The Science of Problem Solving

Expressions and Data Types

The Nature of Functions

Aliens Attack Version 0

Making Decisions

Aliens Attack Version 1

Part I: The Basics of Problem Solving

Marco T. Morazán

Seton Hall University

Outline

1. The Science of Problem Solving
2. Expressions and Data Types
3. The Nature of Functions
4. Aliens Attack Version 0
5. Making Decisions
6. Aliens Attack Version 1

The Science of Problem Solving

- We all solve problems every day

The Science of Problem Solving

- We all solve problems every day
- Have you ever thought about how you go about problem solving?
- Do you randomly go about trying potential solutions to a problem or do you think about how to solve the problem?

The Science of Problem Solving

- We all solve problems every day
- Have you ever thought about how you go about problem solving?
- Do you randomly go about trying potential solutions to a problem or do you think about how to solve the problem?
- Most of the time you probably think about the problem to find a solution
- What steps do we take to arrive to a plausible solution?

The Science of Problem Solving

- We all solve problems every day
- Have you ever thought about how you go about problem solving?
- Do you randomly go about trying potential solutions to a problem or do you think about how to solve the problem?
- Most of the time you probably think about the problem to find a solution
- What steps do we take to arrive to a plausible solution?
- The solution is plausible until we test it and feel confident that it works

The Science of Problem Solving

- We all solve problems every day
- Have you ever thought about how you go about problem solving?
- Do you randomly go about trying potential solutions to a problem or do you think about how to solve the problem?
- Most of the time you probably think about the problem to find a solution
- What steps do we take to arrive to a plausible solution?
- The solution is plausible until we test it and feel confident that it works
- Understanding how to think about problems and solutions is where Computer Science and programming are beneficial to everyone.

The Science of Problem Solving

- Computer Science is not the study of computers just like Chemistry is not the study of test tubes nor Astronomy is the study of telescopes
- So, why is it called Computer Science?

The Science of Problem Solving

- Computer Science is not the study of computers just like Chemistry is not the study of test tubes nor Astronomy is the study of telescopes
- So, why is it called Computer Science?
- The best guess is that Computer Science is an umbrella term for many disciplines whose primary tool is the computer
- What do computer scientists do?

The Science of Problem Solving

- Computer Science is not the study of computers just like Chemistry is not the study of test tubes nor Astronomy is the study of telescopes
- So, why is it called Computer Science?
- The best guess is that Computer Science is an umbrella term for many disciplines whose primary tool is the computer
- What do computer scientists do?
- Computer scientists solve problems that are relevant to all facets of human life
- The best way to describe Computer Science is to say that it is the science of problem solving

The Science of Problem Solving

- Computer Science is not the study of computers just like Chemistry is not the study of test tubes nor Astronomy is the study of telescopes
- So, why is it called Computer Science?
- The best guess is that Computer Science is an umbrella term for many disciplines whose primary tool is the computer
- What do computer scientists do?
- Computer scientists solve problems that are relevant to all facets of human life
- The best way to describe Computer Science is to say that it is the science of problem solving
- Programming is how computer scientists express solutions to problems

The Science of Problem Solving

- Computer Science is not the study of computers just like Chemistry is not the study of test tubes nor Astronomy is the study of telescopes
- So, why is it called Computer Science?
- The best guess is that Computer Science is an umbrella term for many disciplines whose primary tool is the computer
- What do computer scientists do?
- Computer scientists solve problems that are relevant to all facets of human life
- The best way to describe Computer Science is to say that it is the science of problem solving
- Programming is how computer scientists express solutions to problems
- Problem solving is an *essential* skill just like reading, writing, and doing arithmetic

The Science of Problem Solving

- Two of the best-known problem solving techniques are *divide and conquer* and *iterative refinement*

The Science of Problem Solving

- Two of the best-known problem solving techniques are *divide and conquer* and *iterative refinement*
- Problem solvers use divide and conquer when a larger problem is decomposable into smaller problems that are solved independently and then their solutions are combined

The Science of Problem Solving

- Two of the best-known problem solving techniques are *divide and conquer* and *iterative refinement*
- Problem solvers use divide and conquer when a larger problem is decomposable into smaller problems that are solved independently and then their solutions are combined
- Iterative refinement is used to develop a solution in steps
- Useful to manage the complexity of a problems
- Solve a simpler version of the problem, add complexity to the problem, and re-solve it until you have a full answer

The Science of Problem Solving

- Two of the best-known problem solving techniques are *divide and conquer* and *iterative refinement*
- Problem solvers use divide and conquer when a larger problem is decomposable into smaller problems that are solved independently and then their solutions are combined
- Iterative refinement is used to develop a solution in steps
- Useful to manage the complexity of a problems
- Solve a simpler version of the problem, add complexity to the problem, and re-solve it until you have a full answer
- You have been taught to compute solutions most of your life like the meaning from English expressions
- You know that *Thelma is that pig* has a different meaning from *That is Thelma's pig*

The Science of Problem Solving

- Two of the best-known problem solving techniques are *divide and conquer* and *iterative refinement*
- Problem solvers use divide and conquer when a larger problem is decomposable into smaller problems that are solved independently and then their solutions are combined
- Iterative refinement is used to develop a solution in steps
- Useful to manage the complexity of a problems
- Solve a simpler version of the problem, add complexity to the problem, and re-solve it until you have a full answer
- You have been taught to compute solutions most of your life like the meaning from English expressions
- You know that *Thelma is that pig* has a different meaning from *That is Thelma's pig*
- Other languages that you have been trained to do computations with are Arithmetic and Algebra
- You know how to compute the value of the following expression: `5 * (8 + 2)`.

The Science of Problem Solving

- Two of the best-known problem solving techniques are *divide and conquer* and *iterative refinement*
- Problem solvers use divide and conquer when a larger problem is decomposable into smaller problems that are solved independently and then their solutions are combined
- Iterative refinement is used to develop a solution in steps
- Useful to manage the complexity of a problems
- Solve a simpler version of the problem, add complexity to the problem, and re-solve it until you have a full answer
- You have been taught to compute solutions most of your life like the meaning from English expressions
- You know that *Thelma is that pig* has a different meaning from *That is Thelma's pig*
- Other languages that you have been trained to do computations with are Arithmetic and Algebra
- You know how to compute the value of the following expression: `5 * (8 + 2)`.
- We use expressions to describe computations and we evaluate expressions to derive meaning

The Science of Problem Solving

- Two of the best-known problem solving techniques are *divide and conquer* and *iterative refinement*
- Problem solvers use divide and conquer when a larger problem is decomposable into smaller problems that are solved independently and then their solutions are combined
- Iterative refinement is used to develop a solution in steps
- Useful to manage the complexity of a problems
- Solve a simpler version of the problem, add complexity to the problem, and re-solve it until you have a full answer
- You have been taught to compute solutions most of your life like the meaning from English expressions
- You know that *Thelma is that pig* has a different meaning from *That is Thelma's pig*
- Other languages that you have been trained to do computations with are Arithmetic and Algebra
- You know how to compute the value of the following expression: `5 * (8 + 2)`.
- We use expressions to describe computations and we evaluate expressions to derive meaning
- You may have never thought explicitly about divide an conquer or about iterative refinement, but you have been taught to use these techniques
- Writing an essay: first create an outline (divide and conquer)
- Every point is implemented as a different section
- You first write a rough draft and then repeatedly make improvements until you are happy with the result (iterative refinement)

The Science of Problem Solving

- If we are going to express solutions to problems using a computer, we need to use a *programming language* much like we use English to communicate

The Science of Problem Solving

- If we are going to express solutions to problems using a computer, we need to use a *programming language* much like we use English to communicate
- A programming language allows us to communicate to the computer what we want it to do
- We start with a programming language called *Beginning Student Language* (BSL)
- BSL is a programming language specifically designed to teach beginners like yourself how to design and implement solutions to problems
- A solution to a problem written in BSL (or any other programming language) is called a *program*
- When a program is evaluated, we obtain its meaning (the solution to an instance of a problem)

The Science of Problem Solving

Getting Started

- We need a programming development environment
- Download and install:

  `https://download.racket-lang.org/`

The Science of Problem Solving

Getting Started

- We need a programming development environment
- Download and install:

  `https://download.racket-lang.org/`

- After installing `DrRacket` run it and to go to the `Language` menu. Click on `Choose Language...`:

- Click on `Beginning Student`
- Click on `OK`

The Science of Problem Solving

Getting Started

- Click on the `RUN` button towards the top right corner of `DrRacket`:

- Top half: *definitions area* for programming
- Bottom half: *interactions area* for interacting

The Science of Problem Solving

Getting Started

- Type the following string in the definitions area:

```
"Hello World! I am DrRacket."
```

- Click on `RUN`
- In the interactions window you will see the string printed.
- Congratulations! You just wrote your first program

The Science of Problem Solving

Getting Started

- Type the following string in the definitions area:

```
"Hello World! I am DrRacket."
```

- Click on `RUN`
- In the interactions window you will see the string printed.
- Congratulations! You just wrote your first program
- When you clicked on the `RUN` button, you told `DrRacket` to evaluate your program
- The value of a string is just the string itself
- Once `DrRacket` knows the value of a program it is printed in the interactions area

The Science of Problem Solving

HOMEWORK

- Problems: 1–3

The Science of Problem Solving

Getting Started

- So, what is a program?

The Science of Problem Solving

Getting Started

- So, what is a program?
- A program is similar to an essay
- You express the solution to a problem using *expressions*
- In English, your expressions are words that may be used to construct larger expressions like sentences and paragraphs
- In programming, your basic expressions are values (for now), like strings and numbers, that may be used to create larger useful expressions

The Science of Problem Solving

Getting Started

- We need to know how to write valid expressions
- A *grammar* describes expressions
- A grammar consists of a series of production rules for typing valid expressions

The Science of Problem Solving

Getting Started

- We need to know how to write valid expressions
- A *grammar* describes expressions
- A grammar consists of a series of production rules for typing valid expressions

```
        expr   ::=  string
               ::=  number
      string   ::=  "<character>*"
      number   ::=  positive
               ::=  negative
    positive   ::=  mag
    negative   ::=  -mag
         mag   ::=  int
               ::=  real
               ::=  fraction
         int   ::=  digit+
       digit   ::=  0|1|2|3|4|5|6|7|8|9
        real   ::=  digit*.digit+
    fraction   ::=  int/nonzero-int
nonzero-int    ::=  <1|2|3|4|5|6|7|8|9>int
     program   ::=  expr*
```

The Science of Problem Solving

Getting Started

- To write a new program open a new tab in `DrRacket` by using `Ctrl-T`
- Now write and run the following program:

```
"Program By Design"
1000000
-8.7
```

The Science of Problem Solving

Getting Started

- To write a new program open a new tab in `DrRacket` by using `Ctrl-T`
- Now write and run the following program:

```
"Program By Design"
1000000
-8.7
```

- The result in the interactions area after clicking `RUN` is:

```
"Program by Design"
1000000
-8.7
```

The Science of Problem Solving

HOMEWORK

- Problems: 4–5

The Science of Problem Solving

Computing New Values

- BSL provides us with application expressions
- An application expression applies a function to one or more arguments
- An application expression evaluates to the value returned by a function

The Science of Problem Solving

Computing New Values

- BSL provides us with application expressions
- An application expression applies a function to one or more arguments
- An application expression evaluates to the value returned by a function
- The `expr`'s production rules include:

  `expr` ::= (<function> `expr`$^+$)

The Science of Problem Solving

Computing New Values

- BSL provides us with application expressions
- An application expression applies a function to one or more arguments
- An application expression evaluates to the value returned by a function
- The `expr`'s production rules include:

  `expr` ::= (<function> `expr`$^+$)

Function	Sample Application Expression	Value
+	(+ 1 2 3 4 5)	15
-	(- 5 8 2)	-5
*	(* 39 45 29 2)	101790
/	(/ 20 4 2)	2.5
expt	(expt 64 1/2)	8
string-length	(string-length "Program By Design")	17
string-append	(string-append "Hello " "World")	"Hello World"

The Science of Problem Solving

Computing New Values

- Application expressions may be nested

```
(+ 10 (* 4 5))
```

The Science of Problem Solving

Computing New Values

- Application expressions may be nested

```
(+ 10 (* 4 5))
```

- `(* (+ 4 1) 7 (- 15 5))` `(* (+ 4 1 7) (- 15 5))`
- Do all functions take as input an arbitrary number of expressions?

