level**



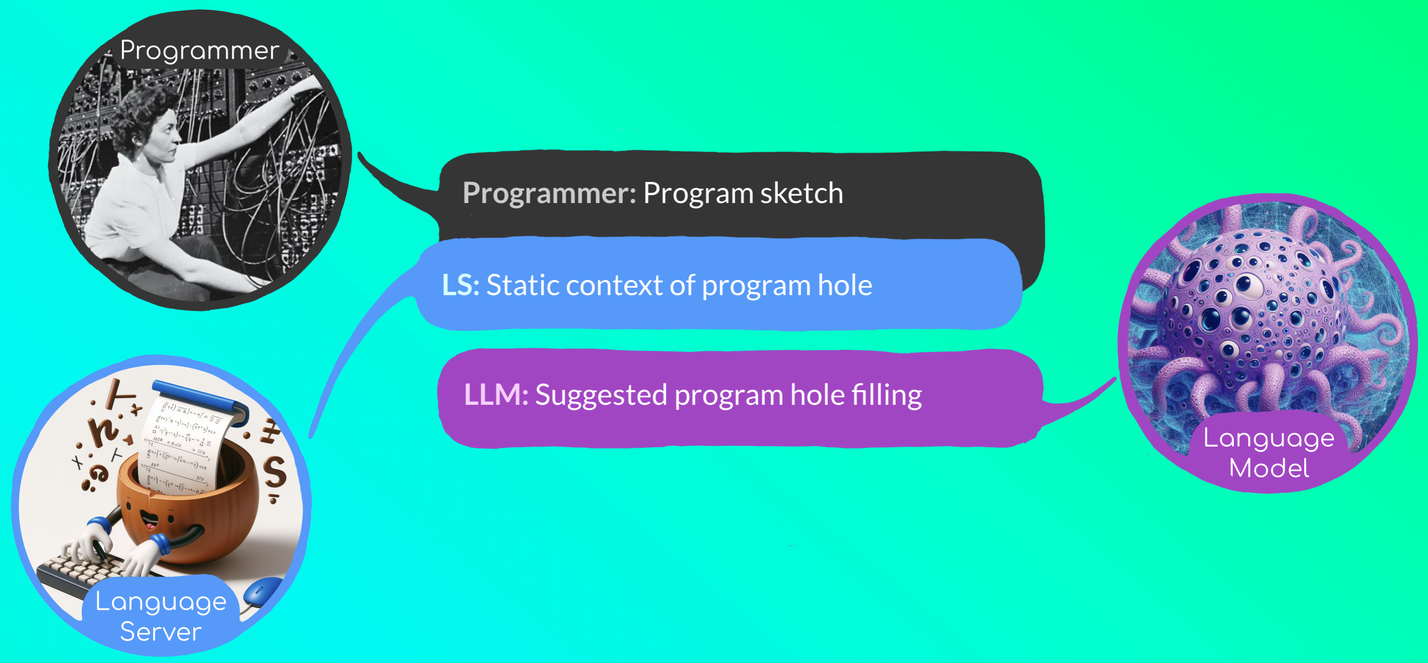
```
# A todo has a description and a status #
type Todo = (String, Bool) in

# A description input buffer and a todo list #
type Model = (String, (Todo)) in

type Action =
  + AddTodo
  + RemoveTodo(Int)
  + ToggleTodo(Int)
  + UpdateBuffer(String) in

type Update = (Model, Action) -> Model in

let Todo.eq: (Todo, Todo) -> Bool =
  fun ((d1, s1), (d2, s2)) ->
    d1 $== d2 && bool_eq(s1, s2) in

let Model.eq: (Model, Model) -> Bool =
  fun ((b1, ts1), (b2, ts2)) ->
    b1 $== b2 && List.equal(Todo.eq, ts1, ts2) in

let Model.init: Model = ("", []) in

type Event = +Inject(String,(Model,Action)->Model, Action) in
type Attr =
  + OnClick(() -> Event)
  + OnMouseDown(() -> Event)
  + OnInput(String -> Event)
  + Create(String, String)
  + Style((StyleAttr))
in
```

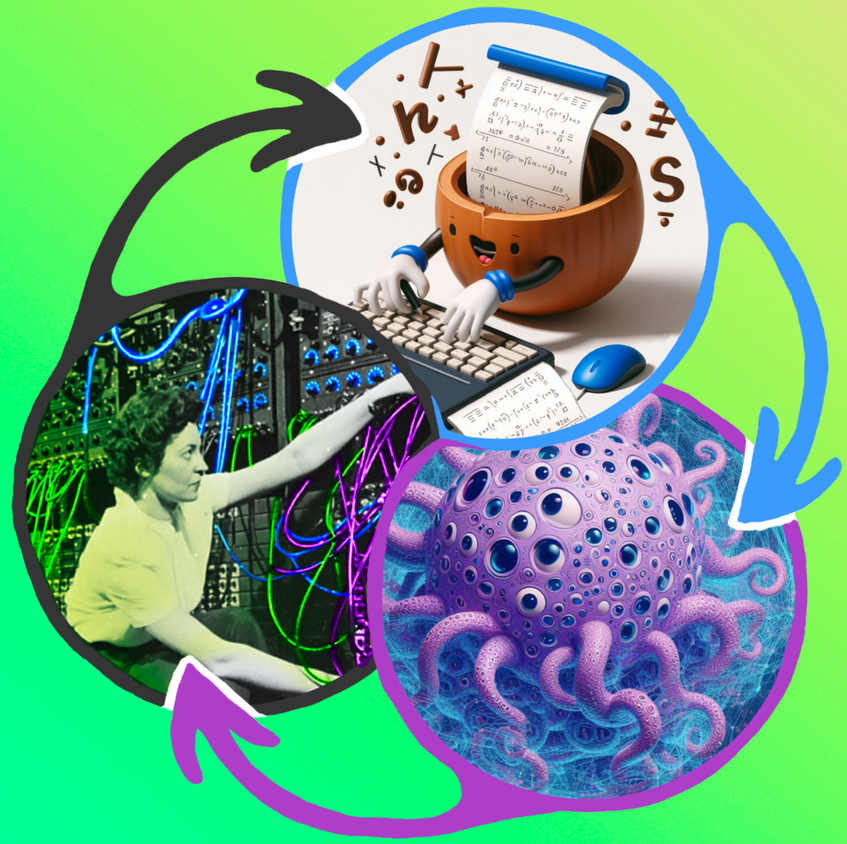
```
let todo_card : (Action -> Event, Int, Todo) -> Node =
  fun (go,idx, (descr: String, status: Bool)) ->
    Div((
      Style((
        Display("flex"),
        Gap("1em"))),
      OnClick(fun () -> go(ToggleTodo(idx)))),
      (
        Checkbox((OnClick(fun () -> go(RemoveTodo(idx)))), []),
        Div([], (Text(descr))),
        Text(if status then "Completed" else "Pending")
      )) in

let todos_deck: (Action -> Event, (Todo)) -> Node =
  fun go, todos ->
    if (List.is_empty(todos))
    then Text("You're caught up")
    else Div(
      (Create("class", "todos")),
      (Text("todos:"))
      @ List.mapi(fun i,t ->todo_card(go,i,t), todos)) in


let add_button: (Action -> Event) -> Node =
  fun go -> Div((
    Style((
      Display("flex"),
      JustifyContent("center"),
      BackgroundColor("#986"),
      BorderRadius("0.3em"),
      Cursor("pointer"))),
    OnClick(fun () -> go(AddTodo))),
    (Text("Add Todo"))) in

let buffer: (Action -> Event) -> Node =
  fun go -> Div(
```

```
# Update the EmojiPainter app #

let update: (Model, Action) -> Model =
  ??
in
```

