

CS450_(section 2)


High Level Languages

UMass Boston Computer Science

Wednesday, September 11, 2024




The Other Side ...

←  r/programmingcirclejerk • 2 yr. ago
LAUAR gofmt urself

Lisp is the worst family of languages in existence. Literally nothing can change my mind. Cool tool thingy, but you basically made a tool to teach blasphemy

It's a language that syntactically is awful. It sacrifices arbitrary elegance for practicality (human readability)

Racket does not compare to real, actual programming languages. In the real world, however, software developers use actual, practical languages like Python, C++ and to a lesser extent Javascript. The thought of using Racket never crosses their mutable variable-corrupted minds.

←  r/uwaterloo • 5 yr. ago
itsatrap12121

I HATE RACKET

UoT first year students are learning **python**

WHY DO WE HAVE TO DO **FUCKING RACKET** SHIT

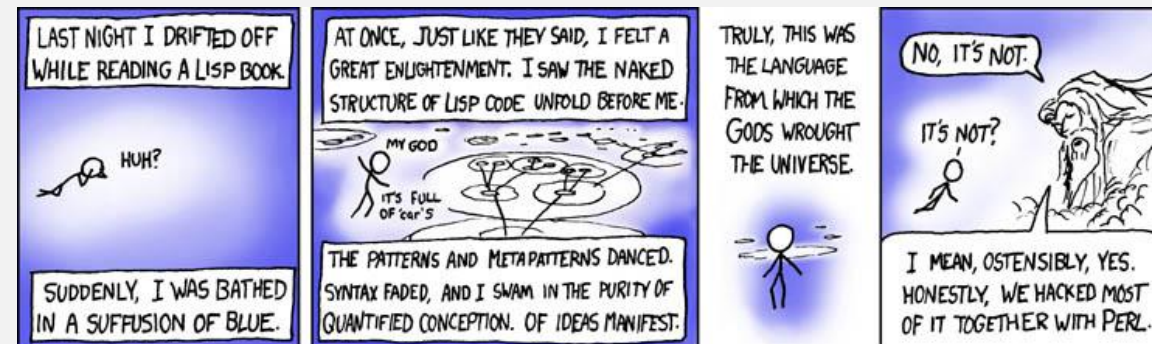
RACKET IS GARBAGE

Fortunately ... this course is not about Lisp, Racket, or any other language. **It is language-agnostic!**

It's about general, high-level programming principles ... that can be used when programming in any language

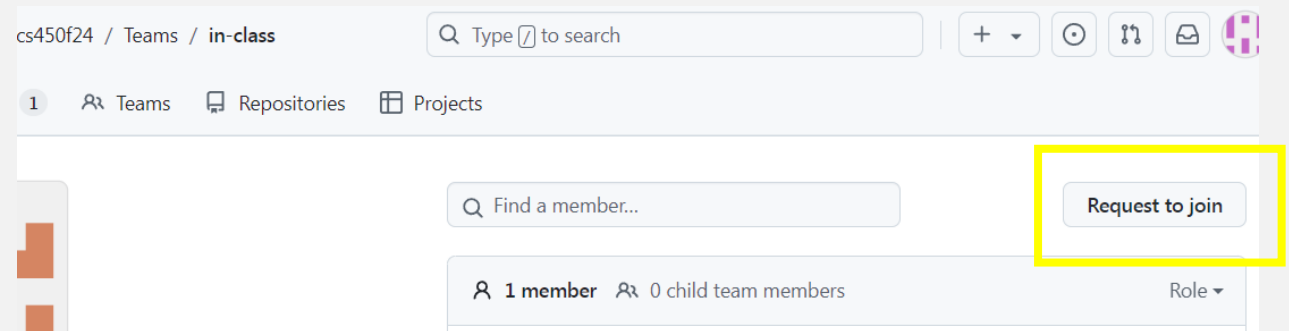
Logistics

- HW 1
 - Due: Mon 9/16 12pm (noon) EST
- Course web site: <https://www.cs.umb.edu/~stchang/cs450/f24>
 - Style: see “Racket Basics and Style” section



In-class exercise

- Using the `2http/image` library:
write a Racket expression that builds a “Traffic Light” image
- Put code in file named: `in-class-09-11-<Lastname>-<Firstname>.rkt`



Submitting In-class work

1. Join the in-class team at:
<https://github.com/orgs/cs450f24/teams/in-class>
2. Commit your file to this repo:
<https://github.com/cs450f24/in-class-09-11>
 - (May need to `merge` or `pull + rebase` if someone `pushes` before you)