The Science of Problem Solving

Computing New Values

- Application expressions may be nested

```
(+ 10 (* 4 5))
```

- `(* (+ 4 1) 7 (- 15 5))` `(* (+ 4 1 7) (- 15 5))`
- Do all functions take as input an arbitrary number of expressions?
- No

The Science of Problem Solving

Computing New Values

- Application expressions may be nested

```
(+ 10 (* 4 5))
```

- `(* (+ 4 1) 7 (- 15 5))` `(* (+ 4 1 7) (- 15 5))`
- Do all functions take as input an arbitrary number of expressions?
- No
- If you are not sure if a function accepts an arbitrary number of arguments, you have two options:
  - Try it
  - Consult `DrRacket`'s `Help Desk`

The Science of Problem Solving

Computing New Values

- Application expressions may be nested

```
(+ 10 (* 4 5))
```

- `(* (+ 4 1) 7 (- 15 5))` `(* (+ 4 1 7) (- 15 5))`
- Do all functions take as input an arbitrary number of expressions?
- No
- If you are not sure if a function accepts an arbitrary number of arguments, you have two options:
  - Try it
  - Consult DrRacket's Help Desk

The Science of Problem Solving

Computing New Values

- Search for `string-append` yields:

```
(string-append s t z ...) → string

  s : string
  t : string
  z : string

Concatenates the characters of several strings.

  > (string-append "hello" " " "world" " " "good bye")
  "hello world good bye"
```

The Science of Problem Solving

Definitions and Interactions Areas Differences

- Expressions are written in both the definitions and the interactions areas
- In the definitions area you need to click `RUN`
- In the interactions area you need to hit `Enter`
- Why are there two areas?

The Science of Problem Solving

Definitions and Interactions Areas Differences

- Expressions are written in both the definitions and the interactions areas
- In the definitions area you need to click `RUN`
- In the interactions area you need to hit `Enter`
- Why are there two areas?
- The two areas are used for different purposes
- In the definitions area we write programs
- In the interactions area, we ask `DrRacket` to evaluate one-time expressions

The Science of Problem Solving

Saving Your Work

- It is important that you regularly save your work
- Create a directory for the programs you will develop as you read this book to keep your files organized.

The Science of Problem Solving

Saving Your Work

- It is important that you regularly save your work
- Create a directory for the programs you will develop as you read this book to keep your files organized.
- To save your work go the `File` menu
- If it is the first time saving your current program click on `Save Definitions As...`

- Find the directory you wish to save your work in and enter a filename

The Science of Problem Solving

Saving Your Work

- It is important that you regularly save your work
- Create a directory for the programs you will develop as you read this book to keep your files organized.
- To save your work go the `File` menu
- If it is the first time saving your current program click on `Save Definitions As...`

- Find the directory you wish to save your work in and enter a filename
- If it is not the first time saving the current program, click on `Save Definitions` inside the `File` menu
- Alternatively, you may simply use `Ctrl+S`

The Science of Problem Solving

Error Messages

- Errors are part of the problem solving process
- Do not panic when you get an error message
- An error message is `DrRacket`'s attempt to help you diagnose the error.

The Science of Problem Solving

Error Messages

- Errors are part of the problem solving process
- Do not panic when you get an error message
- An error message is `DrRacket`'s attempt to help you diagnose the error.
- Error messages help us diagnose and correct *bugs*
- An error message itself does not diagnose the bug nor does it tell us how to fix the bug

The Science of Problem Solving

Grammatical Errors

- The BSL grammar tells you how to write valid BSL programs
- If you fail to write a valid BSL program and click on RUN, DrRacket prints an error message

The Science of Problem Solving

Grammatical Errors

- The BSL grammar tells you how to write valid BSL programs
- If you fail to write a valid BSL program and click on RUN, DrRacket prints an error message
- Type the following in the definitions area and click run:

```
"Hello World! I am DrRacket.
```

The Science of Problem Solving

Grammatical Errors

- The BSL grammar tells you how to write valid BSL programs
- If you fail to write a valid BSL program and click on RUN, DrRacket prints an error message
- Type the following in the definitions area and click run:

```
"Hello World! I am DrRacket.
```

- The following error message appears in the interactions window:

```
read-syntax: expected a closing '"'
```

The Science of Problem Solving

Grammatical Errors

- The BSL grammar tells you how to write valid BSL programs
- If you fail to write a valid BSL program and click on RUN, DrRacket prints an error message
- Type the following in the definitions area and click run:

```
"Hello World! I am DrRacket.
```

- The following error message appears in the interactions window:

```
read-syntax: expected a closing '"'
```

- Grammatical errors may also occur writing application expressions
- Type and run the following program:

```
(sting-append "I'm done" " " "and going home!")
```

The Science of Problem Solving

Grammatical Errors

- The BSL grammar tells you how to write valid BSL programs
- If you fail to write a valid BSL program and click on RUN, DrRacket prints an error message
- Type the following in the definitions area and click run:

```
"Hello World! I am DrRacket.
```

- The following error message appears in the interactions window:

```
read-syntax: expected a closing '"'
```

- Grammatical errors may also occur writing application expressions
- Type and run the following program:

```
(sting-append "I'm done" " " "and going home!")
```

- The following error message is displayed:

```
sting-append: this function is not defined
```

The Science of Problem Solving

Grammatical Errors

- The BSL grammar tells you how to write valid BSL programs
- If you fail to write a valid BSL program and click on RUN, DrRacket prints an error message
- Type the following in the definitions area and click run:

```
"Hello World! I am DrRacket.
```

- The following error message appears in the interactions window:

```
read-syntax: expected a closing '"'
```

- Grammatical errors may also occur writing application expressions
- Type and run the following program:

```
(sting-append "I'm done" " " "and going home!")
```

- The following error message is displayed:

```
sting-append: this function is not defined
```

- Correct it as follows:

```
(string-append "I'm done" " " "and going home!")
```

The Science of Problem Solving

Grammatical Errors

- Failing to properly write balanced parentheses: `(+ (* 4 5 (/ 50 10))`:

The Science of Problem Solving

Grammatical Errors

- Failing to properly write balanced parentheses: `(+ (* 4 5 (/ 50 10))`:
- After clicking `RUN`, the following error message is displayed:

  `read-syntax: expected a ')' to close '('`
- The opening parenthesis before + is highlighted :

The Science of Problem Solving

Grammatical Errors

- Failing to properly write balanced parentheses: `(+ (* 4 5 (/ 50 10))`:
- After clicking `RUN`, the following error message is displayed:

  `read-syntax: expected a ')' to close '('`
- The opening parenthesis before + is highlighted :

- Fixed by adding a closing parenthesis at the end? `(+ (* 4 5 (/ 50 10)))`

The Science of Problem Solving

Grammatical Errors

- Failing to properly write balanced parentheses: `(+ (* 4 5 (/ 50 10))`:
- After clicking `RUN`, the following error message is displayed:

  `read-syntax: expected a ')' to close '('`

- The opening parenthesis before + is highlighted :

- Fixed by adding a closing parenthesis at the end? `(+ (* 4 5 (/ 50 10)))`
- After clicking `RUN`, the following error message is displayed:

  `+: expects at least 2 arguments, but found only 1`

The Science of Problem Solving

Grammatical Errors

- Failing to properly write balanced parentheses: `(+ (* 4 5 (/ 50 10))`:
- After clicking `RUN`, the following error message is displayed:

  `read-syntax: expected a ')' to close '('`
- The opening parenthesis before + is highlighted :

- Fixed by adding a closing parenthesis at the end? `(+ (* 4 5 (/ 50 10)))`
- After clicking `RUN`, the following error message is displayed:

  `+: expects at least 2 arguments, but found only 1`
- This is a type error (discussed later)
- We placed the missing closing parenthesis is the wrong place
- We fix the problem as follows: `(+ (* 4 5) (/ 50 10))`

The Science of Problem Solving

Grammatical Errors

- Another common grammatical mistake is to place an expression inside parentheses when a function is not applied.
- Type after the `DrRacket`'s prompt:

```
(* (10) (30))
```

The Science of Problem Solving

Grammatical Errors

- Another common grammatical mistake is to place an expression inside parentheses when a function is not applied.
- Type after the `DrRacket`'s prompt:

```
(* (10) (30))
```

The Science of Problem Solving

HOMEWORK

- Problem: 6

The Science of Problem Solving

Type Errors

- Type errors mostly occur in two ways
- The first is when the wrong number of arguments are provided to a function (just seen before)
- The second is when the wrong type of argument is given to a function

The Science of Problem Solving

Type Errors

- Type errors mostly occur in two ways
- The first is when the wrong number of arguments are provided to a function (just seen before)
- The second is when the wrong type of argument is given to a function
- `(string-append "My lucky number is: " 8)`

The Science of Problem Solving

Type Errors

- Type errors mostly occur in two ways
- The first is when the wrong number of arguments are provided to a function (just seen before)
- The second is when the wrong type of argument is given to a function
- `(string-append "My lucky number is: " 8)`
- Error message:

  `string-append: expects a string as 2nd argument, given 8`

The Science of Problem Solving

Type Errors

- Type errors mostly occur in two ways
- The first is when the wrong number of arguments are provided to a function (just seen before)
- The second is when the wrong type of argument is given to a function
- `(string-append "My lucky number is: " 8)`
- Error message:

  `string-append: expects a string as 2nd argument, given 8`
- Fix by making the second argument a string:

  `(string-append "My lucky number is: " "8")`

The Science of Problem Solving

Type Errors

- Type errors mostly occur in two ways
- The first is when the wrong number of arguments are provided to a function (just seen before)
- The second is when the wrong type of argument is given to a function
- `(string-append "My lucky number is: " 8)`
- Error message:

  `string-append: expects a string as 2nd argument, given 8`
- Fix by making the second argument a string:

  `(string-append "My lucky number is: " "8")`
- Providing a function with the wrong type of input is a very common mistake
- *type theory* is an important and growing discipline within Computer Science
- We will learn design techniques to help us avoid type errors

The Science of Problem Solving

HOMEWORK

- Problem 7

The Science of Problem Solving

Runtime Errors

- Third kind of error: runtime error
- Runtime errors are not detected until after an evaluation has started

The Science of Problem Solving

Runtime Errors

- Third kind of error: runtime error
- Runtime errors are not detected until after an evaluation has started
- Type the following program in the definitions area:

```
(* 5 0)   (+ 5 0)   (/ 5 0)   (- 5 0)
```

The Science of Problem Solving

Runtime Errors

- Third kind of error: runtime error
- Runtime errors are not detected until after an evaluation has started
- Type the following program in the definitions area:

```
(* 5 0)   (+ 5 0)   (/ 5 0)   (- 5 0)
```

The Science of Problem Solving

Runtime Errors

- Third kind of error: runtime error
- Runtime errors are not detected until after an evaluation has started
- Type the following program in the definitions area:

```
(* 5 0)    (+ 5 0)    (/ 5 0)    (- 5 0)
```

- Programs may contain runtime errors that do not generate an error message
- Write to compute `f(4)`, where `f(x)` = $2x^2$ + `10x + 3`:

```
(+ (* 2 (sqr 4)) (* 10 4) 4)
```

The Science of Problem Solving

Runtime Errors

- Third kind of error: runtime error
- Runtime errors are not detected until after an evaluation has started
- Type the following program in the definitions area:

```
(* 5 0)    (+ 5 0)    (/ 5 0)    (- 5 0)
```

- Programs may contain runtime errors that do not generate an error message
- Write to compute `f(4)`, where `f(x)` = $2x^2$ + `10x + 3`:

```
(+ (* 2 (sqr 4)) (* 10 4) 4)
```

- 76 is printed in the interactions window
- 76 is not the value of `f(4)`
- Can you fix it?