OUR APPROACH

Static Contextualization with Typed Holes





System: Language crash course

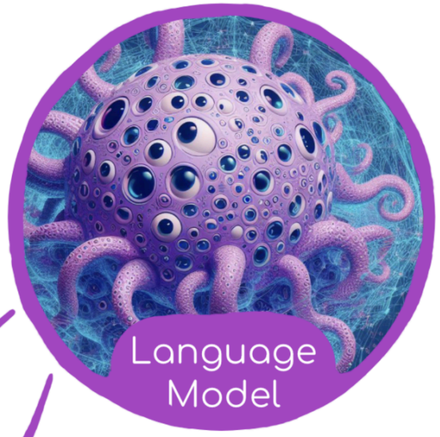
Programmer: Program sketch

LS: Static context of program hole

LLM: Suggested program hole filling

LS: Static error feedback on filled sketch

LLM: Corrected hole filling



System: Language crash course

Programmer: Program sketch

LS: Static context of program hole

LLM: Suggested program hole filling

LS: Static error feedback on filled sketch

LLM: Corrected hole filling

System: Language crash course

Programmer: Program sketch

LS: Static context of program hole

LLM: Suggested program hole filling

LS: Static error feedback on filled sketch

LLM: Corrected hole filling

System: Language crash course

Programmer: Program sketch

LS: Static context of program hole

LLM: Suggested program hole filling

LS: Static error feedback on filled sketch

LLM: Corrected hole filling

System: Language crash course

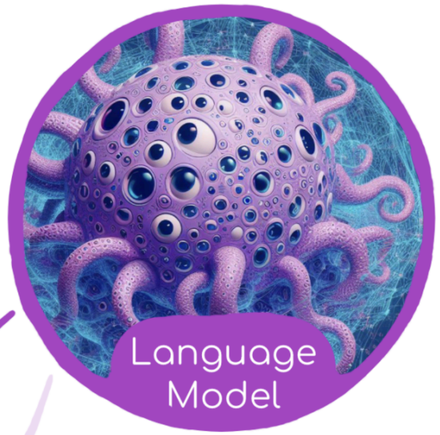
Programmer: Program sketch

LS: Static context of program hole

LLM: Suggested program hole filling

LS: Static error feedback on filled sketch

LLM: Corrected hole filling



System: Language crash course

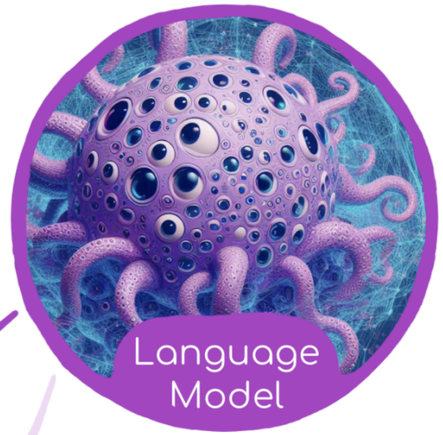
Programmer: Program sketch

LS: Static context of program hole

LLM: Suggested program hole filling

LS: Static error feedback on filled sketch

LLM: Corrected hole filling



System: Language crash course

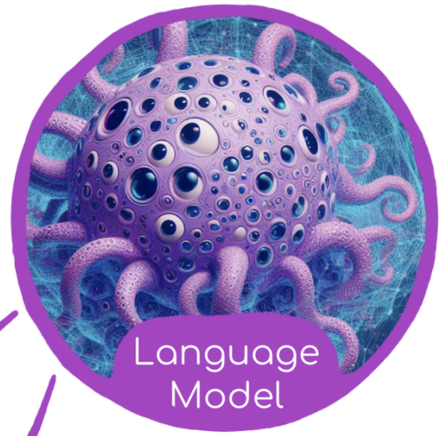
Programmer: Program sketch

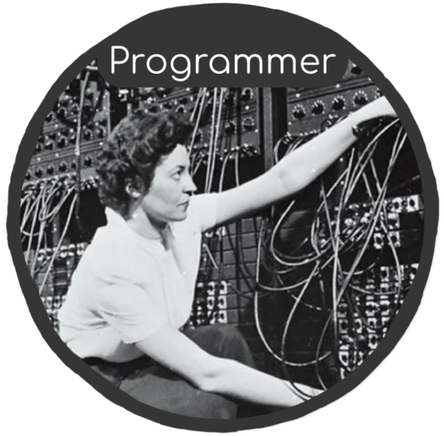
LS: Static context of program hole

LLM: Suggested program hole filling

LS: Static error feedback on filled sketch

LLM: Corrected hole filling



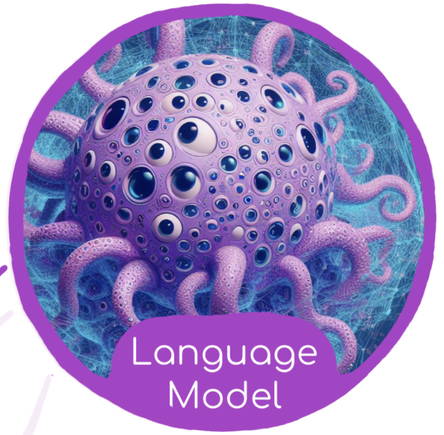


System: Language crash course

Programmer: Program sketch

LS: Static context of program hole

LLM: Suggested program hole filling



LS: Static error feedback on filled sketch

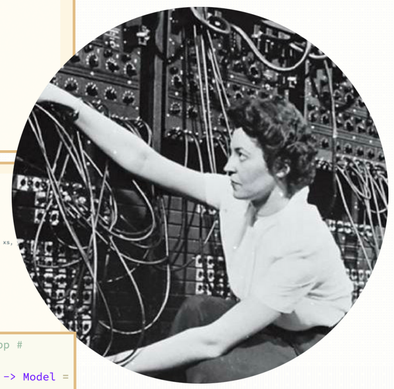
LLM: Corrected hole filling

Codebase

```
let todo_card : (Action -> Event, Int, Todo) -> Node =
  fun (go,idx, (descr: String, status: Bool)) ->
    Div(
      Style(
        Display "flex",
        Gap "1em"),
      OnClick(fun () -> go(ToggleTodo(idx))),

      Checkbox(OnClick(fun () -> go(RemoveTodo(idx)))), [],
      Div([], [Text(descr)]),
      Text(if status then "Completed" else "Pending")
    ) in

let todos_deck: (Action -> Event, (Todo)) -> Node =
  fun go, todos ->
    if (List.is_empty(todos))
    then Text("You're caught up")
    else Div(
      Create("class", "todos"),
      Text("todos:"),
      @ List.map(fun (i,t) ->todo_card(go,i,t), todos)) in


let add_button: (Action -> Event) -> Node =
  fun go -> Div(
    Style(
      Display("flex"),
      JustifyContent("center"),
      BackgroundColor("#336"),
      BorderRadius("0.3em"),
      Cursor("pointer")),
    OnClick(fun () -> go(AddTodo)),
    [Text("Add Todo")]) in

let buffer: (Action -> Event) -> Node =
  fun go -> Div(
    [],
    [TextInput(OnInput(fun s -> go(UpdateBuffer(s))), [])]) in
```

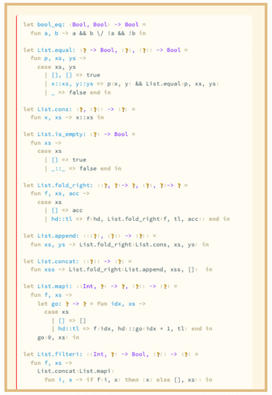
```
let bool_eq: (Bool, Bool) -> Bool =
  fun a, b -> a && b \/ !a && !b in

let List.equal: ('a -> Bool, ('a), ('a)) -> Bool =
  fun p, xs, ys ->
    case xs, ys
      | [], [] => true
      | x::xs, y::ys => p(x, y) && List.equal(p, xs, ys)
      | _ => false end in

let List.cons: ('a, ('a)) -> ('a) =
  fun x, xs -> x::xs in

let List.is_empty: ('a) -> Bool =
  fun xs ->
    case xs
      | [] => true
      | _::_ => false end in

let List.fold_right: (('a, 'b) -> 'b, ('a), 'b) -> 'b =
  fun f, xs, acc ->
    case xs
      | [] => acc
      | hd::tl => f(hd, List.fold_right(f, tl, acc)) end in

let List.append: (('a), ('a)) -> ('a) =
  fun xs, ys -> List.fold_right(List.cons, xs, ys) in

let List.concat: (('a)) -> ('a) =
  fun xss -> List.fold_right(List.append, xss, []) in

let List.map1: ((Int, 'a) -> 'a, ('a)) -> ('a) =
  fun f, xs ->
    let go: 'a -> 'a = fun idx, xs ->
      case xs
        | [] => []
        | hd::tl => f(idx, hd)::go(idx + 1, tl) end in
    go(0, xs) in

let List.filter: ((Int, 'a) -> Bool, ('a)) -> ('a) =
  fun f, xs ->
    List.concat(List.map1(
      fun i, x -> if f(i, x) then (x) else [], xs)) in
```

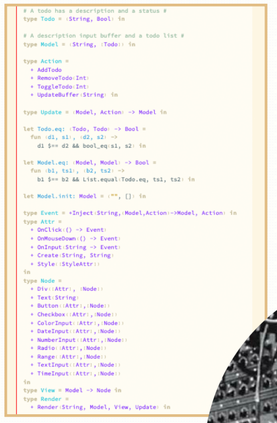
```
# A todo has a description and a status #
type Todo = (String, Bool) in

# A description input buffer and a todo list #
type Model = (String, (Todo)) in

type Action =
  + AddTodo
  + RemoveTodo(Int)
  + ToggleTodo(Int)
  + UpdateBuffer(String) in

type Update = (Model, Action) -> Model in

let Todo.eq: (Todo, Todo) -> Bool =
  fun (d1, s1), (d2, s2) ->
    d1 $== d2 && bool_eq(s1, s2) in

let Model.eq: (Model, Model) -> Bool =
  fun (b1, ts1), (b2, ts2) ->
    b1 $== b2 && List.equal(Todo.eq, ts1, ts2) in

let Model.init: Model = ("", []) in

type Event = +Inject(String,(Model,Action)->Model, Action) in
type Attr =
  + OnClick(() -> Event)
  + OnMouseDown(() -> Event)
  + OnInput(String -> Event)
  + Create(String, String)
  + Style(StyleAttr)
in
type Node =
  + Div(Attr, (Node))
  + Text(String)
  + Button((Attr), Node)
  + Checkbox((Attr), Node)
  + ColorInput((Attr), Node)
  + DateInput((Attr), Node)
  + NumberInput((Attr), Node)
  + Radio((Attr), Node)
  + Range((Attr), Node)
  + TextInput((Attr), Node)
  + TimeInput((Attr), Node)
in
type View = Model -> Node in
type Render =
  + Render(String, Model, View, Update) in
```

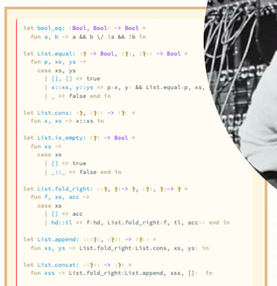


```
let bool_eq: (Bool, Bool) -> Bool =
  fun a, b -> a && b \/ !a && !b in

let List.equal: ('a -> Bool, ('a), ('a)) -> Bool =
  fun p, xs, ys ->
    case xs, ys
      | [], [] => true
      | x::xs, y::ys => p(x, y) && List.equal(p, xs, ys)
      | _ => false end in

let List.cons: ('a, ('a)) -> ('a) =
  fun x, xs -> x::xs in

let List.is_empty: ('a) -> Bool =
  fun xs ->
    case xs
      | [] => true
      | _::_ => false end in
```

```
# Update the EmojiPainter app #

let update: (Model, Action) -> Model =
  @
in
```

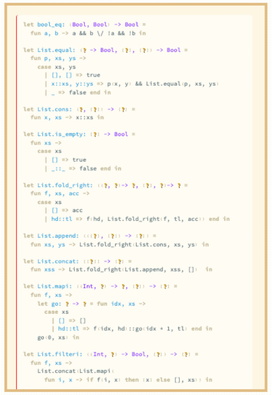


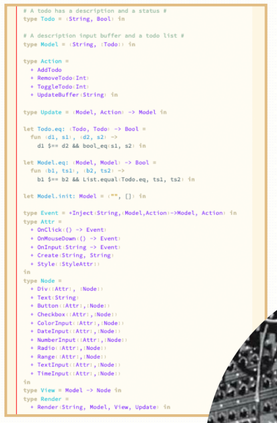


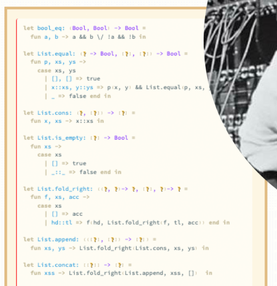


Program Sketch:

```
# Update the EmojiPainter app #


let update: (Model, Action) -> Model =
  |
in
```

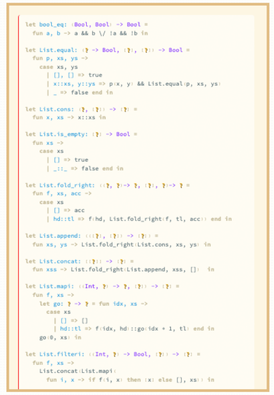


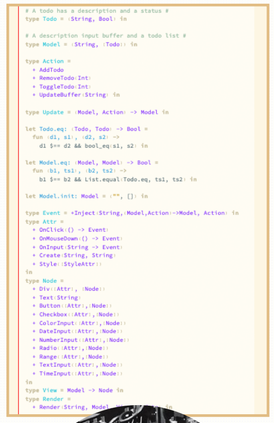


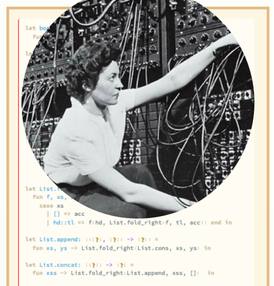


Program Sketch:

```
# Update the EmojiPainter app #


let update: (Model, Action) -> Model =
    ??
in
```

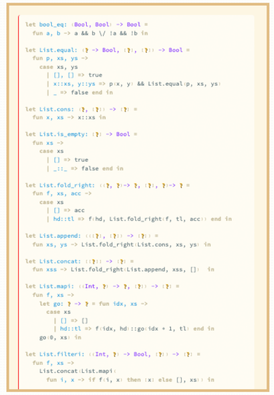


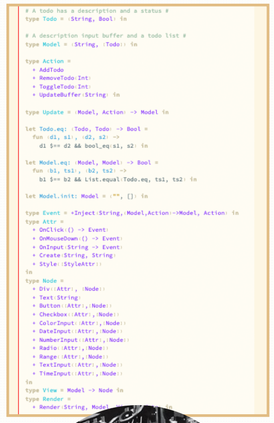


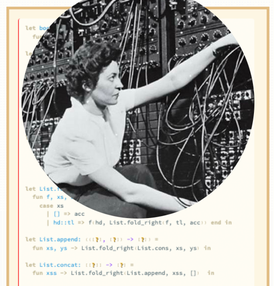


Program Sketch:

```
# Update the EmojiPainter app #


let update: (Model, Action) -> Model =
   ??
in
```







Expected Type

Fillable by any expression of type

```
(Model, Action) -> Model
```

Typing Context

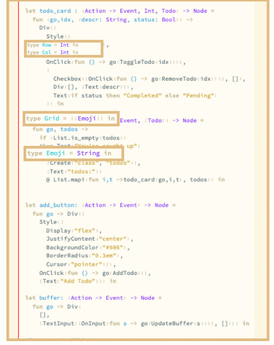
```
SelectEmoji : Emoji -> Actio
model_init : Model
updateGrid : (Grid, Row, Col
clearGrid : Grid -> Grid
fillRowInGrid : (Grid, Row,
```

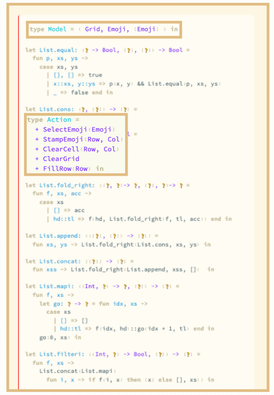
Γ EXP ? Empty expression hole

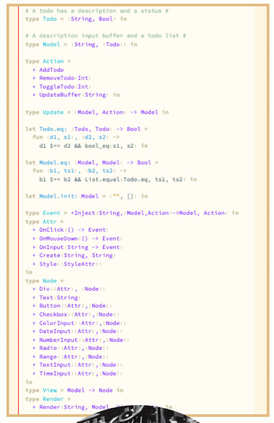
Program Sketch:

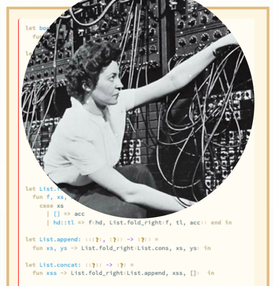
```
# Update the EmojiPainter app #


let update: (Model, Action) -> Model =


in
```

Relevant Types:









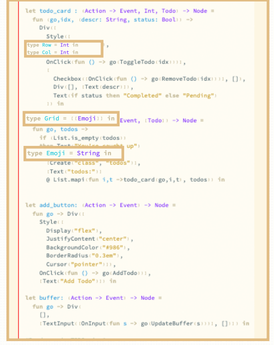
Fillable by any expression of type

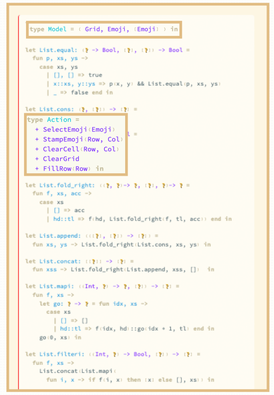
`(Model, Action) -> Model`

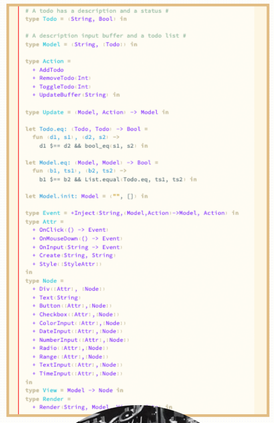
Program Sketch:

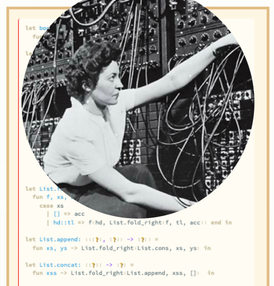
```
# Update the EmojiPainter app #

let update: (Model, Action) -> Model =
  ??
in
```

```
SelectEmoji : Emoji -> Actio
model_init : Model
updateGrid : (Grid, Row, Col
clearGrid : Grid -> Grid
fillRowInGrid : (Grid, Row,
Γ EXP ? Empty expression hole
```









Relevant Types:

Fillable by any expression of type

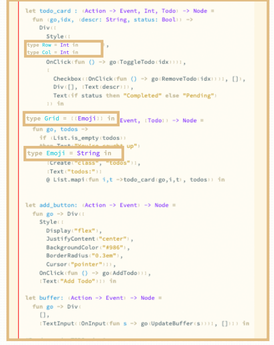
`(Model, Action) -> Model`

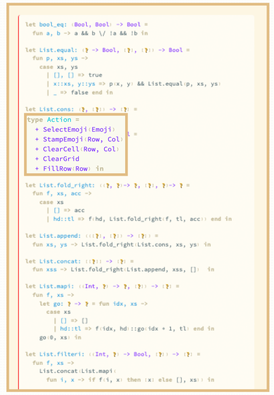
Program Sketch:

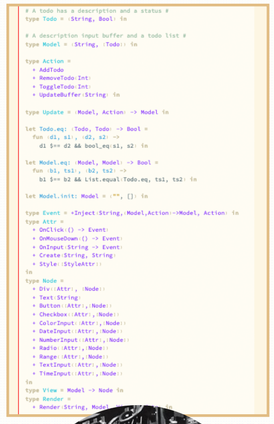
```
# Update the EmojiPainter app #

let update: (Model, Action) -> Model =
  ??
in
```

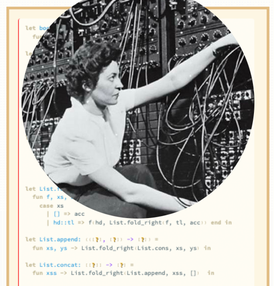
```
SelectEmoji : Emoji -> Actio
model_init : Model
updateGrid : (Grid, Row, Col
clearGrid : Grid -> Grid
fillRowInGrid : (Grid, Row,
Γ EXP ? Empty expression hole
```

Relevant Types:









Fillable by any expression of type

```
(Model, Action) -> Model
```

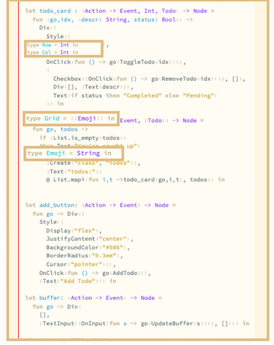
```
type Model = ( Grid, Emoji, [Emoji] ) in
```

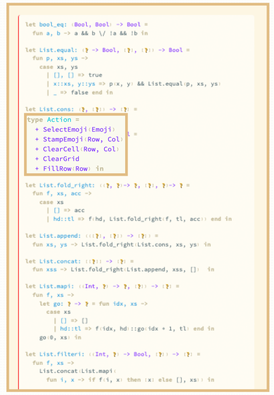
Program Sketch:

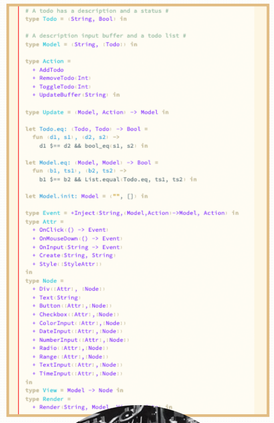
```
# Update the EmojiPainter app #

let update: (Model, Action) -> Model =
  ??
in
```

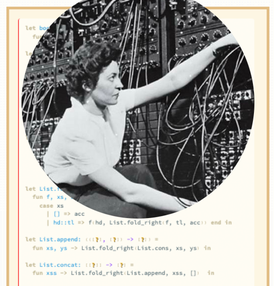
```
SelectEmoji : Emoji -> Actio
model_init : Model
updateGrid : (Grid, Row, Col
clearGrid : Grid -> Grid
fillRowInGrid : (Grid, Row,
```

Γ EXP ? Empty expression hole









Relevant Types:

Fillable by any expression of type
`(Model, Action) -> Model`

```
type Model = ( Grid, Emoji, [Emoji] ) in
```

Program Sketch:

```
# Update the EmojiPainter app #

let update: (Model, Action) -> Model =
  ??
in
```

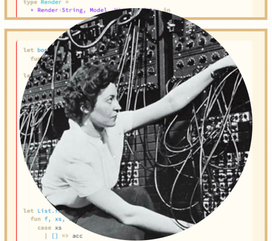
```
SelectEmoji : Emoji -> Actio
model_init : Model
updateGrid : (Grid, Row, Col
clearGrid : Grid -> Grid
fillRowInGrid : (Grid, Row,
Γ EXP ? Empty expression hole
```

Relevant Types:

```
type Action =
  + SelectEmoji(Emoji)
  + StampEmoji(Row, Col)
  + ClearCell(Row, Col)
  + ClearGrid
  + FillRow(Row) in
```

```
type Model = ( Grid, Emoji, [Emoji] ) in
```

Fillable by any expression of type
`(Model, Action) -> Model`



Program Sketch:

```
# Update the EmojiPainter app #

let update: (Model, Action) -> Model =
  ??
in
```

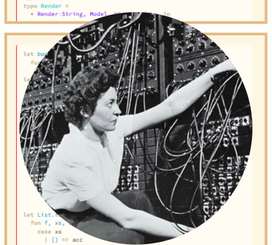
```
SelectEmoji : Emoji -> Actio
model_init : Model
updateGrid : (Grid, Row, Col
clearGrid : Grid -> Grid
fillRowInGrid : (Grid, Row,
Γ EXP ? Empty expression hole
```

Relevant Types:

```
type Action =
  + SelectEmoji(Emoji)
  + StampEmoji(Row, Col)
  + ClearCell(Row, Col)
  + ClearGrid
  + FillRow(Row) in
```

```
type Model = ( Grid, Emoji, [Emoji] ) in
```

Fillable by any expression of type
`(Model, Action) -> Model`



Program Sketch:

```
# Update the EmojiPainter app #

let update: (Model, Action) -> Model =
  ??
in
```

```
SelectEmoji : Emoji -> Actio
model_init : Model
updateGrid : (Grid, Row, Col
clearGrid : Grid -> Grid
fillRowInGrid : (Grid, Row,
Γ EXP ? Empty expression hole
```

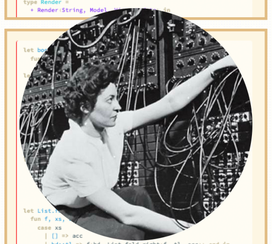
Relevant Types:

```
type Grid = [[Emoji]] in
```

```
type Action =
  + SelectEmoji(Emoji)
  + StampEmoji(Row, Col)
  + ClearCell(Row, Col)
  + ClearGrid
  + FillRow(Row) in
```

```
type Model = ( Grid, Emoji, [Emoji] ) in
```

Fillable by any expression of type
`(Model, Action) -> Model`



Program Sketch:

```
# Update the EmojiPainter app #

let update: (Model, Action) -> Model =
  ??
in
```

```
SelectEmoji : Emoji -> Actio
model_init : Model
updateGrid : (Grid, Row, Col
clearGrid : Grid -> Grid
fillRowInGrid : (Grid, Row,
Γ EXP ? Empty expression hole
```

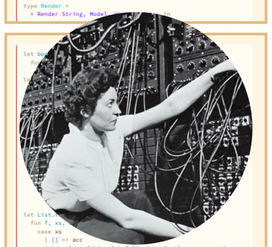
Relevant Types:

```
type Grid = [[Emoji]] in
```

```
type Action =
  + SelectEmoji(Emoji)
  + StampEmoji(Row, Col)
  + ClearCell(Row, Col)
  + ClearGrid
  + FillRow(Row) in
```

```
type Model = ( Grid, Emoji [Emoji] ) in
```

Fillable by any expression of type
`(Model, Action) -> Model`



Program Sketch:

```
# Update the EmojiPainter app #

let update: (Model, Action) -> Model =
  ??
in
```

```
SelectEmoji : Emoji -> Actio
model_init : Model
updateGrid : (Grid, Row, Col
clearGrid : Grid -> Grid
fillRowInGrid : (Grid, Row,
Γ EXP ? Empty expression hole
```

Relevant Types:

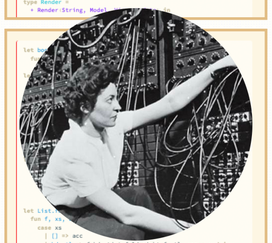
```
type Emoji = String in
```

```
type Grid = [[Emoji]] in
```

```
type Action =
  + SelectEmoji(Emoji)
  + StampEmoji(Row, Col)
  + ClearCell(Row, Col)
  + ClearGrid
  + FillRow(Row) in
```

```
type Model = ( Grid, Emoji, [Emoji] ) in
```

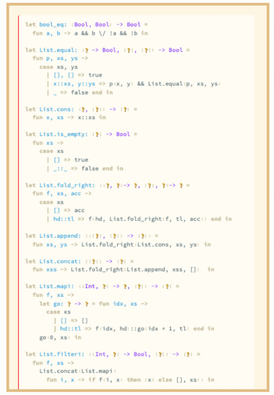
Fillable by any expression of type
`(Model, Action) -> Model`

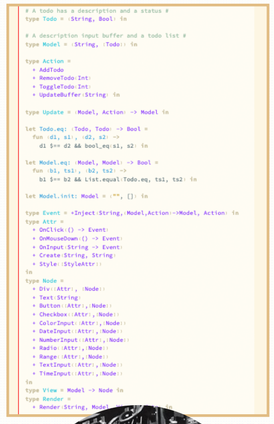


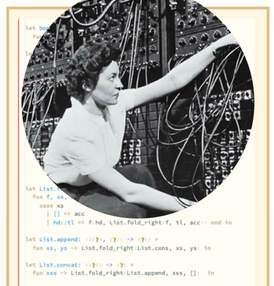
Program Sketch:

```
# Update the EmojiPainter app #

let update: (Model, Action) -> Model =
  ??
in
```

```
SelectEmoji : Emoji -> Actio
model_init : Model
updateGrid : (Grid, Row, Col
clearGrid : Grid -> Grid
fillRowInGrid : (Grid, Row,
Γ EXP ? Empty expression hole
```