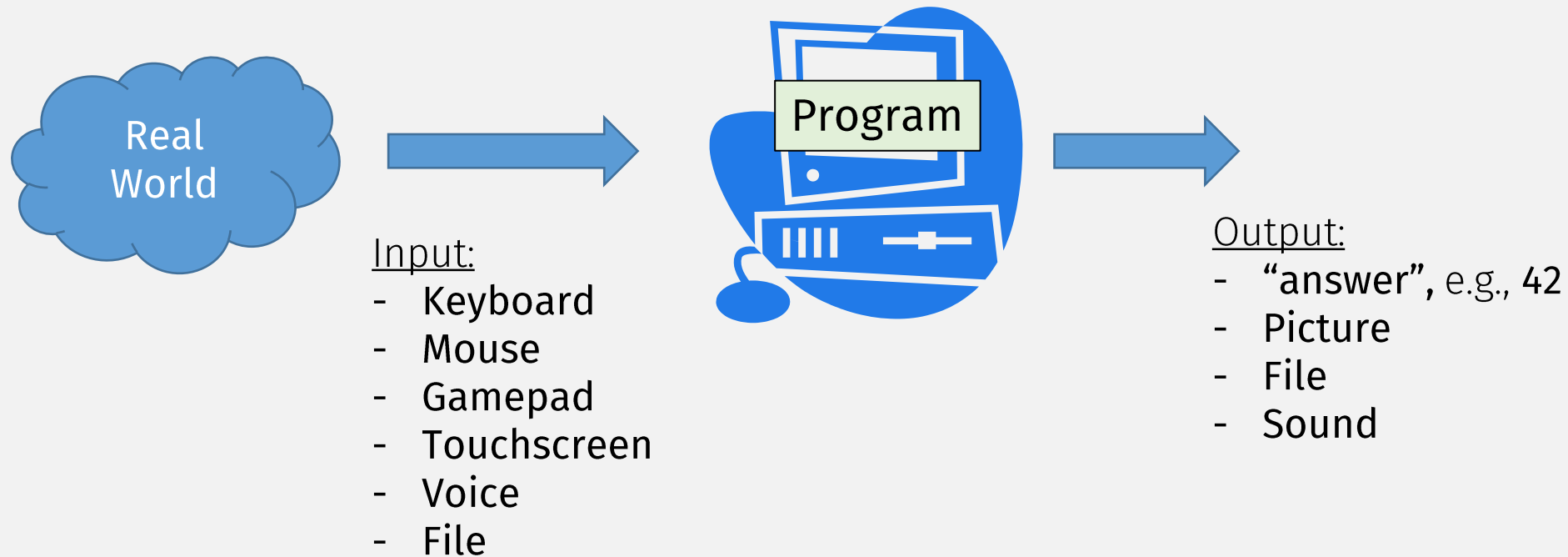
Functions

- `define` defines a function
 - The only non-expression you should use (for now)
 - `(define (fn-name arg1 arg2) ... body-expression ...)`
 - NOTE: `(define var-name expression)` defines a constant
- `lambda`
 - (anonymous) function expression
 - Function position in function call is just another expression!
 - `((lambda (x) (+ x 1)) 10)`
 - `((lambda (f) (f f)) (lambda (f) (f f)))`
- Predicates?
 - Function that evaluates to true or false

Programs

- Programs are sequence of defines and expressions
 - One of them could be a “main” entry point
- When the program is run, each is **evaluated** to get an “answer”
 - similar to “reduction” in math

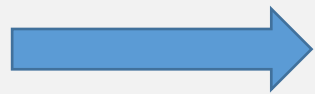
Programs: Still need I/O



Program vs Real World

Real World “things” ...

e.g., Temperature



Input:

- Keyboard
- Mouse
- Gamepad
- Touchscreen

... need a **data representation** in the program



A Data definition name

Specify possible values of the data

```
;; A TempC is an Integer  
;; Interpretation: It represents a temperature in degrees Celsius
```

“Interpretation” = Its connection to a real world concept

```
;; A TempF is an Integer  
;; Interp: It represents a temperature in degrees Fahrenheit
```

```
;; A TempK is an non-negative Integer  
;; Interp: It represents a temperature in degrees Kelvin
```

When programming, **choosing data representations** must be the first task!
(way before writing any code!!!)