The Science of Problem Solving

Runtime Errors

- To help protect ourselves against this type of error: write *unit tests*
- Validates that two different expressions evaluate to the same value

The Science of Problem Solving

Runtime Errors

- To help protect ourselves against this type of error: write *unit tests*
- Validates that two different expressions evaluate to the same value
- Expand the grammar as follows:

```
program   ::=   {expr | test}*
   test   ::=   (check-expect expr expr)
```

- program is 0 or more expressions or tests
- Test: expressions for *actual value* and *expected value*

The Science of Problem Solving

Runtime Errors

- To help protect ourselves against this type of error: write *unit tests*
- Validates that two different expressions evaluate to the same value
- Expand the grammar as follows:

```
program   ::=   {expr | test}*
   test   ::=   (check-expect expr expr)
```

- program is 0 or more expressions or tests
- Test: expressions for *actual value* and *expected value*
- We can now rewrite our buggy program as follows:

```
(+ (* 2 (sqr 4)) (* 10 4) 4)
(check-expect (+ (* 2 (sqr 4)) (* 10 4) 4) 75)
```

The Science of Problem Solving

Runtime Errors

- To help protect ourselves against this type of error: write *unit tests*
- Validates that two different expressions evaluate to the same value
- Expand the grammar as follows:

```
program   ::=   {expr | test}*
   test   ::=   (check-expect expr expr)
```

- program is 0 or more expressions or tests
- Test: expressions for *actual value* and *expected value*
- We can now rewrite our buggy program as follows:

```
(+ (* 2 (sqr 4)) (* 10 4) 4)
(check-expect (+ (* 2 (sqr 4)) (* 10 4) 4) 75)
```

The Science of Problem Solving

Runtime Errors

- Correct the program as follows:

```
(+ (* 2 (sqr 4)) (* 10 4) 3)
(check-expect (+ (* 2 (sqr 4)) (* 10 4) 3) 75)
```

The Science of Problem Solving

Runtime Errors

- Correct the program as follows:

```
(+ (* 2 (sqr 4)) (* 10 4) 3)
(check-expect (+ (* 2 (sqr 4)) (* 10 4) 3) 75)
```

The Science of Problem Solving

HOMEWORK

- Problems: 8–9

Expressions and Data Types

- Need a richer set of data types to make problem solving easier
- These are divided into two broad categories: *primitive* and *compound*
- Primitive data type is any type of data that is indivisible like numbers
- Compound data type is one that contains multiple pieces of data like a coordinate on the two-dimensional Cartesian plane

Expressions and Data Types

- Need a richer set of data types to make problem solving easier
- These are divided into two broad categories: *primitive* and *compound*
- Primitive data type is any type of data that is indivisible like numbers
- Compound data type is one that contains multiple pieces of data like a coordinate on the two-dimensional Cartesian plane
- New data types: Boolean, symbol, character, and image

Expressions and Data Types

(a) Rectangle 1.

(b) Rectangle 2.

(c) Rectangle 3.

-

- Consider computing the area and the perimeter of the rectangles
- Area is computed by multiplying the width and the length
- Perimeter is computed by adding the doubling of the width and the doubling of the length

Expressions and Data Types

```
;; Area for rectangle 1
"The area for rectangle 1 is: "
(* 3 4)
(check-expect (* 3 4) 12)
;; Perimeter for rectangle 1
"The perimeter for rectangle 1 is: "
(+ (* 2 3) (* 2 4))
(check-expect (+ (* 2 3) (* 2 4)) 14)
```

Expressions and Data Types

```
;; Area for rectangle 1
"The area for rectangle 1 is: "
(* 3 4)
(check-expect (* 3 4) 12)
;; Perimeter for rectangle 1
"The perimeter for rectangle 1 is: "
(+ (* 2 3) (* 2 4))
(check-expect (+ (* 2 3) (* 2 4)) 14)
```

```
;; Area for rectangle 2
"The area for rectangle 2 is: "
(* 2 5)
(check-expect (* 2 5) 10)
;; Perimeter for rectangle 2
"The perimeter for rectangle 2 is: "
(+ (* 2 2) (* 2 5))
(check-expect (+ (* 2 2) (* 2 5)) 14)
```

Expressions and Data Types

```
;; Area for rectangle 1
"The area for rectangle 1 is: "
(* 3 4)
(check-expect (* 3 4) 12)
;; Perimeter for rectangle 1
"The perimeter for rectangle 1 is: "
(+ (* 2 3) (* 2 4))
(check-expect (+ (* 2 3) (* 2 4)) 14)
```

```
;; Area for rectangle 2
"The area for rectangle 2 is: "
(* 2 5)
(check-expect (* 2 5) 10)
;; Perimeter for rectangle 2
"The perimeter for rectangle 2 is: "
(+ (* 2 2) (* 2 5))
(check-expect (+ (* 2 2) (* 2 5)) 14)
```

```
;; Area for rectangle 3
"The area for rectangle 3 is: "
(* 2 2)
(check-expect (* 2 2) 4)
;; Perimeter for rectangle 3
"The perimeter for rectangle 3 is: "
(+ (* 2 2) (* 2 2))
(check-expect (+ (* 2 2) (* 2 2)) 8)
```

Expressions and Data Types

Definitions

- Writing the same expression multiple times is error-prone
- It is better to define a variable for the value of an expression that is needed multiple times

Expressions and Data Types

Definitions

- Writing the same expression multiple times is error-prone
- It is better to define a variable for the value of an expression that is needed multiple times
- To define and use variables the BSL syntax is extended as follows:

```
program  ::=  {expr | test | def}*
    def  ::=  (define <variable name> expr)
   expr  ::=  <variable name>
```

- A program consists of 0 or more expressions, tests, and definitions
- An expression may be a variable name

Expressions and Data Types

Definitions

- Writing the same expression multiple times is error-prone
- It is better to define a variable for the value of an expression that is needed multiple times
- To define and use variables the BSL syntax is extended as follows:

```
program  ::=  {expr | test | def}*
    def  ::=  (define <variable name> expr)
   expr  ::=  <variable name>
```

- A program consists of 0 or more expressions, tests, and definitions
- An expression may be a variable name
- A definition binds the variable to the value of the expression::

```
(define DELTA-X 5)  Binds DELTA-X to 5

(define i (* 4 25)) Binds i to 100
```

- After clicking RUN, DrRacket knows about all the definitions

Expressions and Data Types

Definitions

- Writing the same expression multiple times is error-prone
- It is better to define a variable for the value of an expression that is needed multiple times
- To define and use variables the BSL syntax is extended as follows:

```
program  ::=  {expr | test | def}*
    def  ::=  (define <variable name> expr)
   expr  ::=  <variable name>
```

- A program consists of 0 or more expressions, tests, and definitions
- An expression may be a variable name
- A definition binds the variable to the value of the expression::

```
(define DELTA-X 5)  Binds DELTA-X to 5

(define i (* 4 25)) Binds i to 100
```

- After clicking RUN, DrRacket knows about all the definitions
- Variable definitions is how typing and evaluating the same expression more than once is avoided
- Use uppercase letters for constants and lowercase case letters for variables
- DELTA-X is a constant and i is not considered a constant
- In BSL, there is no such convention and it is up to the programmers to use this convention

Expressions and Data Types

Definitions

```
;; Area for rectangle 1
(define AREA1 (* 3 4))
(check-expect AREA1 12)

;; Perimeter for rectangle 1
(define PERIM1 (+ (* 2 3) (* 2 4)))
(check-expect PERIM1 14)
```

Expressions and Data Types

Definitions

- ```
  ;; Area for rectangle 1
  (define AREA1 (* 3 4))
  (check-expect AREA1 12)

  ;; Perimeter for rectangle 1
  (define PERIM1 (+ (* 2 3) (* 2 4)))
  (check-expect PERIM1 14)
  ```
- ```
  ;; Area for rectangle 2
  (define AREA2 (* 2 5))
  (check-expect AREA2 10)

  ;; Perimeter for rectangle 2
  (define PERIM2 (+ (* 2 2) (* 2 5)))
  (check-expect PERIM2 14)
  ```

Expressions and Data Types

Definitions

```
;; Area for rectangle 1
(define AREA1 (* 3 4))
(check-expect AREA1 12)

;; Perimeter for rectangle 1
(define PERIM1 (+ (* 2 3) (* 2 4)))
(check-expect PERIM1 14)
```

```
;; Area for rectangle 2
(define AREA2 (* 2 5))
(check-expect AREA2 10)

;; Perimeter for rectangle 2
(define PERIM2 (+ (* 2 2) (* 2 5)))
(check-expect PERIM2 14)
```

```
;; Area for rectangle 3
(define AREA3 (* 2 2))
(check-expect AREA3 4)

;; Perimeter for rectangle 3
(define PERIM3 (+ (* 2 2) (* 2 2)))
(check-expect PERIM3 8)
```

Expressions and Data Types

Numbers

- A number is either an integer, a real, or a fraction
- Number is a primitive type (of data) in BSL

Expressions and Data Types

Numbers

- A number is either an integer, a real, or a fraction
- Number is a primitive type (of data) in BSL
- Write a program for the circumference of a circle with a radius of 7

Expressions and Data Types

Numbers

- A number is either an integer, a real, or a fraction
- Number is a primitive type (of data) in BSL
- Write a program for the circumference of a circle with a radius of 7
-

```
;; The circumference of a circle with radius r is 2 * pi * r

;; The circumference of a circle with radius 7
(define CIRCUM-7 (* 2 pi 7))

(check-expect CIRCUM-7 43.98)
```

- In BSL, pi stores the value of π
- Run the program

Expressions and Data Types

Numbers

- A number is either an integer, a real, or a fraction
- Number is a primitive type (of data) in BSL
- Write a program for the circumference of a circle with a radius of 7
-

```
;; The circumference of a circle with radius r is 2 * pi * r

;; The circumference of a circle with radius 7
(define CIRCUM-7 (* 2 pi 7))

(check-expect CIRCUM-7 43.98)
```

- In BSL, pi stores the value of π
- Run the program
-

Expressions and Data Types

Numbers

- A real number (called `inexact number` in BSL) cannot be checked using `check-expect`
- The reason is that numbers in computer are not exactly the same as numbers in Mathematics
- A real number is limited to a finite number, `n`, of decimal places
- Computations using real numbers in BSL may contain errors

Expressions and Data Types

Numbers

- A real number (called `inexact number` in BSL) cannot be checked using `check-expect`
- The reason is that numbers in computer are not exactly the same as numbers in Mathematics
- A real number is limited to a finite number, `n`, of decimal places
- Computations using real numbers in BSL may contain errors
- `DrRacket` is nice enough to print the value prefixed with `#i` to warn that it is inexact and, therefore, may contain an error
- When the magnitude of a real number is very small or very large `DrRacket` prints it using scientific notation
- `#i4.3290784900439874e-010` corresponds to the inexact number `#i4.3290784900439874 x` 10^{-10}.