Relevant Types:

```
type Emoji = String in
```

```
type Grid = [[Emoji]] in
```

```
type Action =
  + SelectEmoji(Emoji)
  + StampEmoji(Row, Col)
  + ClearCell(Row, Col)
  + ClearGrid
  + FillRow(Row) in
```

```
type Model = ( Grid, Emoji, [Emoji] ) in
```

Fillable by any expression of type

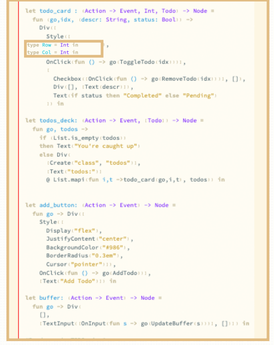
```
(Model, Action) -> Model
```

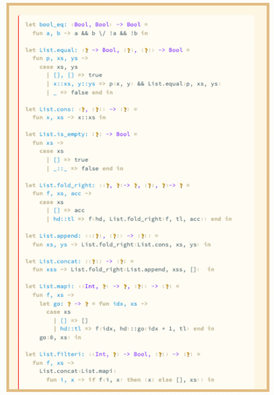
```
SelectEmoji : Emoji -> Actio
model_init : Model
updateGrid : (Grid, Row, Col
clearGrid : Grid -> Grid
fillRowInGrid : (Grid, Row,
```

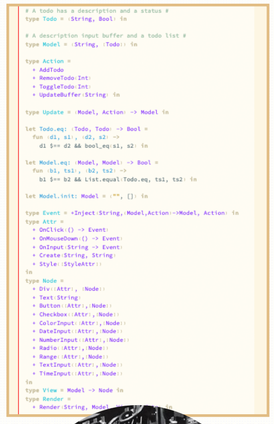
Γ EXP ? Empty expression hole

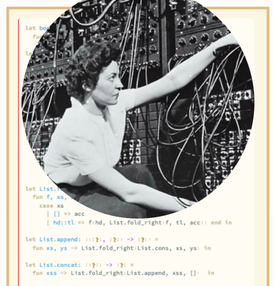
Program Sketch:

```
# Update the EmojiPainter app #

let update: (Model, Action) -> Model =
  ??
in
```









Relevant Types:

```
type Emoji = String in
```

```
type Grid = [[Emoji]] in
```

```
type Action =
  + SelectEmoji(Emoji)
  + StampEmoji(Row, Col)
  + ClearCell(Row, Col)
  + ClearGrid
  + FillRow(Row) in
```

```
type Model = ( Grid, Emoji, [Emoji] ) in
```

Relevant Headers:

Type of hole = `(Model, Action) -> Model`

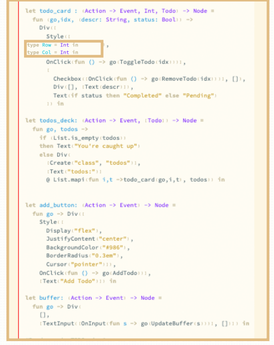
Fillable by any expression of type
`(Model, Action) -> Model`

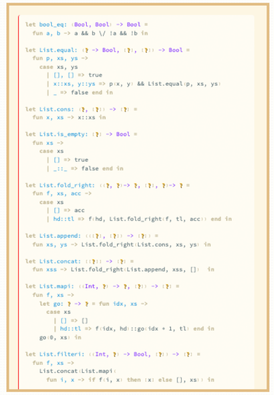
```
SelectEmoji : Emoji -> Actio
model_init : Model
updateGrid : (Grid, Row, Col
clearGrid : Grid -> Grid
fillRowInGrid : (Grid, Row,
```

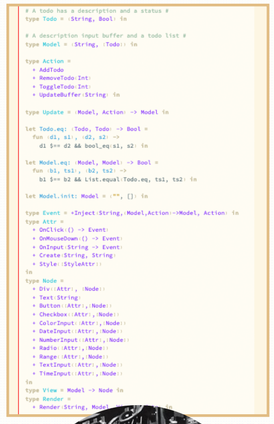
Γ EXP ? Empty expression hole

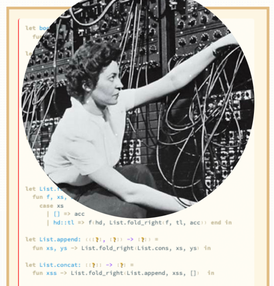
Program Sketch:

```
# Update the EmojiPainter app #

let update: (Model, Action) -> Model =
  ??
in
```









Relevant Types:

```
type Emoji = String in
```

```
type Grid = [[Emoji]] in
```

```
type Action =
  + SelectEmoji(Emoji)
  + StampEmoji(Row, Col)
  + ClearCell(Row, Col)
  + ClearGrid
  + FillRow(Row) in
```

```
type Model = ( Grid, Emoji, [Emoji] ) in
```

Relevant Headers:

Type of hole = `(Model, Action) -> Model`

Target types =

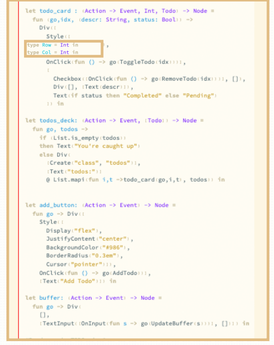
Fillable by any expression of type
`(Model, Action) -> Model`

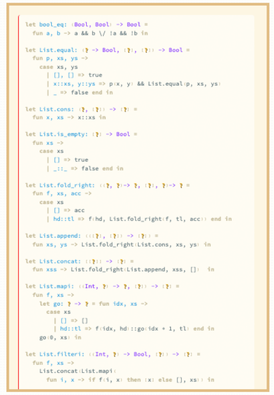
```
SelectEmoji : Emoji -> Actio
model_init : Model
updateGrid : (Grid, Row, Col
clearGrid : Grid -> Grid
fillRowInGrid : (Grid, Row,
```

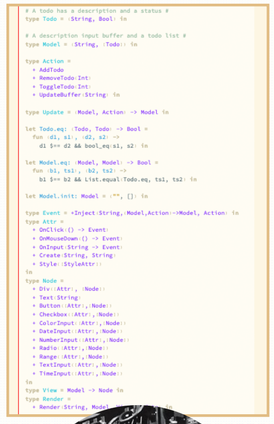
Γ EXP ? Empty expression hole

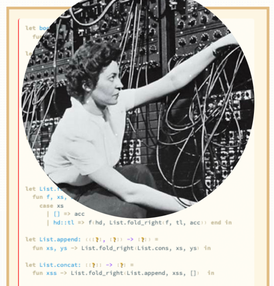
Program Sketch:

```
# Update the EmojiPainter app #

let update: (Model, Action) -> Model =
  ??
in
```









Relevant Types:

```
type Emoji = String in
```

```
type Grid = [[Emoji]] in
```

```
type Action =
  + SelectEmoji(Emoji)
  + StampEmoji(Row, Col)
  + ClearCell(Row, Col)
  + ClearGrid
  + FillRow(Row) in
```

```
type Model = ( Grid, Emoji, [Emoji] ) in
```

Relevant Headers:

Type of hole = `(Model, Action) -> Model`

Target types = `(Model, Action) -> Model`

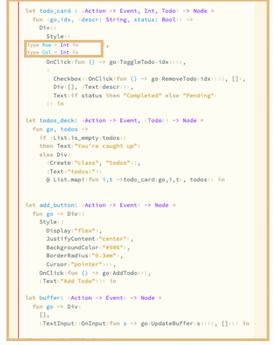
Fillable by any expression of type
`(Model, Action) -> Model`

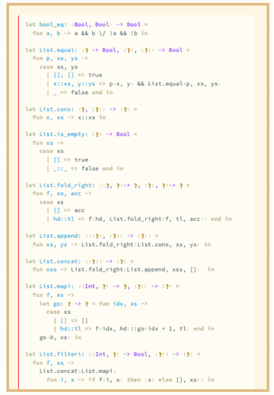
```
SelectEmoji : Emoji -> Actio
model_init : Model
updateGrid : (Grid, Row, Col
clearGrid : Grid -> Grid
fillRowInGrid : (Grid, Row,
```

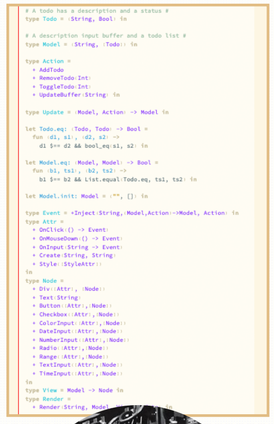
Γ EXP ? Empty expression hole

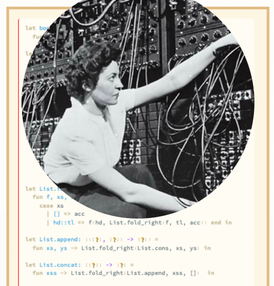
Program Sketch:

```
# Update the EmojiPainter app #

let update: (Model, Action) -> Model =
  ??
in
```









Relevant Types:

```
type Emoji = String in
```

```
type Grid = [[Emoji]] in
```

```
type Action =
  + SelectEmoji(Emoji)
  + StampEmoji(Row, Col)
  + ClearCell(Row, Col)
  + ClearGrid
  + FillRow(Row) in
```

```
type Model = ( Grid, Emoji, [Emoji] ) in
```

Relevant Headers:

Type of hole = `(Model, Action) -> Model`

Target types = `(Model, Action) -> Model`, `Model`

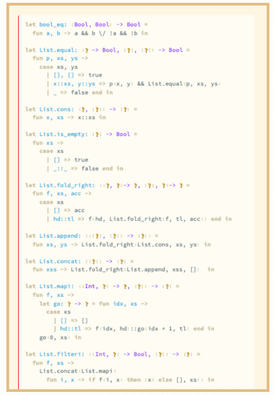
Fillable by any expression of type
`(Model, Action) -> Model`

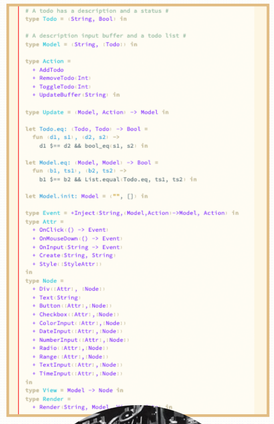
```
SelectEmoji : Emoji -> Actio
model_init : Model
updateGrid : (Grid, Row, Col
clearGrid : Grid -> Grid
fillRowInGrid : (Grid, Row,
```

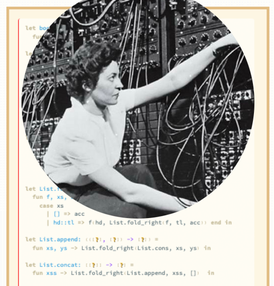
Γ EXP ? Empty expression hole

Program Sketch:

```
# Update the EmojiPainter app #

let update: (Model, Action) -> Model =
  ??
in
```







Relevant Types:

```
type Emoji = String in
```

```
type Grid = [[Emoji]] in
```

```
type Action =
  + SelectEmoji(Emoji)
  + StampEmoji(Row, Col)
  + ClearCell(Row, Col)
  + ClearGrid
  + FillRow(Row) in
```

```
type Model = ( Grid, Emoji, [Emoji] ) in
```

Relevant Headers:

Type of hole = `(Model, Action) -> Model`

Target types = `(Model, Action) -> Model`, `Model`

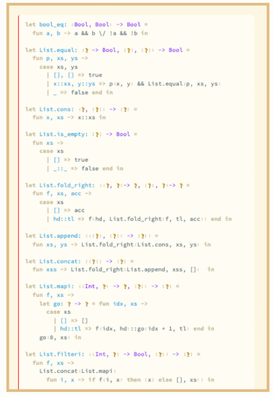
Fillable by any expression of type
`(Model, Action) -> Model`

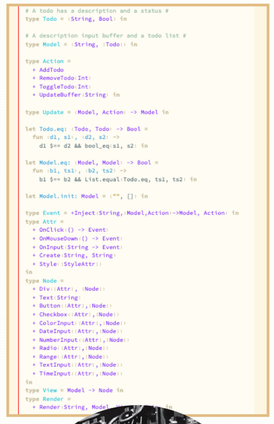
```
SelectEmoji : Emoji -> Actio
model_init : Model
updateGrid : (Grid, Row, Col
clearGrid : Grid -> Grid
fillRowInGrid : (Grid, Row,
```