Design Recipe

(Steps to follow when writing a program)

1. Data Design

- Define the needed **Data Definitions**
 - A **Data Definition** is a representation of a real world concept that is manipulated by the program
 - The **Interpretation** explains the connection to the real world

Design Recipe

1. Data Design
2. Function Design(s)

Designing Functions

```
;; A TempC is an Integer  
;; Interp: represents a temp in degrees Celsius  
;; A TempF is an Integer  
;; Interp: represents a temp in degrees Fahrenheit
```

1. Name

```
;; c2f: TempC -> TempF
```

2. Signature

```
;; Converts a Celsius temperature to Fahrenheit
```

- # of arguments and their data type
- Output type
- Use or create new Data Definitions (if needed)

3. Description

4. Examples

- Show how the function works

```
(check-equal? (c2f 0) 32)  
(check-equal? (c2f 100) 212)  
(check-equal? (c2f -40) -40)
```

5. Code

```
(define (c2f ctemp)  
  (+ (/ (* ctemp 9) 5) 32))
```

6. Tests

```
(check-equal? (c2f 1) (+ (/ 9 5) 32))
```

Designing Functions

```
;; A TempC is an Integer  
;; Interp: represents a temp in degrees Celsius  
;; A TempF is an Integer Rational  
;; Interp: represents a temp in degrees Fahrenheit
```

1. Name

```
;; c2f: TempC -> TempF
```

2. Signature

```
;; Converts a Celsius temperature to Fahrenheit
```

- # of arguments and their data type
- Output type
- Use or create new Data Definitions (if needed)

3. Description

4. Examples

- Show how the function works

```
(check-equal? (c2f 0) 32)  
(check-equal? (c2f 100) 212)  
(check-equal? (c2f -40) -40)
```

5. Code

```
(define (c2f ctemp)  
  (+ (/ (* ctemp 9) 5) 32))
```

6. Tests

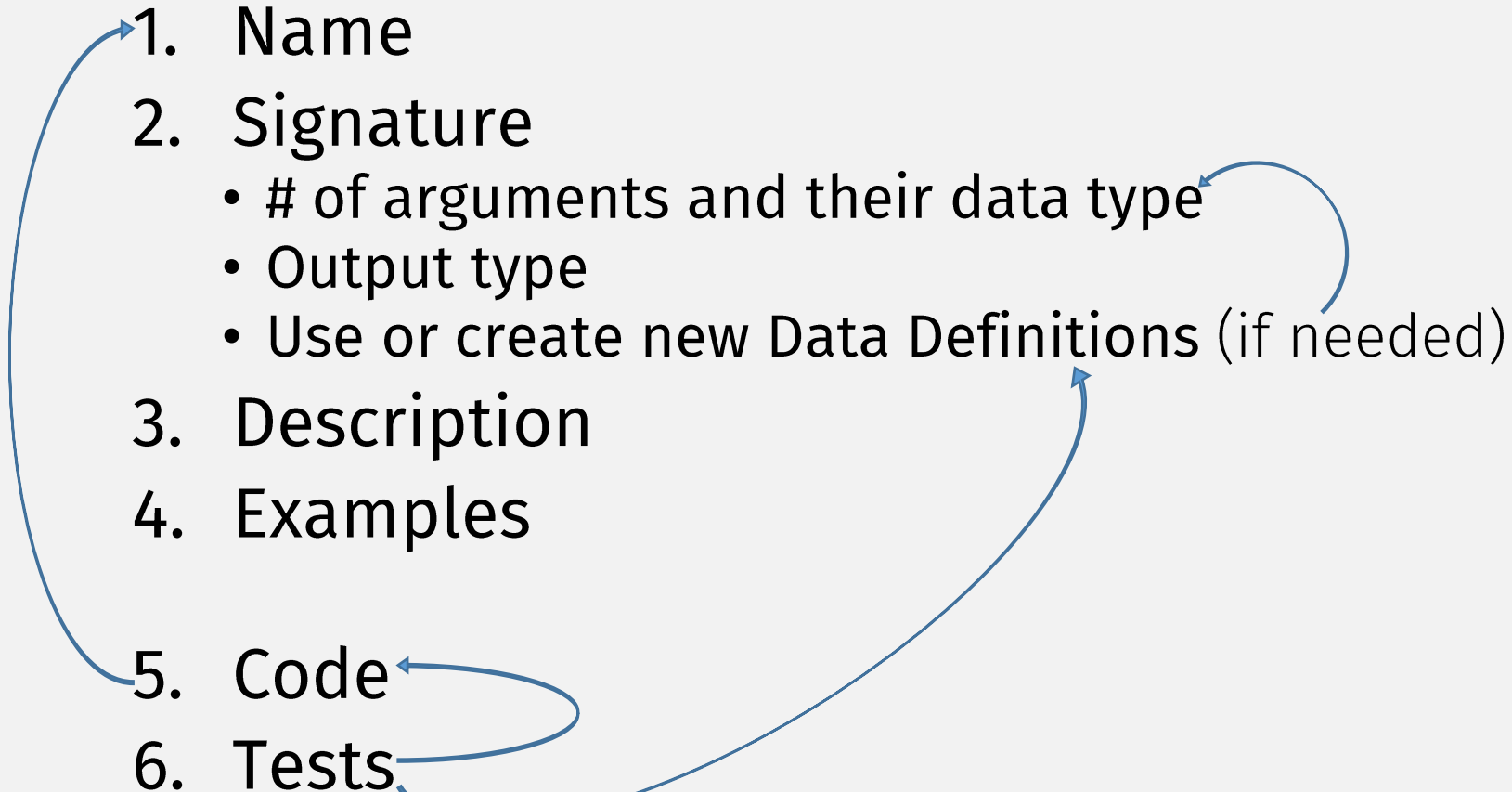
```
(check-equal? (c2f 1) (+ (/ 9 5) 32))
```