Expressions and Data Types

Numbers

- To test computations involving real numbers, the grammar production rules for `test` must be expanded as follows:

  `test` ::= (check-within `expr` `expr` `expr`)

- As with `check-expect`, the first two expressions are, respectively, for the value being tested and the expected value
- The third expression defines the error tolerance
- For the test to pass the actual value must be at most the tolerance away from the expected value.

Expressions and Data Types

Numbers

- To test computations involving real numbers, the grammar production rules for `test` must be expanded as follows:

  `test` ::= (check-within `expr` `expr` `expr`)

- As with `check-expect`, the first two expressions are, respectively, for the value being tested and the expected value
- The third expression defines the error tolerance
- For the test to pass the actual value must be at most the tolerance away from the expected value.
-

```
;; The circumference of a circle with radius r is 2 * pi * r

;; The circumference of a circle with radius 7
(define CIRCUM-7 (* 2 pi 7))

(check-within CIRCUM-7 43.98 0.01)
```

Expressions and Data Types

HOMEWORK

- Problems: 10–12

Expressions and Data Types

Strings and Characters

- A string is described as anything that is inside double quotes
- "anything" suggests that there may be more than one piece of data
- String is our first example of a compound data type

Expressions and Data Types

Strings and Characters

- A string is described as anything that is inside double quotes
- "anything" suggests that there may be more than one piece of data
- String is our first example of a compound data type
- The string `"cat"` may be constructed using the strings: `"c" "a" "t"`
- `"cat" = (string-append "c" "a" "t")`

Expressions and Data Types

Strings and Characters

- A string is described as anything that is inside double quotes
- "anything" suggests that there may be more than one piece of data
- String is our first example of a compound data type
- The string `"cat"` may be constructed using the strings: `"c" "a" "t"`
- `"cat" = (string-append "c" "a" "t")`
- When data is compound, it is possible to access its components
- Functions that extract a component from an instance of compound data are called *selectors*

Expressions and Data Types

Strings and Characters

- A string is described as anything that is inside double quotes
- "anything" suggests that there may be more than one piece of data
- String is our first example of a compound data type
- The string `"cat"` may be constructed using the strings: `"c" "a" "t"`
- `"cat" = (string-append "c" "a" "t")`
- When data is compound, it is possible to access its components
- Functions that extract a component from an instance of compound data are called *selectors*
- BSL function to extract substrings:

```
;; string ℕ [ℕ] → string
;; Purpose: Extract from s the substring starting in
;;          position i and ending in position j-1
(substring s i j)
```

- Requires at least two inputs: a string and a natural number less than or equal to the length of s

Expressions and Data Types

Strings and Characters

- A string is described as anything that is inside double quotes
- "anything" suggests that there may be more than one piece of data
- String is our first example of a compound data type
- The string `"cat"` may be constructed using the strings: `"c" "a" "t"`
- `"cat" = (string-append "c" "a" "t")`
- When data is compound, it is possible to access its components
- Functions that extract a component from an instance of compound data are called *selectors*
- BSL function to extract substrings:

```
;; string ℕ [ℕ] → string
;; Purpose: Extract from s the substring starting in
;;          position i and ending in position j-1
(substring s i j)
```

- Requires at least two inputs: a string and a natural number less than or equal to the length of `s`
- The `[]` indicates that the third argument is optional
- The optional third argument is a natural number, `j`, such that 0<=j<=length of `s`
- Think of `i` and `j` as defining the interval `[i..j)` and `substring` as extracting the substring from position `i` to position `j`
- No third argument is shorthand for `j` being the length of the `s`
- `(substring "Wow!" 2)` is shorthand for `(substring "Wow!" 2 4)`
- Note that strings positions start at 0 (not 1).

Expressions and Data Types

Numbers

```
(define CAT "cat")

(check-expect CAT (string-append "c" "a" "t"))
(check-expect (substring CAT 0 1) "c")
(check-expect (substring CAT 1 2) "a")
(check-expect (substring CAT 2 3) "t")
(check-expect (substring CAT 3 3) "")
(check-expect (substring CAT 0) CAT)
(check-expect (substring CAT 1) (substring CAT 1 3))
```

Expressions and Data Types

Numbers

- ```
  (define CAT "cat")

  (check-expect CAT (string-append "c" "a" "t"))
  (check-expect (substring CAT 0 1) "c")
  (check-expect (substring CAT 1 2) "a")
  (check-expect (substring CAT 2 3) "t")
  (check-expect (substring CAT 3 3) "")
  (check-expect (substring CAT 0) CAT)
  (check-expect (substring CAT 1) (substring CAT 1 3))
  ```
- Strings are not really made up of substrings
- Every element of a string is a character (abbreviated as `char`)
- `substring` in reality converts the selected characters into a string for the programmer to make programming easier
  ```

Expressions and Data Types

Definitions

- Expressions may evaluate to characters:

  `expr ::= #\<character constant>`

- To denote characters, they are typed by prefixing them with `#\` followed by a character constant

Expressions and Data Types

Definitions

- Expressions may evaluate to characters:

  `expr ::= #\<character constant>`

- To denote characters, they are typed by prefixing them with `#\` followed by a character constant
- To access the characters of a string use the following function:

```
;; string ℕ → char
;; Purpose: Extract the ith char from s
(string-ref s i)
```

- The index `i` must satisfy `0<=i<=length of s`

Expressions and Data Types

Definitions

- Expressions may evaluate to characters:

  expr ::= #\<character constant>

- To denote characters, they are typed by prefixing them with #\ followed by a character constant
- To access the characters of a string use the following function:

```
;; string ℕ → char
;; Purpose: Extract the i^th char from s
(string-ref s i)
```

- The index i must satisfy 0<=i<=length of s
-

```
(define DOG "dog")

(check-expect (string-ref DOG 0) #\d)
(check-expect (string-ref DOG 1) #\o)
(check-expect (string-ref DOG 2) #\g)
```

Expressions and Data Types

HOMEWORK

- Problems: 13–17

Expressions and Data Types

Symbols

- Symbols are like strings that are not decomposable
- Symbol is a primitive type like number
- Symbols are used when we are not interested in manipulating or accessing the characters

Expressions and Data Types

Symbols

- Symbols are like strings that are not decomposable
- Symbol is a primitive type like number
- Symbols are used when we are not interested in manipulating or accessing the characters
- Grammar:

$$\texttt{expr} \quad ::= \quad {}^{\prime}\langle\text{character}\rangle^{+}$$

Expressions and Data Types

Symbols

- Symbols are like strings that are not decomposable
- Symbol is a primitive type like number
- Symbols are used when we are not interested in manipulating or accessing the characters
- Grammar:

$$\texttt{expr} \quad ::= \quad \texttt{'}\langle\text{character}\rangle^{+}$$

- `DrRacket` denotes symbols as a `'` followed by the name of the symbol
- `(string->symbol "Apple")`
- `(string->symbol "1024")` = `'`|1024|.

Expressions and Data Types

Symbols

- A symbol is convertible to a string and vice versa:

```
(define NAME 'Joshua)
(define NAME2 'Sachin)

(define STR-NAME (symbol->string NAME))
(define STR-NAME2 (symbol->string NAME2))

(check-expect STR-NAME "Joshua")
(check-expect STR-NAME2 "Sachin")

(define JOSH (string->symbol STR-NAME))
(define SACH (string->symbol STR-NAME2))

(check-expect JOSH NAME)
(check-expect SACH NAME2)
```

- Two symbols are converted to strings using `symbol->string`
- Converted back to symbols using `string->symbol`

Expressions and Data Types

Symbols

- A symbol is convertible to a string and vice versa:

```
(define NAME 'Joshua)
(define NAME2 'Sachin)

(define STR-NAME (symbol->string NAME))
(define STR-NAME2 (symbol->string NAME2))

(check-expect STR-NAME "Joshua")
(check-expect STR-NAME2 "Sachin")

(define JOSH (string->symbol STR-NAME))
(define SACH (string->symbol STR-NAME2))

(check-expect JOSH NAME)
(check-expect SACH NAME2)
```

- Two symbols are converted to strings using `symbol->string`
- Converted back to symbols using `string->symbol`
- The functions `symbol->string` and `string->symbol` are interesting because they are *inverses* of each other.
- Two functions are inverses if when composed the original input is returned:

  $(f \circ g)(x_1) = x_1 = (g \circ f)(x_1)$

- Do not panic if you do not remember or recognize function composition and inverses right now

Expressions and Data Types

HOMEWORK

- 8–20

Expressions and Data Types

Booleans

- Boolean is a primitive type of data
- There are only two kinds of Boolean values: `#true` and `#false`
- Important because it allows to perform computations that depend on whether conditions hold or fail to hold

Expressions and Data Types

Booleans

- Boolean is a primitive type of data
- There are only two kinds of Boolean values: `#true` and `#false`
- Important because it allows to perform computations that depend on whether conditions hold or fail to hold
- In Mathematics you have studied the absolute value function:

$$absVal(x) = \begin{cases} x & \text{if } x \geq 0 \\ -x & \text{otherwise} \end{cases}$$

- How do you compute `absVal(-100)`?

Expressions and Data Types

Booleans

- Boolean is a primitive type of data
- There are only two kinds of Boolean values: `#true` and `#false`
- Important because it allows to perform computations that depend on whether conditions hold or fail to hold
- In Mathematics you have studied the absolute value function:

$$absVal(x) = \begin{cases} x & \text{if } x \geq 0 \\ -x & \text{otherwise} \end{cases}$$

- How do you compute `absVal(-100)`?
- You plug-in -100 for `x`, evaluate the condition, $-100 \geq 0$, and determine that it is false
- Since this condition is false, the value of the function is given by -(-100) = 100.

Expressions and Data Types

Booleans

- Boolean is a primitive type of data
- There are only two kinds of Boolean values: `#true` and `#false`
- Important because it allows to perform computations that depend on whether conditions hold or fail to hold
- In Mathematics you have studied the absolute value function:

$$absVal(x) = \begin{cases} x & \text{if } x \geq 0 \\ -x & \text{otherwise} \end{cases}$$

- How do you compute `absVal(-100)`?
- You plug-in -100 for `x`, evaluate the condition, $-100 \geq 0$, and determine that it is false
- Since this condition is false, the value of the function is given by -(-100) = 100.
- Two new `expr` production rules for Booleans:

```
expr   ::=   #true
       ::=   #false
```

Expressions and Data Types

Booleans

- Booleans have the following basic functions: `and` (also denoted by $\wedge$), `or` (also denoted by $\vee$), and `not` (also denoted by $\neg$)

Expressions and Data Types

Booleans

- Booleans have the following basic functions: `and` (also denoted by $\wedge$), `or` (also denoted by $\vee$), and `not` (also denoted by $\neg$)
- A truth table is used to illustrate the four basic Boolean functions:

x	**y**	**x** $\wedge$ **y**	**x** $\vee$ **y**	$\neg$**x**
false	false	false	false	true
false	true	false	true	true
true	false	false	true	false
true	true	true	true	false

Expressions and Data Types

Booleans

- Booleans have the following basic functions: `and` (also denoted by $\wedge$), `or` (also denoted by $\vee$), and `not` (also denoted by $\neg$)
- A truth table is used to illustrate the four basic Boolean functions:

x	**y**	**x** $\wedge$ **y**	**x** $\vee$ **y**	$\neg$**x**
false	false	false	false	true
false	true	false	true	true
true	false	false	true	false
true	true	true	true	false

- It is not always necessary to evaluate all the arguments to `and` or to `or`:

  `(and #false (≤ x 20))` `(or #true (> i 100))`

Expressions and Data Types

Booleans

- BSL takes advantage of the fact that no all arguments to `and` and `or` may have to be evaluated to make them faster
- Stops evaluating arguments to `and` after the first that evaluates to `#false`
- Stops evaluating arguments to `or` after the first that evaluates to `#true`

Expressions and Data Types

Booleans

- BSL takes advantage of the fact that no all arguments to `and` and `or` may have to be evaluated to make them faster
- Stops evaluating arguments to `and` after the first that evaluates to `#false`
- Stops evaluating arguments to `or` after the first that evaluates to `#true`
- To achieve this, `and` and `or` are **not** functions in BSL
- They are new types of expressions:

```
expr   ::=   (and expr expr expr*)
       ::=   (or  expr expr expr*)
```

- `not` is a one-input function

Expressions and Data Types

Booleans

- Program to validate Booleans:

```
(define AND-FF (and #false #false))
(define AND-FT (and #false #true))
(define AND-TF (and #true #false))
(define AND-TT (and #true #true))
(define OR-FF  (or #false #false))
(define OR-FT  (or #false #true))
(define OR-TF  (or #true #false))
(define OR-TT  (or #true #true))
(define NOT-F  (not #false))  (define NOT-T  (not #true))

(check-expect AND-FF #false)
(check-expect AND-FT #false)
(check-expect AND-TF #false)
(check-expect AND-TT #true)
(check-expect OR-FF  #false)
(check-expect OR-FT  #true)
(check-expect OR-TF  #true)
(check-expect OR-TT #true)
(check-expect NOT-F #true)
(check-expect NOT-T #false)
```

- In BSL, the expressions that remain unevaluated are highlighted in black

Expressions and Data Types

HOMEWORK

- Problem: 21

Expressions and Data Types

Predicates

- Functions that return a Boolean are called *predicate* functions
- They are useful to determine if the input meets some condition

Expressions and Data Types

Predicates

- Functions that return a Boolean are called *predicate* functions
- They are useful to determine if the input meets some condition
- In BSL built-in types have a predicate to determine if the input is of that type:

Type	Predicate
number	`number?`
string	`string?`
character	`char?`
symbol	`symbol?`
boolean	`boolean?`
image	`image?`

- `(string? "Hi there!")` evaluates to `#true`
- `(boolean? 103167)` evaluates to `#false`.

Expressions and Data Types

Predicates

- You have used predicates in high school Mathematics: $<$, $>$, $\geq$, $\leq$, and $=$

Expressions and Data Types

Predicates

- You have used predicates in high school Mathematics: <, >, $\geq$, $\leq$, and =
- In BSL: <, >, >=, <=, and =.
- These functions all require 1 or more inputs

Expressions and Data Types

Predicates

- You have used predicates in high school Mathematics: <, >, $\geq$, $\leq$, and =
- In BSL: <, >, >=, <=, and =.
- These functions all require 1 or more inputs
- Typing `(< 10)` at the prompt returns `#true`
- Why does this make sense?

Expressions and Data Types

Predicates

- You have used predicates in high school Mathematics: <, >, $\geq$, $\leq$, and =
- In BSL: <, >, >=, <=, and =.
- These functions all require 1 or more inputs
- Typing `(< 10)` at the prompt returns `#true`
- Why does this make sense?
- A numerical predicate returns `#true` unless one of its input makes it `#false`

Expressions and Data Types

Predicates

- You have used predicates in high school Mathematics: <, >, $\geq$, $\leq$, and =
- In BSL: <, >, >=, <=, and =.
- These functions all require 1 or more inputs
- Typing `(< 10)` at the prompt returns `#true`
- Why does this make sense?
- A numerical predicate returns `#true` unless one of its input makes it `#false`
- What does `(< 10 12 18 31)` evaluating to `#true` mean?

Expressions and Data Types

Predicates

- You have used predicates in high school Mathematics: <, >, $\geq$, $\leq$, and =
- In BSL: <, >, >=, <=, and =.
- These functions all require 1 or more inputs
- Typing `(< 10)` at the prompt returns `#true`
- Why does this make sense?
- A numerical predicate returns `#true` unless one of its input makes it `#false`
- What does `(< 10 12 18 31)` evaluating to `#true` mean?
- This is the BSL expression for `10 < 12 < 18 < 31`

Expressions and Data Types

Predicates

- Strings share an important property with numbers: they are *ordinal*
- This means that they may be ordered

Expressions and Data Types

Predicates

- Strings share an important property with numbers: they are *ordinal*
- This means that they may be ordered
- Strings may be lexicographically ordered (i.e., alphabetically ordered)

Expressions and Data Types

Predicates

- Strings share an important property with numbers: they are *ordinal*
- This means that they may be ordered
- Strings may be lexicographically ordered (i.e., alphabetically ordered)
- We can ask if one string is less than another or if one string is greater than or equal to another
- We cannot, however, use numerical predicates with strings
- The corresponding string predicates are: `string<?`, `string>?`, `string<=?`, `string>=?`, and `string=?`.
- Observe that the `?` suggests that a question is being asked
- `(string<?  "a" "b")` is asking if `"a"` comes before `"b"`.

Expressions and Data Types

Predicates

- Symbols are not an ordinal type
- There are no functions to compare symbols like > or `string<?`

Expressions and Data Types

Predicates

- Symbols are not an ordinal type
- There are no functions to compare symbols like > or `string<?`
- `symbol?` tests if its input is a symbol
- `eq?` may be used to test if two symbols are equal

Expressions and Data Types

Predicates

- Symbols are not an ordinal type
- There are no functions to compare symbols like > or `string<?`
- `symbol?` tests if its input is a symbol
- `eq?` may be used to test if two symbols are equal
- `eq?` does not exclusively work with symbols
- `(eq? 1 1)` = `#true`
- `(eq? "Alpha Centauri" "Alpha")` = `#falseT`

Expressions and Data Types

Predicates

- Symbols are not an ordinal type
- There are no functions to compare symbols like > or `string<?`
- `symbol?` tests if its input is a symbol
- `eq?` may be used to test if two symbols are equal
- `eq?` does not exclusively work with symbols
- `(eq? 1 1)` = `#true`
- `(eq? "Alpha Centauri" "Alpha")` = `#falseT`
- Functions, like `eq?`, that work for many types of inputs are called *generic functions*

Expressions and Data Types

Predicates

- Predicates for Booleans include `boolean?` and `false?`

Expressions and Data Types

Predicates

- Predicates for Booleans include `boolean?` and `false?`
- Why is `true?` not a predicate in BSL?

Expressions and Data Types

Predicates

- Predicates for Booleans include `boolean?` and `false?`
- Why is `true?` not a predicate in BSL?
- Creators of BSL cannot implement every conceivable function
- Criteria: may a value be computed using other functions?

Expressions and Data Types

Predicates

- Predicates for Booleans include `boolean?` and `false?`
- Why is `true?` not a predicate in BSL?
- Creators of BSL cannot implement every conceivable function
- Criteria: may a value be computed using other functions?
- Can we write expressions to determine if a variable is `#true`?

Expressions and Data Types

Predicates

- Predicates for Booleans include `boolean?` and `false?`
- Why is `true?` not a predicate in `BSL`?
- Creators of `BSL` cannot implement every conceivable function
- Criteria: may a value be computed using other functions?
- Can we write expressions to determine if a variable is `#true`?
- The value must be a Boolean and must not be `#false`.

Expressions and Data Types

Predicates

- Test our observations:

```
(define A-STRING "This is not true.")
(define A-SYMBOL 'not-true)
(define A-NUMBER 87) (define A-BOOL #true) (define A-BOOL2 #false)
```

Expressions and Data Types

Predicates

- Test our observations:

```
(define A-STRING "This is not true.")
(define A-SYMBOL 'not-true)
(define A-NUMBER 87) (define A-BOOL #true) (define A-BOOL2 #false)
```

-

```
(define IS-TRUE-A-STRING (and (boolean? A-STRING)
                              (not (false? A-STRING))))
(define IS-TRUE-A-SYMBOL (and (boolean? A-SYMBOL)
                              (not (false? A-SYMBOL))))
(define IS-TRUE-A-NUMBER (and (boolean? A-NUMBER)
                              (not (false? A-NUMBER))))
(define IS-TRUE-A-BOOL   (and (boolean? A-BOOL)
                              (not (false? A-BOOL))))
(define IS-TRUE-A-BOOL2  (and (boolean? A-BOOL2)
                              (not (false? A-BOOL2))))
```

Expressions and Data Types

Predicates

- Test our observations:

```
(define A-STRING "This is not true.")
(define A-SYMBOL 'not-true)
(define A-NUMBER 87) (define A-BOOL #true) (define A-BOOL2 #false)
```

-
```
(define IS-TRUE-A-STRING (and (boolean? A-STRING)
                              (not (false? A-STRING))))
(define IS-TRUE-A-SYMBOL (and (boolean? A-SYMBOL)
                              (not (false? A-SYMBOL))))
(define IS-TRUE-A-NUMBER (and (boolean? A-NUMBER)
                              (not (false? A-NUMBER))))
(define IS-TRUE-A-BOOL   (and (boolean? A-BOOL)
                              (not (false? A-BOOL))))
(define IS-TRUE-A-BOOL2  (and (boolean? A-BOOL2)
                              (not (false? A-BOOL2))))
```

-
```
(check-expect IS-TRUE-A-STRING #false)
     (check-expect IS-TRUE-A-SYMBOL #false)
(check-expect IS-TRUE-A-NUMBER #false)
     (check-expect IS-TRUE-A-BOOL   #true)
(check-expect IS-TRUE-A-BOOL2  #false)
```

Expressions and Data Types

Predicates

- Test our observations:

```
(define A-STRING "This is not true.")
(define A-SYMBOL 'not-true)
(define A-NUMBER 87) (define A-BOOL #true) (define A-BOOL2 #false)
```

-
```
(define IS-TRUE-A-STRING (and (boolean? A-STRING)
                              (not (false? A-STRING))))
(define IS-TRUE-A-SYMBOL (and (boolean? A-SYMBOL)
                              (not (false? A-SYMBOL))))
(define IS-TRUE-A-NUMBER (and (boolean? A-NUMBER)
                              (not (false? A-NUMBER))))
(define IS-TRUE-A-BOOL   (and (boolean? A-BOOL)
                              (not (false? A-BOOL))))
(define IS-TRUE-A-BOOL2  (and (boolean? A-BOOL2)
                              (not (false? A-BOOL2))))
```

-
```
(check-expect IS-TRUE-A-STRING #false)
     (check-expect IS-TRUE-A-SYMBOL #false)
(check-expect IS-TRUE-A-NUMBER #false)
     (check-expect IS-TRUE-A-BOOL   #true)
(check-expect IS-TRUE-A-BOOL2  #false)
```

- The values returned by `boolean?` and `not` are inputs to `and` and the value returned by `false?` is input to `not`
- We begin to see that function composition is useful to design programs.

Expressions and Data Types

HOMEWORK

- Problems: 22–26

Expressions and Data Types

Images

- An image is a visual rectangular type of compound data

Expressions and Data Types

Images

- An image is a visual rectangular type of compound data
- Every image consists of pixels: representing a colored dot
- Every image has a length and a width measured in pixels

Expressions and Data Types

Images

- An image is a visual rectangular type of compound data
- Every image consists of pixels: representing a colored dot
- Every image has a length and a width measured in pixels
- In BSL, expressions may evaluate to an image:

```
expr    ::=    <image>
```

- <image> is a graphic that we either create or paste

Expressions and Data Types

Images

- An image is a visual rectangular type of compound data
- Every image consists of pixels: representing a colored dot
- Every image has a length and a width measured in pixels
- In BSL, expressions may evaluate to an image:

  `expr ::= <image>`

- <image> is a graphic that we either create or paste
- BSL provides us with a *teachpack* to create and manipulate images called `2htdp/image`
- To use the first line in you program needs to be:

```
(require 2htdp/image)
```

Expressions and Data Types

Images

- An image is a visual rectangular type of compound data
- Every image consists of pixels: representing a colored dot
- Every image has a length and a width measured in pixels
- In BSL, expressions may evaluate to an image:

  `expr ::= <image>`

- <image> is a graphic that we either create or paste
- BSL provides us with a *teachpack* to create and manipulate images called `2htdp/image`
- To use the first line in you program needs to be:

```
(require 2htdp/image)
```

- This teachpack contains many functions that are split into three broad categories:

  Basic Constructors These are functions to create basic images like circles, rectangles, ellipses, stars, and text.

  Property Selectors These are functions to extract properties of images such as width and length.

  Image Composers These are functions that combine existing images to create new images.

Expressions and Data Types

Images

- r

Expressions and Data Types

Images

- Useful to understand:

  mode This refers to how basic geometric shapes are drawn: not filled or filled. Use `'outline` or `"outline"` when an outline of a shape is desired. Use `'solid` or `"solid"` when a solid shape is desired.

  image color This is a string or a symbol representing the desired color. Search for `image-color?` in `DrRacket`'s `Help Desk` to see the list of predefined colors.

Expressions and Data Types

Images: Constructors

- ```
  (require 2htdp/image)
  (define RECT-WIDTH 35)
  (define RECT-HEIGHT 55)
  (define RECT-MODE 'outline)
  (define RECT-COLOR 'red)
  ```
  ```

Expressions and Data Types

Images: Constructors

```
(require 2htdp/image)
(define RECT-WIDTH 35)
(define RECT-HEIGHT 55)
(define RECT-MODE 'outline)
(define RECT-COLOR 'red)

(define CIRC-RADIUS 25)
(define CIRC-MODE 'solid)
(define CIRC-COLOR 'green)

(define RED-OUTLINE-RECT   (rectangle RECT-WIDTH RECT-HEIGHT
                                      RECT-MODE  RECT-COLOR))
(define GREEN-SOLID-CIRCLE (circle CIRC-RADIUS
                                   CIRC-MODE
                                   CIRC-COLOR))
```

Expressions and Data Types

Images: Constructors

```
(require 2htdp/image)
(define RECT-WIDTH 35)
(define RECT-HEIGHT 55)
(define RECT-MODE 'outline)
(define RECT-COLOR 'red)

(define CIRC-RADIUS 25)
(define CIRC-MODE 'solid)
(define CIRC-COLOR 'green)

(define RED-OUTLINE-RECT   (rectangle RECT-WIDTH RECT-HEIGHT
                                      RECT-MODE  RECT-COLOR))
(define GREEN-SOLID-CIRCLE (circle CIRC-RADIUS
                                   CIRC-MODE
                                   CIRC-COLOR))
(check-expect RED-OUTLINE-RECT (rectangle 35 55 'outline 'red))
```

```
(check-expect RED-OUTLINE-RECT
```

```
)
(check-expect GREEN-SOLID-CIRCLE (circle 25 'solid 'green))
```

```
(check-expect GREEN-SOLID-CIRCLE
```

```
)
```

Expressions and Data Types

Images: Selectors

- There are two selector functions:

image-width	Returns the width of an image in pixels.
image-height	Returns the height of an image in pixels.

Expressions and Data Types

Images: Selectors

- There are two selector functions:

  image-width Returns the width of an image in pixels.

  image-height Returns the height of an image in pixels.

- Consider the problem of computing half the width, half the height, and the number of pixels for the following images:

```
(define A-STAR-IMG   (radial-star 10 5 35 'outline 'gold))
(define A-RHOMBUS-IMG (rhombus 60 75 'solid 'red))
```

Expressions and Data Types

Images: Selectors

- There are two selector functions:

  image-width Returns the width of an image in pixels.

  image-height Returns the height of an image in pixels.

- Consider the problem of computing half the width, half the height, and the number of pixels for the following images:

```
(define A-STAR-IMG   (radial-star 10 5 35 'outline 'gold))
(define A-RHOMBUS-IMG (rhombus 60 75 'solid 'red))
```

- First: clearly identify what needs to be computed and how:

  Half the Width Divide the image's width by 2.

  Half the Height Divide the image's height by 2.

  Number of Pixels Compute the image's area.

Expressions and Data Types

Images: Selectors

- There are two selector functions:

  image-width Returns the width of an image in pixels.

  image-height Returns the height of an image in pixels.

- Consider the problem of computing half the width, half the height, and the number of pixels for the following images:

```
(define A-STAR-IMG   (radial-star 10 5 35 'outline 'gold))
(define A-RHOMBUS-IMG (rhombus 60 75 'solid 'red))
```

- First: clearly identify what needs to be computed and how:

  Half the Width Divide the image's width by 2.

  Half the Height Divide the image's height by 2.

  Number of Pixels Compute the image's area.

- How are we to test our computations?

Expressions and Data Types

Images: Selectors

- There are two selector functions:

  image-width Returns the width of an image in pixels.

  image-height Returns the height of an image in pixels.
- Consider the problem of computing half the width, half the height, and the number of pixels for the following images:

```
(define A-STAR-IMG   (radial-star 10 5 35 'outline 'gold))
(define A-RHOMBUS-IMG (rhombus 60 75 'solid 'red))
```

- First: clearly identify what needs to be computed and how:

  Half the Width Divide the image's width by 2.

  Half the Height Divide the image's height by 2.

  Number of Pixels Compute the image's area.
- How are we to test our computations?
- When we do not know what a specific value ought to be or it is too difficult to write a specific value we can use *property-based testing*
- Instead of testing for the specific value of an expression, we test properties that the value of the expression ought to have
- Twice half the width ought to be the width
- Twice half the height ought to be the height
- The number of pixels divided by the image's height ought to be the image's width
- The number of pixels divided by the image's width ought to be the image's height

Expressions and Data Types

Images

- ```
  (require 2htdp/image)
  (define A-STAR-IMG (radial-star 10 5 35 'outline 'gold))
  (define STAR-W (image-width A-STAR-IMG))
  (define STAR-H (image-height A-STAR-IMG))
  (define STAR-HALF-W (/ STAR-W 2))
  (define STAR-HALF-H (/ STAR-H 2))
  (define STAR-PIXELS (* STAR-W STAR-H))
  ;; Star tests
  (check-expect (* 2 STAR-HALF-W) STAR-W)
  (check-expect (* 2 STAR-HALF-H) STAR-H)
  (check-expect (/ STAR-PIXELS STAR-W) STAR-H)
  (check-expect (/ STAR-PIXELS STAR-H) STAR-W)
  ```
  ```

Expressions and Data Types

Images

- ```
  (require 2htdp/image)
  (define A-STAR-IMG (radial-star 10 5 35 'outline 'gold))
  (define STAR-W (image-width A-STAR-IMG))
  (define STAR-H (image-height A-STAR-IMG))
  (define STAR-HALF-W (/ STAR-W 2))
  (define STAR-HALF-H (/ STAR-H 2))
  (define STAR-PIXELS (* STAR-W STAR-H))
  ;; Star tests
  (check-expect (* 2 STAR-HALF-W) STAR-W)
  (check-expect (* 2 STAR-HALF-H) STAR-H)
  (check-expect (/ STAR-PIXELS STAR-W) STAR-H)
  (check-expect (/ STAR-PIXELS STAR-H) STAR-W)
  ```
- ```
  (define A-RHOMBUS-IMG (rhombus 60 75 'solid 'red))
  (define RHOMBUS-W (image-width A-RHOMBUS-IMG))
  (define RHOMBUS-H (image-height A-RHOMBUS-IMG))
  (define RHOMBUS-HALF-W (/ RHOMBUS-W 2))
  (define RHOMBUS-HALF-H (/ RHOMBUS-H 2))
  (define RHOMBUS-PIXELS (* RHOMBUS-W RHOMBUS-H))
  ;; Rhombus tests
  (check-expect (* 2 RHOMBUS-HALF-W) RHOMBUS-W)
  (check-expect (* 2 RHOMBUS-HALF-H) RHOMBUS-H)
  (check-expect (/ RHOMBUS-PIXELS RHOMBUS-W) RHOMBUS-H)
  (check-expect (/ RHOMBUS-PIXELS RHOMBUS-H) RHOMBUS-W)
  ```

Expressions and Data Types

Images: Composers

- Image composers are used to create new images from existing images
- The image teachpack provides a myriad of such functions and you ought to experiment with them as you read the documentation in the `Help Desk`

Expressions and Data Types

Images: Composers

- Image composers are used to create new images from existing images
- The image teachpack provides a myriad of such functions and you ought to experiment with them as you read the documentation in the `Help Desk`
- Create:

Expressions and Data Types

Images: Composers

- Image composers are used to create new images from existing images
- The image teachpack provides a myriad of such functions and you ought to experiment with them as you read the documentation in the `Help Desk`
- Create:

- What needs to be computed?

Expressions and Data Types

Images: Composers

- Image composers are used to create new images from existing images
- The image teachpack provides a myriad of such functions and you ought to experiment with them as you read the documentation in the `Help Desk`
- Create:

- What needs to be computed?
- The image consists of 6 nested squares
- The largest square is yellow and at the bottom
- The smallest square is blue and on the top
- Suggests a divide and conquer strategy

Expressions and Data Types

Images: Composers

- Image composers are used to create new images from existing images
- The image teachpack provides a myriad of such functions and you ought to experiment with them as you read the documentation in the `Help Desk`
- Create:

- What needs to be computed?
- The image consists of 6 nested squares
- The largest square is yellow and at the bottom
- The smallest square is blue and on the top
- Suggests a divide and conquer strategy
- First, compute all the needed squares
- Second, overlay the squares to create the desired image

Expressions and Data Types

Images: Composers

- Image composers are used to create new images from existing images
- The image teachpack provides a myriad of such functions and you ought to experiment with them as you read the documentation in the `Help Desk`
- Create:

- What needs to be computed?
- The image consists of 6 nested squares
- The largest square is yellow and at the bottom
- The smallest square is blue and on the top
- Suggests a divide and conquer strategy
- First, compute all the needed squares
- Second, overlay the squares to create the desired image
- We can test the result using property-based testing

Expressions and Data Types

Images: Composers

```
;; Tests
;; NOTE: SQUARE0 is the largest square.
(check-expect (image-width NESTED-SQS)  (image-width SQUARE0))
(check-expect (image-height NESTED-SQS) (image-height SQUARE0))

;; The squares
(define SQUARE0 (square 60 'solid 'yellow)) ;; largest square
(define SQUARE1 (square 50 'solid 'blue))
(define SQUARE2 (square 40 'solid 'yellow))
(define SQUARE3 (square 30 'solid 'blue))
(define SQUARE4 (square 20 'solid 'yellow))
(define SQUARE5 (square 10 'solid 'blue)) ;; smallest square