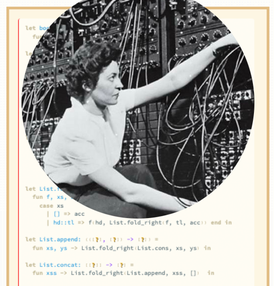
Γ EXP ? Empty expression hole

Program Sketch:

```
# Update the EmojiPainter app #

let update: (Model, Action) -> Model =
  ??
in
```







Relevant Types:

```
type Emoji = String in
```

```
type Grid = [[Emoji]] in
```

```
type Action =
  + SelectEmoji(Emoji)
  + StampEmoji(Row, Col)
  + ClearCell(Row, Col)
  + ClearGrid
  + FillRow(Row) in
```

```
type Model = ( Grid, Emoji, [Emoji] ) in
```

Relevant Headers:

Type of hole = `((String, (Grid, Emoji, [Emoji])), Action) -> (Grid, Emoji, [Emoji])`

Target types = `(Model, Action) -> Model, Model`

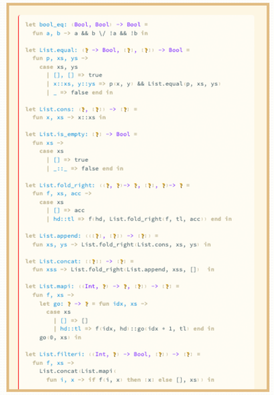
Fillable by any expression of type
`(Model, Action) -> Model`

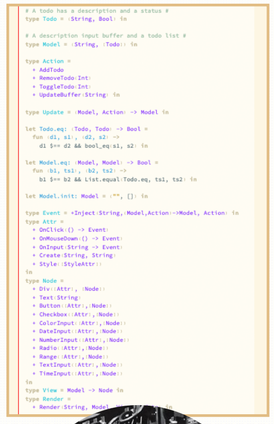
```
SelectEmoji : Emoji -> Actio
model_init : Model
updateGrid : (Grid, Row, Col
clearGrid : Grid -> Grid
fillRowInGrid : (Grid, Row,
```

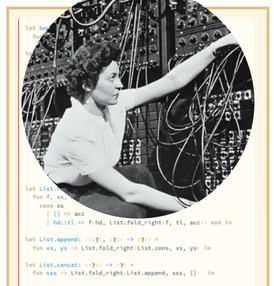
Γ EXP ? Empty expression hole

Program Sketch:

```
# Update the EmojiPainter app #

let update: (Model, Action) -> Model =
  ??
in
```







Relevant Types:

```
type Emoji = String in
```

```
type Grid = [[Emoji]] in
```

```
type Action =
  + SelectEmoji(Emoji)
  + StampEmoji(Row, Col)
  + ClearCell(Row, Col)
  + ClearGrid
  + FillRow(Row) in
```

```
type Model = ( Grid, Emoji, [Emoji] ) in
```

Relevant Headers:

Type of hole = `((String, (Grid, Emoji, [Emoji])), Action) -> (Grid, Emoji, [Emoji])`

Target types = `(Model, Action) -> Model`, `Model`, `Grid`, `Emoji`, `[Emoji]`

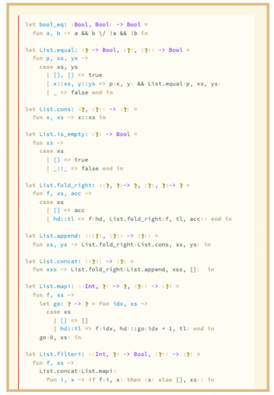
Fillable by any expression of type
`(Model, Action) -> Model`

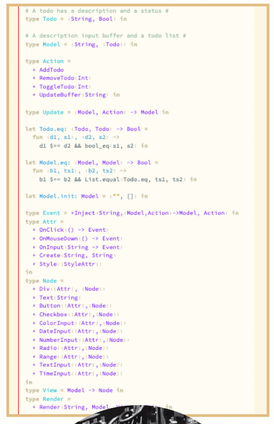
```
SelectEmoji : Emoji -> Actio
model_init : Model
updateGrid : (Grid, Row, Col
clearGrid : Grid -> Grid
fillRowInGrid : (Grid, Row,
```

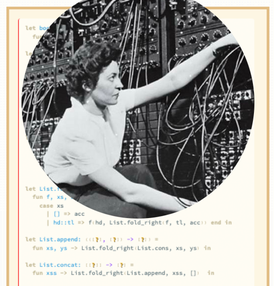
Γ EXP ? Empty expression hole

Program Sketch:

```
# Update the EmojiPainter app #

let update: (Model, Action) -> Model =
  ??
in
```







Relevant Types:

```
type Emoji = String in
```

```
type Grid = [[Emoji]] in
```

```
type Action =
  + SelectEmoji(Emoji)
  + StampEmoji(Row, Col)
  + ClearCell(Row, Col)
  + ClearGrid
  + FillRow(Row) in
```

```
type Model = ( Grid, Emoji, [Emoji] ) in
```

Relevant Headers:

Type of hole = ((String, (Grid, Emoji, [Emoji])), Action) -> (Grid, Emoji, [Emoji])

Target types = `(Model, Action) -> Model` Model, Grid, Emoji, [Emoji]

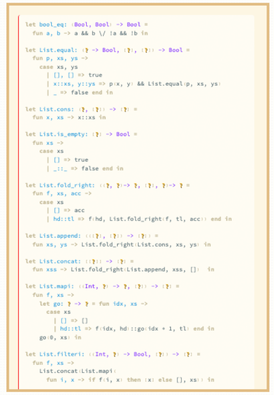
Fillable by any expression of type
`(Model, Action) -> Model`

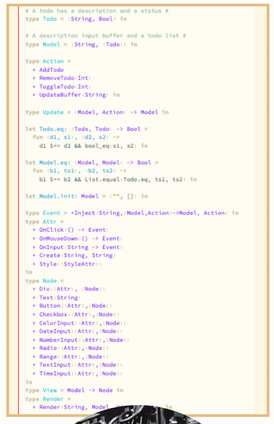
```
SelectEmoji : Emoji -> Actio
model_init : Model
updateGrid : (Grid, Row, Col
clearGrid : Grid -> Grid
fillRowInGrid : (Grid, Row,
```

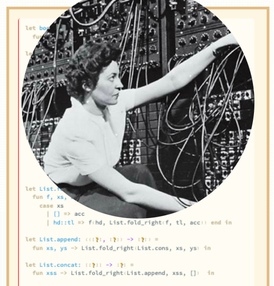
Γ EXP ? Empty expression hole

Program Sketch:

```
# Update the EmojiPainter app #

let update: (Model, Action) -> Model =
  ??
in
```







Relevant Types:

```
type Emoji = String in
```

```
type Grid = [[Emoji]] in
```

```
type Action =
  + SelectEmoji(Emoji)
  + StampEmoji(Row, Col)
  + ClearCell(Row, Col)
  + ClearGrid
  + FillRow(Row) in
```

```
type Model = ( Grid, Emoji, [Emoji] ) in
```

Relevant Headers:

Type of hole = `((String, (Grid, Emoji, [Emoji])), Action) -> (Grid, Emoji, [Emoji])`

Target types = `(Model, Action) -> Model` | `Model` | `Grid, Emoji, [Emoji]`

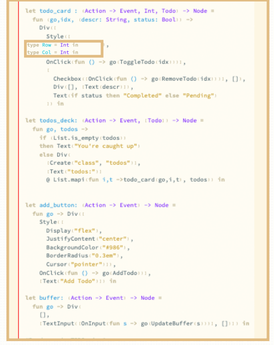
Fillable by any expression of type
`(Model, Action) -> Model`

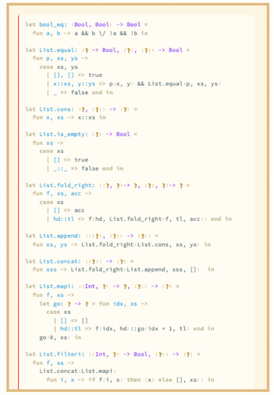
```
SelectEmoji : Emoji -> Actio
model_init : Model
updateGrid : (Grid, Row, Col
clearGrid : Grid -> Grid
fillRowInGrid : (Grid, Row,
```

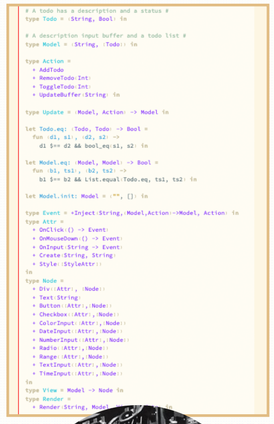
Γ EXP ? Empty expression hole

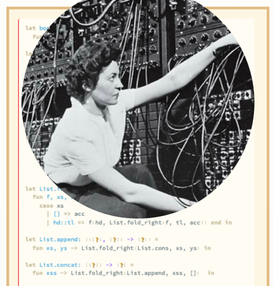
Program Sketch:

```
# Update the EmojiPainter app #

let update: (Model, Action) -> Model =
  ??
in
```









Relevant Types:

```
type Emoji = String in
```

```
type Grid = [[Emoji]] in
```

```
type Action =
  + SelectEmoji(Emoji)
  + StampEmoji(Row, Col)
  + ClearCell(Row, Col)
  + ClearGrid
  + FillRow(Row) in
```

```
type Model = ( Grid, Emoji, [Emoji] ) in
```

Relevant Headers:

Type of hole = `((String, (Grid, Emoji, [Emoji])), Action) -> (Grid, Emoji, [Emoji])`

Target types = `(Model, Action) -> Model` | `Model` | `Grid, Emoji, [Emoji]`

Fillable by any expression of type
`(Model, Action) -> Model`

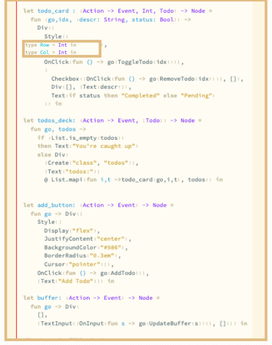
```
let model_init: Model = … in
```

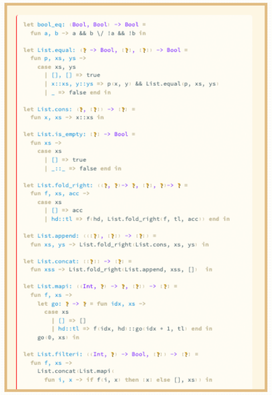
```
SelectEmoji : Emoji -> Actio
model_init : Model
updateGrid : (Grid, Row, Col
clearGrid : Grid -> Grid
fillRowInGrid : (Grid, Row,
```

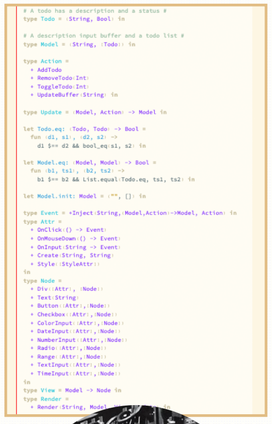
Γ EXP ? Empty expression hole

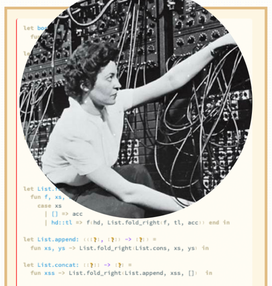
Program Sketch:

```
# Update the EmojiPainter app #

let update: (Model, Action) -> Model =
  ??
in
```









Relevant Types:

```
type Emoji = String in
```

```
type Grid = [[Emoji]] in
```

```
type Action =
  + SelectEmoji(Emoji)
  + StampEmoji(Row, Col)
  + ClearCell(Row, Col)
  + ClearGrid
  + FillRow(Row) in
```

```
type Model = ( Grid, Emoji, [Emoji] ) in
```

Relevant Headers:

Type of hole = `((String, (Grid, Emoji, [Emoji])), Action) -> (Grid, Emoji, [Emoji])`