Something is wrong!

- Code?
- Signature?
- Data Definition?

Iterative Programming

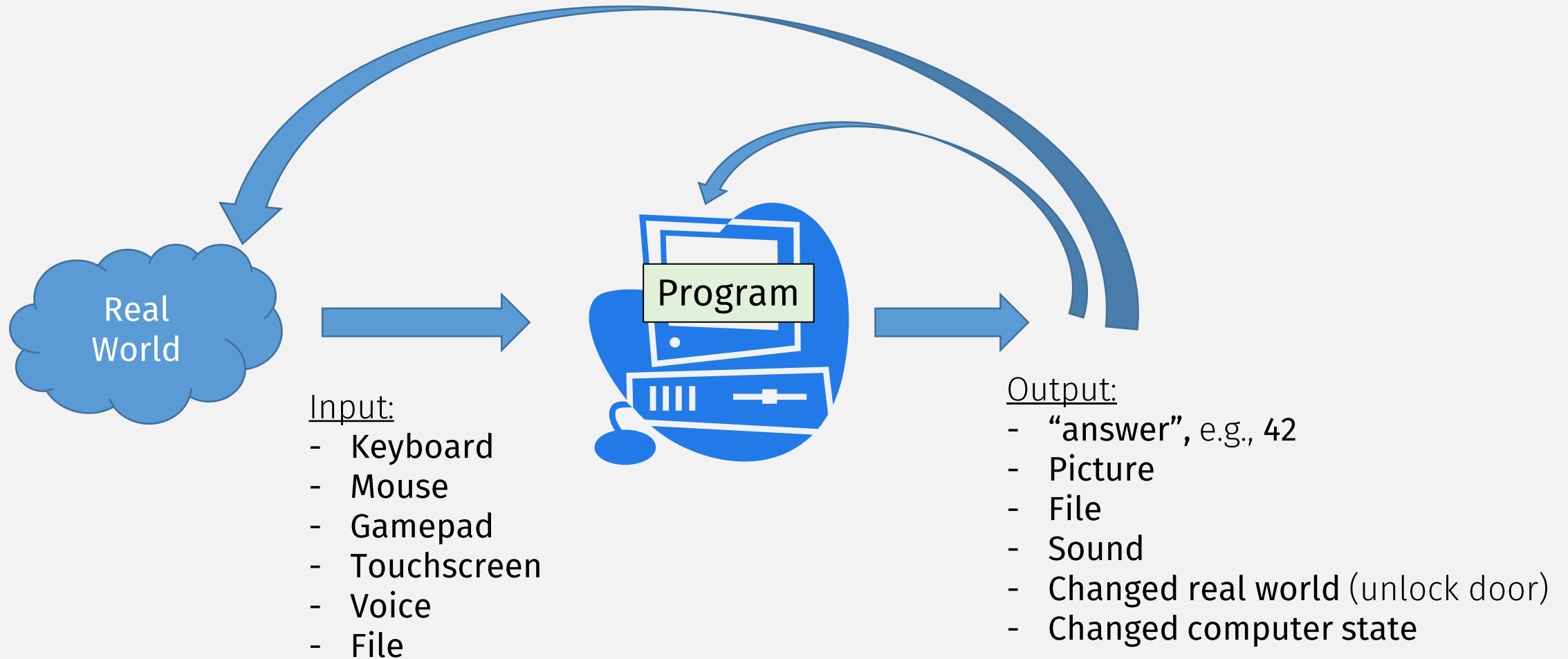
Other functions (“wish list”)

1. Name
 2. Signature
 - # of arguments and their data type
 - Output type
 - Use or create new Data Definitions (if needed)
 3. Description
 4. Examples
 5. Code
 6. Tests
- 

Programming is an
iterative process!

Programs: Interactive

Are more fun to write and use!



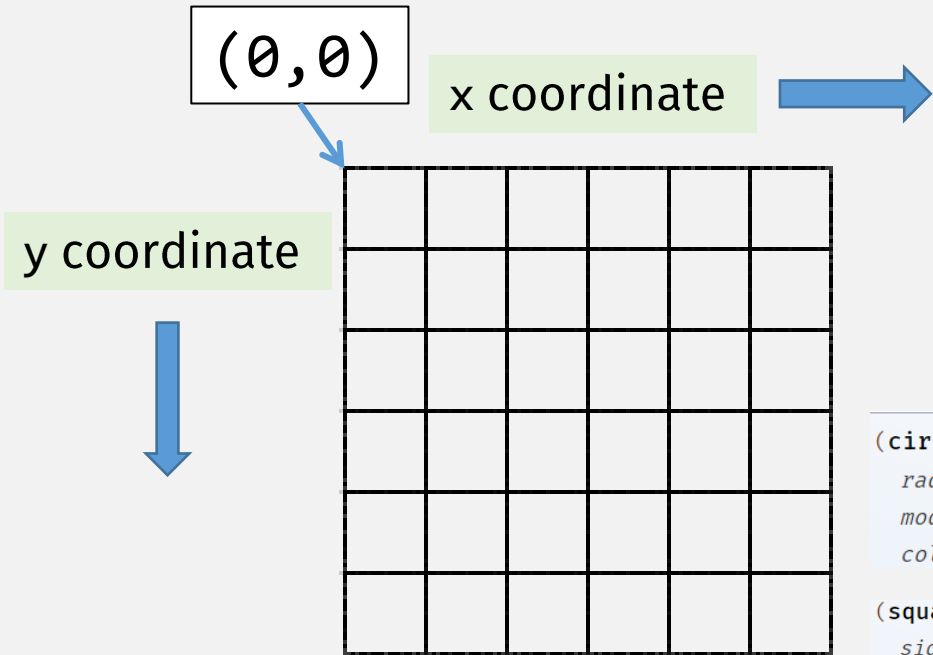
(require 2htdp/universe)

Interactive Programs (with **big-bang**)

- **big-bang** starts an (MVC-like) interactive loop
 - repeatedly updates a “world state”
 - Programmer must define what the “World” is ...
 - ... with a Data Definition!

```
;; A WorldState is a non-negative integer  
;; Interp: represents the y coordinate of a  
ball in a big-bang animation
```

Interlude: htdp universe coordinates



```
(place-image image x y scene) → image?
```

procedure

```
image : image?  
x : real?  
y : real?  
scene : image?
```

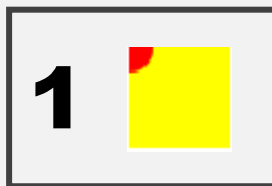
Places *image* onto *scene* with its center at the coordinates (x,y) and crops the resulting image so that it has the same size as *scene*. The coordinates are relative to the top-left of *scene*.

```
(circle radius mode color) → image?  
radius : (and/c real? (not/c negative?))  
mode : mode?  
color : image-color?
```

```
(square side-len mode color) → image?  
side-len : (and/c real? (not/c negative?))  
mode : mode?  
color : image-color?
```

```
(place-image  
  (circle 10 "solid" "red")  
  0 0  
  (square 40 "solid" "yellow"))
```

???



(require 2htdp/universe)

Interactive Programs (with **big-bang**)

- **big-bang** starts an (MVC-like) interactive loop
 - repeatedly updates a “world state”
 - Programmer must define what the “World” is ...
 - ... with a Data Definition!

```
;; A WorldState is a non-negative integer  
;; Interp: represents the y coordinate of a  
ball in an animation
```

- Programmers specify “handler” functions to manipulate “World”
 - Render
 - Input handlers
 - World update

Big-bang code demo