Fillable by any expression of type
`(Model, Action) -> Model`

Target types = `(Model, Action) -> Model, Model, Grid, Emoji, [Emoji]`

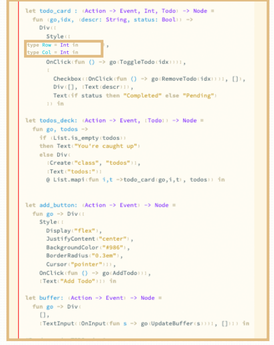
```
let model_init: Model = … in
```

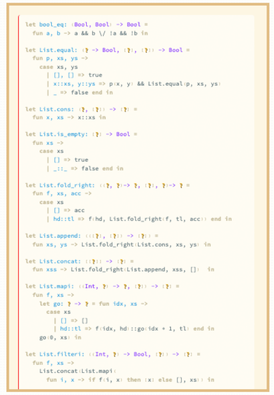
```
SelectEmoji : Emoji -> Actio
model_init : Model
updateGrid : (Grid, Row, Col
clearGrid : Grid -> Grid
fillRowInGrid : (Grid, Row,
```

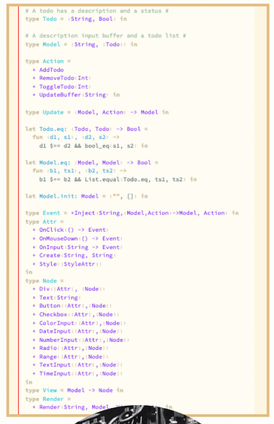
Γ EXP ? Empty expression hole

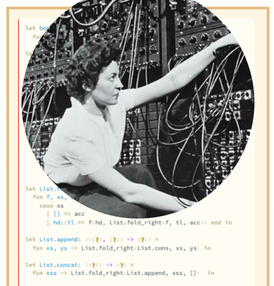
Program Sketch:

```
# Update the EmojiPainter app #

let update: (Model, Action) -> Model =
  ??
in
```









Relevant Types:

```
type Emoji = String in
```

```
type Grid = [[Emoji]] in
```

```
type Action =
  + SelectEmoji(Emoji)
  + StampEmoji(Row, Col)
  + ClearCell(Row, Col)
  + ClearGrid
  + FillRow(Row) in
```

```
type Model = ( Grid, Emoji, [Emoji] ) in
```

Relevant Headers:

Type of hole = `((String, (Grid, Emoji, [Emoji])), Action) -> (Grid, Emoji, [Emoji])`

Fillable by any expression of type
`(Model, Action) -> Model`

Target types = `(Model, Action) -> Model` `Model` `Grid` `Emoji` `[Emoji]`

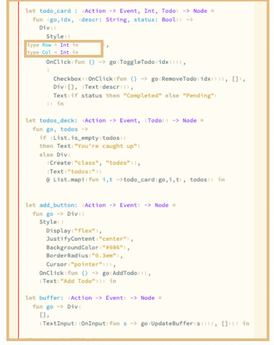
```
let model_init: Model = … in
let updateGrid: (Grid, Row, Col, Emoji) -> Grid = … in
let clearGrid: Grid -> Grid = … in
let fillRowInGrid: (Grid, Row, Emoji) -> Grid = … in
```

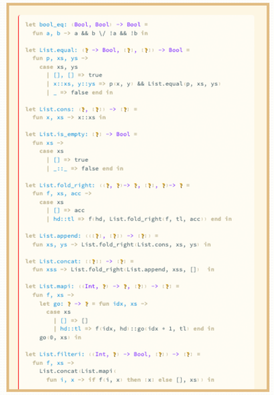
```
SelectEmoji : Emoji -> Actio
model_init : Model
updateGrid : (Grid, Row, Col
clearGrid : Grid -> Grid
fillRowInGrid : (Grid, Row,
```

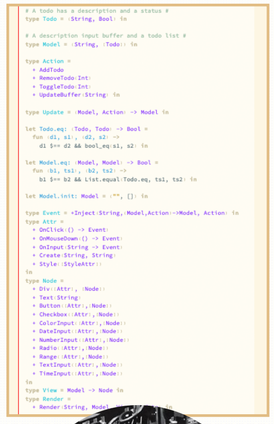
Γ EXP ? Empty expression hole

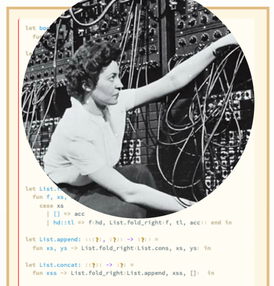
Program Sketch:

```
# Update the EmojiPainter app #

let update: (Model, Action) -> Model =
  ??
in
```









Relevant Types:

```
type Emoji = String in
```

```
type Grid = [[Emoji]] in
```

```
type Action =
  + SelectEmoji(Emoji)
  + StampEmoji(Row, Col)
  + ClearCell(Row, Col)
  + ClearGrid
  + FillRow(Row) in
```

```
type Model = ( Grid, Emoji, [Emoji] ) in
```

Relevant Headers:

Type of hole = `((String, (Grid, Emoji, [Emoji])), Action) -> (Grid, Emoji, [Emoji])`

Fillable by any expression of type
`(Model, Action) -> Model`

Target types = `(Model, Action) -> Model, Model, Grid, Emoji, [Emoji]`

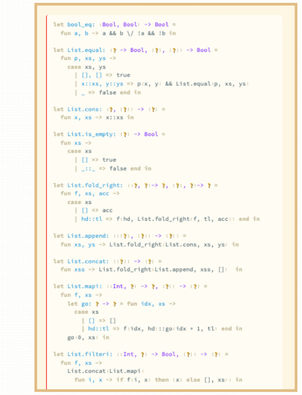
```
let model_init: Model = … in
let updateGrid: (Grid, Row, Col, Emoji) -> Grid = … in
let clearGrid: Grid -> Grid = … in
let fillRowInGrid: (Grid, Row, Emoji) -> Grid = … in
```

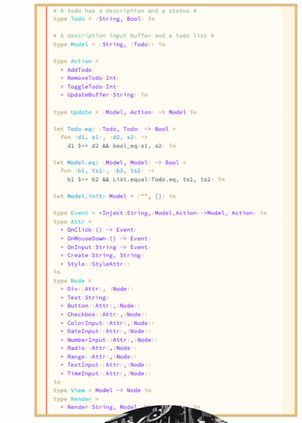
```
SelectEmoji : Emoji -> Actio
model_init : Model
updateGrid : (Grid, Row, Col
clearGrid : Grid -> Grid
fillRowInGrid : (Grid, Row,
```

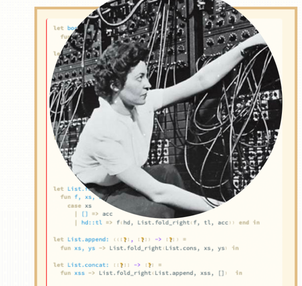
Γ EXP ? Empty expression hole

Program Sketch:

```
# Update the EmojiPainter app #

let update: (Model, Action) -> Model =
  ??
in
```







Expected Type:

Fillable by any expression of type
`(Model, Action) -> Model`

Relevant Types:

```
type Emoji = String in
```

```
type Grid = [[Emoji]] in
```

```
type Action =
  + SelectEmoji(Emoji)
  + StampEmoji(Row, Col)
  + ClearCell(Row, Col)
  + ClearGrid
  + FillRow(Row) in
```

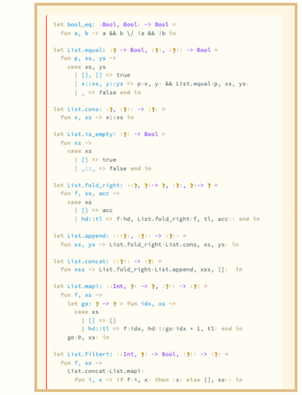
```
type Model = ( Grid, Emoji, [Emoji] ) in
```

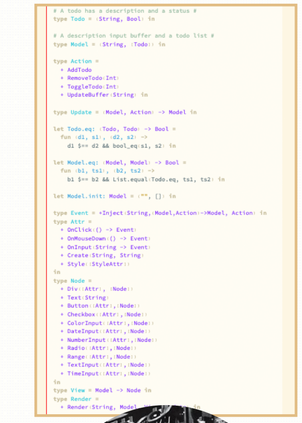
Relevant Headers:

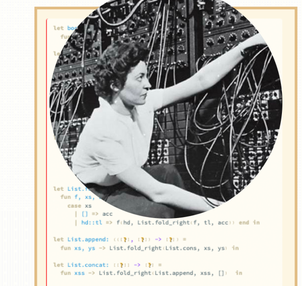
```
let model_init: Model = … in
let updateGrid: (Grid, Row, Col, Emoji) -> Grid = … in
let clearGrid: Grid -> Grid = … in
let fillRowInGrid: (Grid, Row, Emoji) -> Grid = … in
```

Program Sketch:

```
# Update the EmojiPainter app #

let update: (Model, Action) -> Model =
  ??
in
```







Expected Type:

Fillable by any expression of type
`(Model, Action) -> Model`

Relevant Types:

```
type Emoji = String in
```

```
type Grid = [[Emoji]] in
```

```
type Action =
  + SelectEmoji(Emoji)
  + StampEmoji(Row, Col)
  + ClearCell(Row, Col)
  + ClearGrid
  + FillRow(Row) in
```

```
type Model = ( Grid, Emoji, [Emoji] ) in
```

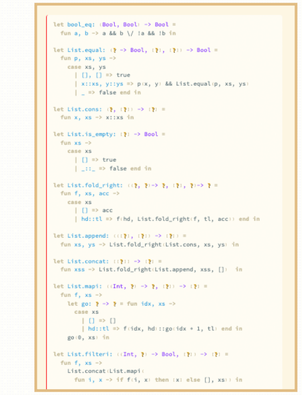
Relevant Headers:

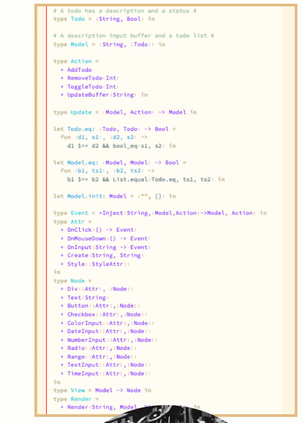
```
let model_init: Model = … in
let updateGrid: (Grid, Row, Col, Emoji) -> Grid = … in
let clearGrid: Grid -> Grid = … in
let fillRowInGrid: (Grid, Row, Emoji) -> Grid = … in
```

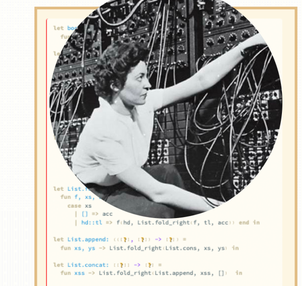
Program Sketch:

```
# Update the EmojiPainter app #

let update: (Model, Action) -> Model =
  ??
in
```

```
# ChatLSP: The hole ?? expects a value having          #
# a type consistent with (Model, Action) -> Model.     #
# The following type definitions are likely relevant: #
type Emoji = String in
type Action = SelectEmoji(Emoji) + StampEmoji(Row, Col…
type Grid = [[Emoji]] in
type Model = (Grid, Emoji, [Emoji]) in
# Consider these values relevant to the expected type #
let model_init: Model = … in
let updateGrid: (Grid, Row, Col, Emoji) -> Grid = … in
let clearGrid: Grid -> Grid = … in
let fillRowInGrid: (Grid, Row, Emoji) -> Grid = … in
# Complete the following program sketch:              #
# Update the EmojiPainter app #
let update: (Model, Action) -> Model =
 ??
```





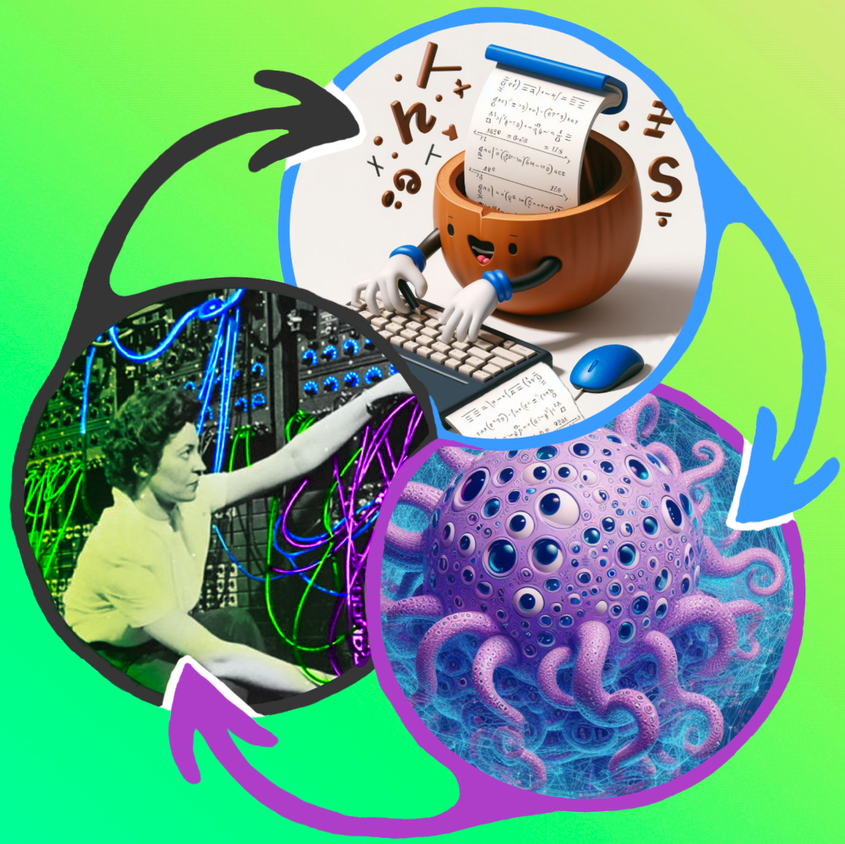


```
let update: (Model, Action) -> Model =
  fun model, action ->
    case action
      | SelectEmoji(emoji) =>
      let (_, _, emoji_palette) = model in
      (updateGrid(model, -1, -1, emoji), emoji, emoji_palette)
      | StampEmoji((row, col)) =>
      let (grid, selected_emoji, emoji_palette) = model in
      (updateGrid(grid, row, col, selected_emoji),
       selected_emoji, emoji_palette)
      | ClearCell((row, col)) =>
      let (grid, selected_emoji, emoji_palette) = model in
      (updateGrid(grid, row, col, ""), selected_emoji,
       emoji_palette)
      | ClearGrid =>
      let (_, selected_emoji, emoji_palette) = model in
      (clearGrid(grid), selected_emoji, emoji_palette)
      | FillRow(row) =>
      let (grid, selected_emoji, emoji_palette) = model in
      (fillRowInGrid(grid, row, selected_emoji), selected_emoji,
       emoji_palette)
    end in
```

```
# ChatLSP: The hole ?? expects a value having          #
# a type consistent with (Model, Action) -> Model.     #
# The following type definitions are likely relevant: #
type Emoji = String in
type Action = SelectEmoji(Emoji) + StampEmoji(Row, Col…
type Grid = [[Emoji]] in
type Model = (Grid, Emoji, [Emoji]) in
# Consider these values relevant to the expected type #
let model_init: Model = … in
let updateGrid: (Grid, Row, Col, Emoji) -> Grid = … in
let clearGrid: Grid -> Grid = … in
let fillRowInGrid: (Grid, Row, Emoji) -> Grid = … in
# Complete the following program sketch:              #
# Update the EmojiPainter app #
let update: (Model, Action) -> Model =
 ??
```

Evaluation

Design



Evaluation

Problems with existing evals

- Standard evals generally consist of single-file tasks which are low-context (e.g. HumanEval, EvalPlus, LiveCodeBench)
- Repo-level evals tend to be language exclusive (often focusing on dynamic langs like Python), and focus on syntactic rather than semantic (unit) testing (e.g. RepoEval, RepoBench, CrossCodeEval)

MVUBench

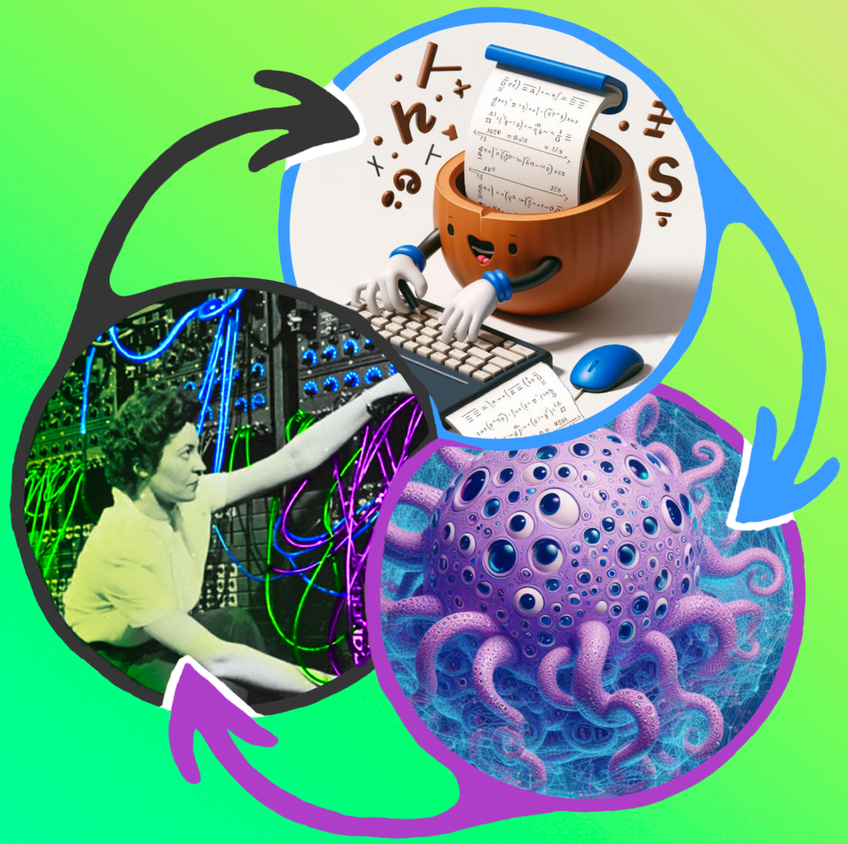
Model-View-Update Web Apps

- New suite of small-but-full applications developed from scratch
- Minimal external dependencies (easy to port across languages)
- Many domain-specific data types defined across different files

  - **Todo (*TO*)**: Maintains a list of tasks
  - **Room Booking (*BO*)**: Manages a room booking schedule
  - **Emoji Painter (*EM*)**: Paints emoji stamps on a small canvas
  - **Playlist Manager (*PL*)**: Manages a music playlist
  - **Password Strength Checker (*PA*)**: Rates a password via a dynamic set of properties

Evaluation

Results



Results: Hazel + GPT-4

Retrieval		Error	Average % Tests Pass					
Types	Headers	Rounds	BO	EM	PA	PL	TO	ALL
no	no	no	0	0	0	0	0	0
		yes	0	0	0	0	0	0
	yes	no	1	0	0	0	0	0
		yes	0	0	0	0	0	0
yes	no	no	26	0	7	10	15	12
		yes	31	1	40	53	47	34
	yes	no	28	79	29	68	35	48
		yes	41	95	93	81	71	76

```
let update: (Model, Action) -> Model =
  fun model, action ->
    case action
    | Paint(row, col, emoji) => updateGrid(model.grid, row, col, emoji)
    | FillRow(row, emoji) => fillRowInGrid(model.grid, row, emoji)
    | Clear => clearGrid(model.grid)
    end in
```

Relevant values but no types

Results for guided completion (20 trials per, temperature 0.6)

Results: Hazel + GPT-4

Retrieval		Error	Average % Tests Pass					
Types	Headers	Rounds	BO	EM	PA	PL	TO	ALL
no	no	no	0	0	0	0	0	0
		yes	0	0	0	0	0	0
	yes	no	1	0	0	0	0	0
		yes	0	0	0	0	0	0
yes	no	no	26	0	7	10	15	12
		yes	31	1	40	53	47	34
	yes	no	28	79	29	68	35	48
		yes	41	95	93	81	71	76

Results for guided completion (20 trials per, temperature 0.6)

```
let update: (Model, Action) -> Model =
  fun model, action ->
    let (grid, selected_emoji, emoji_palette), (action) =
      model, action in
    case action
    | SelectEmoji(emoji) => (grid, emoji, emoji_palette)
    | StampEmoji((row, col)) =>
    let grid = Grid.stamp(row, col, selected_emoji, grid) in
    (grid, selected_emoji, emoji_palette)
    | ClearCell((row, col)) =>
    let grid = Grid.clear(row, col, grid) in
    (grid, selected_emoji, emoji_palette)
    | ClearGrid => (Grid.empty_grid(), selected_emoji, emoji_palette)
```

Type definitions but no values

Results: Hazel + GPT-4

Retrieval		Error	Average % Tests Pass					
Types	Headers	Rounds	BO	EM	PA	PL	TO	ALL
no	no	no	0	0	0	0	0	0
		yes	0	0	0	0	0	0
	yes	no	1	0	0	0	0	0
		yes	0	0	0	0	0	0
yes	no	no	26	0	7	10	15	12
		yes	31	1	40	53	47	34
	yes	no	28	79	29	68	35	48
		yes	41	95	93	81	71	76

Results for guided completion (20 trials per, temperature 0.6)

```
let update: (Model, Action) -> Model =
  fun model, action ->
    let (grid, current_emoji, emoji_list) = model in
    case action
    | SelectEmoji(emoji) => (grid, emoji, emoji_list)
    | StampEmoji((row, col)) =>
    (updateGrid(grid, row, col, current_emoji), current_emoji, emoji_list)
    | ClearCell((row, col)) =>
    (updateGrid(grid,row, col, ""), current_emoji, emoji_list)
    | ClearGrid => (clearGrid(grid), current_emoji, emoji_list)
    | FillRow(row) =>
     (fillRowInGrid(grid, row, current_emoji), current_emoji, emoji_list)
    end in
```

Both values and types (correct implementation)

Results: Hazel + GPT-4 vs Baselines

Retrieval		Error	Average % Tests Pass					
Types	Headers	Rounds	BO	EM	PA	PL	TO	ALL
no	no	no	0	0	0	0	0	0
		yes	0	0	0	0	0	0
	yes	no	1	0	0	0	0	0
		yes	0	0	0	0	0	0
yes	no	no	26	0	7	10	15	12
		yes	31	1	40	53	47	34
yes	yes	no	28	79	29	68	35	48
		yes	41	95	93	81	71	76

Vector Retrieval	Error	Average % Tests Pass					
	Rounds	BO	EM	PA	PL	TO	ALL
yes	no	0	0	0	15	33	10
	yes	0	3	0	20	67	18

Results for guided completion (20 trials per, temperature 0.6)

Results: Hazel + GPT-4 vs Baselines

Retrieval		Error	Average % Tests Pass					
Types	Headers	Rounds	BO	EM	PA	PL	TO	ALL
no	no	no	0	0	0	0	0	0
		yes	0	0	0	0	0	0
	yes	no	1	0	0	0	0	0
		yes	0	0	0	0	0	0
yes	no	no	26	0	7	10	15	12
		yes	31	1	40	53	47	34
yes	yes	no	28	79	29	68	35	48
		yes	41	95	93	81	71	76

```
1  # SNIPPET 1 #
2  , ts2) in
3
4  let Model.init: Model = ("", []) in
5
6  type Action =
7    + AddTodo
8    + RemoveTodo(Int)
9    + ToggleTodo(Int)
10   + UpdateBuffer(String) in
```

Vector Retrieval	Error	Average % Tests Pass					
	Rounds	BO	EM	PA	PL	TO	ALL
yes	no	0	0	0	15	33	10
	yes	0	3	0	20	67	18

Results for guided completion (20 trials per, temperature 0.6)

Results: Hazel + GPT-4 vs Baselines

Retrieval		Error	Average % Tests Pass					
Types	Headers	Rounds	BO	EM	PA	PL	TO	ALL
no	no	no	0	0	0	0	0	0
		yes	0	0	0	0	0	0
	yes	no	1	0	0	0	0	0
		yes	0	0	0	0	0	0
yes	no	no	26	0	7	10	15	12
		yes	31	1	40	53	47	34
yes	yes	no	28	79	29	68	35	48
		yes	41	95	93	81	71	76

Exhaustive Retrieval	Error	Average % Tests Pass					
	Rounds	BO	EM	PA	PL	TO	ALL
yes	no	9	90	61	43	69	54
	yes	43	100	91	81	80	79

Results for guided completion (20 trials per, temperature 0.6)

Results: Typescript + GPT-4

Retrieval		Error	Average % Tests Pass					
Types	Headers	Rounds	BO	EM	PA	PL	TO	ALL
no	no	no	0	0	0	0	0	0
		yes	0	0	0	0	0	0
	yes	no	0	0	0	0	0	0
		yes	0	4	0	3	0	1
yes	no	no	46	26	43	85	66	53
		yes	69	39	46	93	66	63
	yes	no	47	95	83	64	75	73
		yes	58	99	85	86	75	80

TypeScript GPT-4

Retrieval		Error	Average % Tests Pass					
Types	Headers	Rounds	BO	EM	PA	PL	TO	ALL
no	no	no	0	0	0	0	0	0
		yes	0	0	0	0	0	0
	yes	no	1	0	0	0	0	0
		yes	0	0	0	0	0	0
yes	no	no	26	0	7	10	15	12
		yes	31	1	40	53	47	34
	yes	no	28	79	29	68	35	48
		yes	41	95	93	81	71	76

Hazel GPT-4

Results: Typescript + GPT4

Retrieval		Error	Average % Tests Pass					
Types	Headers	Rounds	BO	EM	PA	PL	TO	ALL
no	no	no	0	0	0	0	0	0
		yes	0	0	0	0	0	0
	yes	no	0	0	0	0	0	0
		yes	0	4	0	3	0	1
yes	no	no	46	26	43	85	66	53
		yes	69	39	46	93	66	63
	yes	no	47	95	83	64	75	73
		yes	58	99	85	86	75	80

TypeScript GPT-4

Retrieval		Error	Average % Tests Pass					
Types	Headers	Rounds	BO	EM	PA	PL	TO	ALL
no	no	no	0	0	0	0	0	0
		yes	0	0	0	0	0	0
	yes	no	1	0	0	0	0	0
		yes	0	0	0	0	0	0
yes	no	no	26	0	7	10	15	12
		yes	31	1	40	53	47	34
	yes	no	28	79	29	68	35	48
		yes	41	95	93	81	71	76

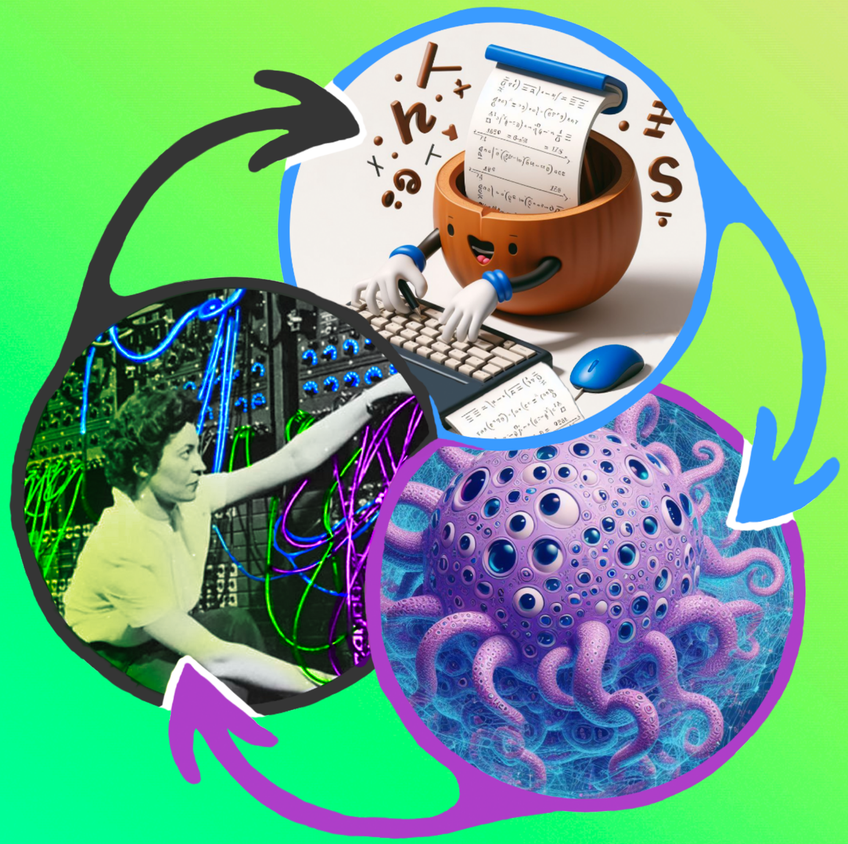
Hazel GPT-4

- See paper for more results, including re-running the GPT-4 experiments using the open source StarCoder2-15B LLM

ChatLSP??

5.1 ChatLSP API Methods

(1) **aiTutorial**: A constant (lexical-context-independent) method for low resource languages (like Hazel) to specify a textual tutorial intended for LLMs having robust support for in-context learning. For high resource languages, the default implementation will simply return a string stating which language is in use.
(2) **expectedType**: Returns a string specifying the expected type at the cursor, if available
(3) **retrieveRelevantTypes**: Returns a string containing type definitions that may be relevant at the cursor location
(4) **retrieveRelevantHeaders**: Returns a string containing headers that may be relevant at the cursor location
(5) **errorReport**: Returns an error report that can be used to determine if an error round is needed, and if so, how the feedback should be presented to the LLM.



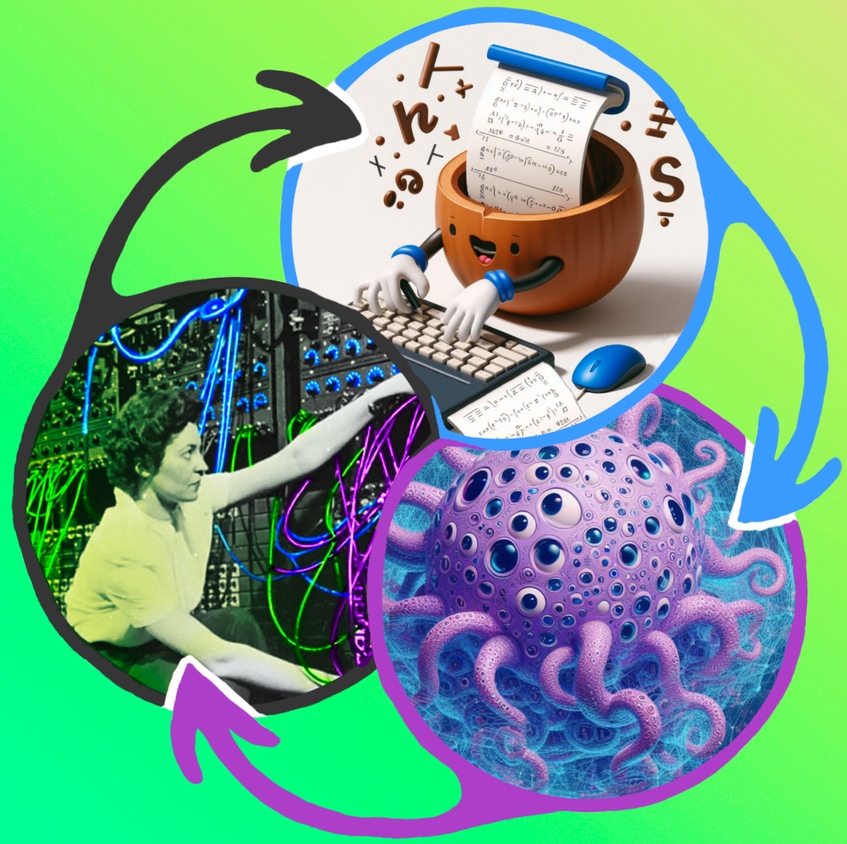
ChatLSP??

5.1 ChatLSP API Methods

(1) **aiTutorial**: A constant (lexical-context-independent) method for low resource languages (like Hazel) to specify a textual tutorial intended for LLMs having robust support for in-context learning. For high resource languages, the default implementation will simply return a string stating which language is in use.
(2) **expectedType**: Returns a string specifying the expected type at the cursor, if available
(3) **retrieveRelevantTypes**: Returns a string containing type definitions that may be relevant at the cursor location
(4) **retrieveRelevantHeaders**: Returns a string containing headers that may be relevant at the cursor location
(5) **errorReport**: Returns an error report that can be used to determine if an error round is needed, and if so, how the feedback should be presented to the LLM.

Talk to Jacob

TypeScript Static Contextualization,
VSCode Extension



Related Work

Repository-level:
RepoCoder (Zhang et al. EMNLP23)
CoCoMIC (Ding et al. LREC-COLING 2024)
Repo-Level Prompt Gen
(Shrivastava et al. ICML23)

(with emphasis on) **Semantic Context:**
STALL+ (Liu et al. 2024)
RLCoder (Wang et al. 2024)
Better Context Makes Better Code Language Models (Pei et al. AAAI23)
CodeTrek (Pashakhanloo et al. ICLR22)
Dehallucinator (Eghbali et al. 2024)
Copiloting the Copilots
(Wei et al, ESEC/FSE 2023)
IDECoder (Li et al. LLM4Code24)

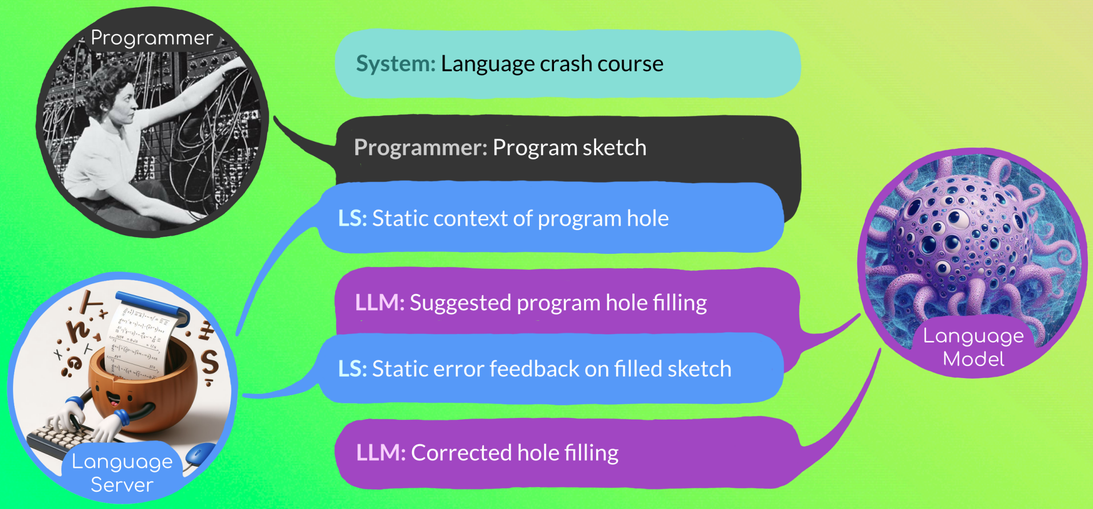
AutoCodeRover (Zhang et al. 2024)
Private-library-oriented code generation with LLMs (Zan et al. 2023)

Contextualizing Proofs: Towards Neural Synthesis for SMT-Assisted Proof-Oriented Programming (Chakraborty et al. ICSE2024)

Program Repair / Error Correction: The Fact Selection Problem in LLM-Based Program Repair (Parasaram et al. 2024)
Repair is nearly generation (Joshi et al. AAAI'23)

And in industry:
Sourcegraph Cody, Cursor, Zed, Aider

Thanks for Listening!



Conclusion & Future Directions

- **(In)human factors:** When considering IDE features for LLMs, we've found it useful to consider how they would play out for humans. Where does this framing work/fail?
- **LLMs need IDEs, too**: Giving models access to a full TUI (Text User Interface) version of Hazel; fine-tuning models on Hazel user edit actions histories
- Scaffolding and refining complex code changes using type-driven development and semantic edit actions (leveraging explicit programming strategies ala LaToza)