;; The Nested Squares
(define NESTED-SQS (overlay SQUARE5 SQUARE4 SQUARE3
                            SQUARE2 SQUARE1 SQUARE0))
```

Expressions and Data Types

Images: Composers

- It is natural to wonder how the placing is done

Expressions and Data Types

Images: Composers

- It is natural to wonder how the placing is done
- One pixel serves as an *anchor point* around which images are placed
- Unless otherwise specified, image composing functions use the pixel at the center of the image as the anchor point

Expressions and Data Types

Images: Composers

- It is natural to wonder how the placing is done
- One pixel serves as an *anchor point* around which images are placed
- Unless otherwise specified, image composing functions use the pixel at the center of the image as the anchor point
- `overlay` places all the center points of the images one on top of another

Expressions and Data Types

Images: Composers

- It is natural to wonder how the placing is done
- One pixel serves as an *anchor point* around which images are placed
- Unless otherwise specified, image composing functions use the pixel at the center of the image as the anchor point
- `overlay` places all the center points of the images one on top of another
- You may contrast this behavior with the behavior of the application expression:

```
(overlay/xy img1 i j img2)
```

- The images start aligned with respect of their top-left corner pixel
- Overlays `img1` over `img2`, but moves `img2` `i` pixels to the right and `j` pixels down

Expressions and Data Types

Images: Composers

- It is natural to wonder how the placing is done
- One pixel serves as an *anchor point* around which images are placed
- Unless otherwise specified, image composing functions use the pixel at the center of the image as the anchor point
- `overlay` places all the center points of the images one on top of another
- You may contrast this behavior with the behavior of the application expression:

```
(overlay/xy img1 i j img2)
```

- The images start aligned with respect of their top-left corner pixel
- Overlays `img1` over `img2`, but moves `img2` `i` pixels to the right and `j` pixels down
- Another characteristic of our program is that the testing is not satisfying
- Consider what happens if we mistakenly define `NESTED-SQS` as follows:

```
(define NESTED-SQS (overlay SQUARE0 SQUARE1 SQUARE2
                            SQUARE3 SQUARE4 SQUARE5))
```

- No tests fail despite that `NESTED-SQS` is the image of a yellow square

Expressions and Data Types

Images: Composers

- It is natural to wonder how the placing is done
- One pixel serves as an *anchor point* around which images are placed
- Unless otherwise specified, image composing functions use the pixel at the center of the image as the anchor point
- `overlay` places all the center points of the images one on top of another
- You may contrast this behavior with the behavior of the application expression:

```
(overlay/xy img1 i j img2)
```

- The images start aligned with respect of their top-left corner pixel
- Overlays `img1` over `img2`, but moves `img2` `i` pixels to the right and `j` pixels down
- Another characteristic of our program is that the testing is not satisfying
- Consider what happens if we mistakenly define `NESTED-SQS` as follows:

```
(define NESTED-SQS (overlay SQUARE0 SQUARE1 SQUARE2
                            SQUARE3 SQUARE4 SQUARE5))
```

- No tests fail despite that `NESTED-SQS` is the image of a yellow square
- To remedy such a situation, add unit tests using actual values that are computed
- Experiment with our program
- Once the desired image is computed, copy and paste it as the expected result for testing `NESTED-SQS`
- Any reader of your code can see that both your model and your result are consistent with expectations

Expressions and Data Types

Images: Empty Scenes and Placing Images

- There are two image composing functions that are used extensively to create animations and video games:

`empty-scene`	Creates an empty image of some given width and height. Optionally, the color of the empty image may be provided.
`place-image`	Places the first given image at the given `x` and `y` coordinates in the second given image.

Expressions and Data Types

Images: Empty Scenes and Placing Images

- There are two image composing functions that are used extensively to create animations and video games:

  `empty-scene` Creates an empty image of some given width and height. Optionally, the color of the empty image may be provided.

  `place-image` Places the first given image at the given `x` and `y` coordinates in the second given image.

- As characters and elements change in a video game, the displayed image must change
- This is done by computing a new image
- The empty image is used to create the base image of the video game or animation
- The function `place-image` is used to place elements.

Expressions and Data Types

Images: Empty Scenes and Placing Images

- To properly use `place-image`, a brief description of the computer graphics coordinate system is needed

- `x` grows from left to right
- `y` grows from top to bottom

Expressions and Data Types

Images: Empty Scenes and Placing Images

- To properly use `place-image`, a brief description of the computer graphics coordinate system is needed

- `x` grows from left to right
- `y` grows from top to bottom
- Image of `WIDTH x HEIGHT`, `x` in `[0..WIDTH-1]` and `y` in `[0..HEIGHT-1]`
- If any part of a placed image falls outside these coordinate ranges, the image is cropped and parts of the placed image do not appear in the result

Expressions and Data Types

Images: Empty Scenes and Placing Images

- Create an image with three different alien ships in a black empty scene.

Expressions and Data Types

Images: Empty Scenes and Placing Images

- Create an image with three different alien ships in a black empty scene.
- Divide and conquer:
  - Define the dimensions of the empty scene.
  - Compute three different alien ships.
  - Compute an image with one alien ship.
  - Compute an image with two alien ships.
  - Compute an image with three alien ships.
  - Write tests.

Expressions and Data Types

Images: Empty Scenes and Placing Images

- Partial Solution (there's a bug)

```
(require 2htdp/image)

(define WIDTH 200)
(define HEIGHT 100)

(define E-SCENE (empty-scene WIDTH HEIGHT 'black))

(define ALIEN-SHIP0 (overlay (circle 7 'solid 'gray)
                             (rectangle 23 3 'solid 'gray)))

(define ALIEN-SHIP1 (overlay (circle 7 'solid 'pink)
                             (rectangle 23 3 'solid 'pink)))

(define ALIEN-SHIP2 (overlay (circle 7 'solid 'white)
                             (rectangle 23 3 'solid 'white)))

(define SCN0 (place-image ALIEN-SHIP0 15 30 E-SCENE))

(define SCN1 (place-image ALIEN-SHIP1 197 80 SCN0))

(define SCN2 (place-image ALIEN-SHIP2 (/ WIDTH 2) (/ HEIGHT 2)
```

Expressions and Data Types

Images: Empty Scenes and Placing Images

-

Expressions and Data Types

HOMEWORK

- Problems: 27–30
- Quiz: Problem 31 (due in 1 week)
- On take-home quizzes always work in groups of 2–4 (mandatory)

The Nature of Functions

- What for? Have you ever wondered where functions come from? Why do they exist?

The Nature of Functions

- What for? Have you ever wondered where functions come from? Why do they exist?
- Consider the following function:

  $$f(x) = x^2 + 3x - 10$$

- `x` is an input element from the domain
- The value of `f(10)` is 120

The Nature of Functions

- What for? Have you ever wondered where functions come from? Why do they exist?
- Consider the following function:

  $f(x) = x^2 + 3x - 10$

- `x` is an input element from the domain
- The value of `f(10)` is 120
- Use functions to compute values by "plugging in":

  $f(10) = 10^2 + 3 \cdot 10 - 10$

- Now, it is all a matter of evaluating function's body
- This is screaming to us that functions are likely to play an important role in programming:

```
;; Evaluating f(x) = x^2 + 3x - 10
(define F-X10 (+ (sqr 10) (* 3 10) -10))
(check-expect F-X10 120)
```

- We still, however, do not know where `f(x)` came from.

The Nature of Functions

The Rise of Functions

- Also have your program compute `f(0)`

The Nature of Functions

The Rise of Functions

- Also have your program compute `f(0)`
- and `f(1)`

The Nature of Functions

The Rise of Functions

- Also have your program compute `f(0)`
- and `f(1)`
- and `f(20)`

The Nature of Functions

The Rise of Functions

- Also have your program compute `f(0)`
- and `f(1)`
- and `f(20)`
- and `f(100)`

The Nature of Functions

The Rise of Functions

- Also have your program compute f(0)
- and f(1)
- and f(20)
- and f(100)
-

```
;; Evaluating f(x) f(x) = x^2 + 3x - 10

(define F-X10  (+ (sqr 10)  (* 3 10)  -10))
(define F-X0   (+ (sqr 0)   (* 3 0)   -10))
(define F-X1   (+ (sqr 1)   (* 3 1)   -10))
(define F-X20  (+ (sqr 20)  (* 3 20)  -10))
(define F-X100 (+ (sqr 100) (* 3 100) -10))

(check-expect F-X10  120)
(check-expect F-X0   -10)
(check-expect F-X1   -6)
(check-expect F-X20  450)
(check-expect F-X100 10290)
```

The Nature of Functions

The Rise of Functions

- Also have your program compute `f(0)`
- and `f(1)`
- and `f(20)`
- and `f(100)`
-
```
;; Evaluating f(x) f(x) = x^2 + 3x - 10

(define F-X10  (+ (sqr 10)  (* 3 10)  -10))
(define F-X0   (+ (sqr 0)   (* 3 0)   -10))
(define F-X1   (+ (sqr 1)   (* 3 1)   -10))
(define F-X20  (+ (sqr 20)  (* 3 20)  -10))
(define F-X100 (+ (sqr 100) (* 3 100) -10))

(check-expect F-X10  120)
(check-expect F-X0   -10)
(check-expect F-X1   -6)
(check-expect F-X20  450)
(check-expect F-X100 10290)
```
- Compute all the values of `f(x)` for the integers in `[11..19]`
- A lot of the same typing over and over again
- There must be a better way to get the job done.

The Nature of Functions

The Rise of Functions

- ```
  (define F-X10  (+ (sqr 10)  (* 3 10)  -10))
  (define F-X0   (+ (sqr 0)   (* 3 0)   -10))
  (define F-X1   (+ (sqr 1)   (* 3 1)   -10))
  (define F-X20  (+ (sqr 20)  (* 3 20)  -10))
  (define F-X100 (+ (sqr 100) (* 3 100) -10))
  ```
- Expressions are almost identical
- One difference from one expression to the next: the number plugged in
  ```

The Nature of Functions

The Rise of Functions

- ```
  (define F-X10  (+ (sqr 10)  (* 3 10)  -10))
  (define F-X0   (+ (sqr 0)   (* 3 0)   -10))
  (define F-X1   (+ (sqr 1)   (* 3 1)   -10))
  (define F-X20  (+ (sqr 20)  (* 3 20)  -10))
  (define F-X100 (+ (sqr 100) (* 3 100) -10))
  ```
- Expressions are almost identical
- One difference from one expression to the next: the number plugged in
- The elements that vary among similar expressions are called *variables*
- Abstraction: associate each difference with a variable
  ```

The Nature of Functions

The Rise of Functions

- ```
  (define F-X10  (+ (sqr 10)  (* 3 10)  -10))
  (define F-X0   (+ (sqr 0)   (* 3 0)   -10))
  (define F-X1   (+ (sqr 1)   (* 3 1)   -10))
  (define F-X20  (+ (sqr 20)  (* 3 20)  -10))
  (define F-X100 (+ (sqr 100) (* 3 100) -10))
  ```
- Expressions are almost identical
- One difference from one expression to the next: the number plugged in
- The elements that vary among similar expressions are called *variables*
- Abstraction: associate each difference with a variable
- Let's call the single difference `x`
- Reduces all the expressions above to a single expression:

  ```
  (+ (sqr x) (* 3 x) -10))
  ```
  ```

The Nature of Functions

The Rise of Functions

- ```
  (define F-X10  (+ (sqr 10)  (* 3 10)  -10))
  (define F-X0   (+ (sqr 0)   (* 3 0)   -10))
  (define F-X1   (+ (sqr 1)   (* 3 1)   -10))
  (define F-X20  (+ (sqr 20)  (* 3 20)  -10))
  (define F-X100 (+ (sqr 100) (* 3 100) -10))
  ```
- Expressions are almost identical
- One difference from one expression to the next: the number plugged in
- The elements that vary among similar expressions are called *variables*
- Abstraction: associate each difference with a variable
- Let's call the single difference `x`
- Reduces all the expressions above to a single expression:

  ```
  (+ (sqr x) (* 3 x) -10))
  ```
- A mechanism is needed to provide `x` with a value
- Mathematicians the syntax `f(x) = e`, where `e` is a mathematical expression
- `f(x)` is the function `header`: names of function and input
- The input variables are called the function's *parameters*
- `e` is the function body
  ```

The Nature of Functions

The Rise of Functions

- ```
  (define F-X10  (+ (sqr 10)  (* 3 10)  -10))
  (define F-X0   (+ (sqr 0)   (* 3 0)   -10))
  (define F-X1   (+ (sqr 1)   (* 3 1)   -10))
  (define F-X20  (+ (sqr 20)  (* 3 20)  -10))
  (define F-X100 (+ (sqr 100) (* 3 100) -10))
  ```
- Expressions are almost identical
- One difference from one expression to the next: the number plugged in
- The elements that vary among similar expressions are called *variables*
- Abstraction: associate each difference with a variable
- Let's call the single difference `x`
- Reduces all the expressions above to a single expression:

  ```
  (+ (sqr x) (* 3 x) -10))
  ```
- A mechanism is needed to provide `x` with a value
- Mathematicians the syntax `f(x) = e`, where `e` is a mathematical expression
- `f(x)` is the function `header`: names of function and input
- The input variables are called the function's *parameters*
- `e` is the function body
- Mathematicians use a mathematical application expression to *bind* parameters to values: `f(10)` binds `x` to 10
- The process of substituting every variable with its binding is called *β-reduction*
- The β-reduction transformation yields an expression that has no variables and may be evaluated.
  ```

The Nature of Functions

The Rise of Functions

- BSL allows programmers to write their own functions much like mathematicians do
- To allow programmers to do so, there is a new `def` production rule:

  `def` ::= (define (<name> <name>$^+$) `expr`)
- Header: (<name> <name>$^+$)
- Body: `expr`

The Nature of Functions

The Rise of Functions

- BSL allows programmers to write their own functions much like mathematicians do
- To allow programmers to do so, there is a new `def` production rule:

  `def` ::= (define (<name> <name>$^+$) `expr`)
- Header: (<name> <name>$^+$)
- Body: `expr`
- It is not reasonable to assume that anyone will know how to correctly use a function just by looking at it
- Any reader or user of a function needs to know the types of the parameters, the type of the value returned, and the purpose of the function
- Functions need to be documented with a signature and a purpose statement

The Nature of Functions

The Rise of Functions

- BSL allows programmers to write their own functions much like mathematicians do
- To allow programmers to do so, there is a new `def` production rule:

  `def` ::= (define (<name> <name>$^+$) `expr`)
- Header: (<name> <name>$^+$)
- Body: `expr`
- It is not reasonable to assume that anyone will know how to correctly use a function just by looking at it
- Any reader or user of a function needs to know the types of the parameters, the type of the value returned, and the purpose of the function
- Functions need to be documented with a signature and a purpose statement
- `x`'s type is number
- Type returned by the function is number
- The signature is `number` $\rightarrow$ `number`
- The purpose is to compute the value of `f(x)`.

The Nature of Functions

The Rise of Functions

```
;; Evaluating f(x) f(x) = x^2 + 3x - 10
(define F-X10  (+ (sqr 10)  (* 3 10)  -10))
(define F-X0   (+ (sqr 0)   (* 3 0)   -10))
(define F-X1   (+ (sqr 1)   (* 3 1)   -10))
(define F-X20  (+ (sqr 20)  (* 3 20)  -10))
(define F-X100 (+ (sqr 100) (* 3 100) -10))

(check-expect F-X10  120)   (check-expect F-X0   -10)
(check-expect F-X1   -6)    (check-expect F-X20  450)
(check-expect F-X100 10290) (check-expect (f 11) 144)
(check-expect (f 12) 170)   (check-expect (f 13) 198)
(check-expect (f 14) 228)   (check-expect (f 15) 260)
(check-expect (f 16) 294)   (check-expect (f 17) 330)
(check-expect (f 18) 368)   (check-expect (f 19) 408)

;; number → number
;; Purpose: To compute f(x)
(define (f x)
  (+ (sqr x) (* 3 x) -10))
```

The Nature of Functions

The Rise of Functions

```
;; Evaluating f(x) f(x) = x^2 + 3x - 10
(define F-X10  (+ (sqr 10)  (* 3 10)  -10))
(define F-X0   (+ (sqr 0)   (* 3 0)   -10))
(define F-X1   (+ (sqr 1)   (* 3 1)   -10))
(define F-X20  (+ (sqr 20)  (* 3 20)  -10))
(define F-X100 (+ (sqr 100) (* 3 100) -10))

(check-expect F-X10  120)   (check-expect F-X0   -10)
(check-expect F-X1   -6)    (check-expect F-X20  450)
(check-expect F-X100 10290) (check-expect (f 11) 144)
(check-expect (f 12) 170)   (check-expect (f 13) 198)
(check-expect (f 14) 228)   (check-expect (f 15) 260)
(check-expect (f 16) 294)   (check-expect (f 17) 330)
(check-expect (f 18) 368)   (check-expect (f 19) 408)

;; number → number
;; Purpose: To compute f(x)
(define (f x)
  (+ (sqr x) (* 3 x) -10))
```

- For the tests, do you prefer to use `f` or always use defined constants?
- Can this program compute any value of `f(x)`?
- Instead of asking you to modify the program for every value of `f(x)` needed, can others use the program to compute these values?

The Nature of Functions

General Design Recipe for Functions

- Abstraction over expressions led to functions
- How do we systematically develop functions to solve problems?

The Nature of Functions

General Design Recipe for Functions

- Abstraction over expressions led to functions
- How do we systematically develop functions to solve problems?
- The Design Recipe
  1. Outline the representation of values and the computation.
  2. Define constants for the value of sample expressions.
  3. Identify and name the differences among the sample expressions.
  4. Write the function's signature and purpose.
  5. Write the function's header.
  6. Write tests.
  7. Write the function's body.
  8. Run the tests and, if necessary, redesign.

The Nature of Functions

General Design Recipe for Functions

- The design recipe suggests a *function template* may be used for function design
- A function template is like a skeleton for functions. It captures the main components a problem solution needs
- Every time a problem is being solved, the template is copied and specialized following the design recipe

The Nature of Functions

General Design Recipe for Functions

- The design recipe suggests a *function template* may be used for function design
- A function template is like a skeleton for functions. It captures the main components a problem solution needs
- Every time a problem is being solved, the template is copied and specialized following the design recipe
-

```
#|    ;; type+ → type    Purpose:
      (define (f-on-args var+)
        ...)

      ;; Sample expression definitions for f-on-args
      (define CONSTANT-1 expr)
           ...
      (define CONSTANT-N expr)

      ;; Tests using sample computations for f-on-args
      (check-expect (f-on-args expr+) CONSTANT-1)
           ...
      (check-expect (f-on-args expr+) CONSTANT-N)
      ;; Tests using sample values for f-on-args
      (check-expect (f-on-args expr+) value-1)
           ...
      (check-expect (f-on-args expr+) value-m)      |#
```

The Nature of Functions

The Design Recipe in Action

- Consider the problem of creating name-banner images for Craig, Julia (Ohio), Steve, Little Nick, Big Nick, and Jeremy
- The name image should contain only the name in a green 36 size font

The Nature of Functions

The Design Recipe in Action

- Consider the problem of creating name-banner images for Craig, Julia (Ohio), Steve, Little Nick, Big Nick, and Jeremy
- The name image should contain only the name in a green 36 size font
- The answers to each step of the design recipe:

The Nature of Functions

The Design Recipe in Action

- Consider the problem of creating name-banner images for Craig, Julia (Ohio), Steve, Little Nick, Big Nick, and Jeremy
- The name image should contain only the name in a green 36 size font
- The answers to each step of the design recipe:

  STEP 1 Represent each name as a string. Compute banner by applying the `text` to a name, 36, and 'olive.

The Nature of Functions

The Design Recipe in Action

- Consider the problem of creating name-banner images for Craig, Julia (Ohio), Steve, Little Nick, Big Nick, and Jeremy
- The name image should contain only the name in a green 36 size font
- The answers to each step of the design recipe:

  STEP 1 Represent each name as a string. Compute banner by applying the `text` to a name, 36, and 'olive.

  STEP 2 Three constants for the value of sample expressions are:

```
(define CRAIG (text "Craig"        36 'olive))
(define JULIA (text "Julia (Ohio)" 36 'olive))
(define STEVE (text "Steve"        36 'olive))
```

The Nature of Functions

The Design Recipe in Action

- Consider the problem of creating name-banner images for Craig, Julia (Ohio), Steve, Little Nick, Big Nick, and Jeremy
- The name image should contain only the name in a green 36 size font
- The answers to each step of the design recipe:

  STEP 1 Represent each name as a string. Compute banner by applying the `text` to a name, 36, and 'olive.

  STEP 2 Three constants for the value of sample expressions are:

```
(define CRAIG (text "Craig"        36 'olive))
(define JULIA (text "Julia (Ohio)" 36 'olive))
(define STEVE (text "Steve"        36 'olive))
```

  STEP 3 Only difference among the three sample expressions is the string representing the name: variable is `name`.

The Nature of Functions

The Design Recipe in Action

- Consider the problem of creating name-banner images for Craig, Julia (Ohio), Steve, Little Nick, Big Nick, and Jeremy
- The name image should contain only the name in a green 36 size font
- The answers to each step of the design recipe:

  STEP 1 Represent each name as a string. Compute banner by applying the `text` to a name, 36, and 'olive.

  STEP 2 Three constants for the value of sample expressions are:

```
(define CRAIG (text "Craig"        36 'olive))
(define JULIA (text "Julia (Ohio)" 36 'olive))
(define STEVE (text "Steve"        36 'olive))
```

  STEP 3 Only difference among the three sample expressions is the string representing the name: variable is `name`.

  STEP 4 Signature and purpose statement are:

```
;; string → image  Purpose: Create name banner
```

The Nature of Functions

The Design Recipe in Action

- Consider the problem of creating name-banner images for Craig, Julia (Ohio), Steve, Little Nick, Big Nick, and Jeremy
- The name image should contain only the name in a green 36 size font
- The answers to each step of the design recipe:

STEP 1 Represent each name as a string. Compute banner by applying the `text` to a name, 36, and 'olive.

STEP 2 Three constants for the value of sample expressions are:

```
(define CRAIG (text "Craig"        36 'olive))
(define JULIA (text "Julia (Ohio)" 36 'olive))
(define STEVE (text "Steve"        36 'olive))
```

STEP 3 Only difference among the three sample expressions is the string representing the name: variable is `name`.

STEP 4 Signature and purpose statement are:

```
;; string → image  Purpose: Create name banner
```

STEP 5 Function header: `(define (make-banner name)`

The Nature of Functions

The Design Recipe in Action

STEP 6 Tests

```
;; Tests using sample computations
(check-expect (make-banner "Craig")          CRAIG)
(check-expect (make-banner "Julia (Ohio)")   JULIA)
(check-expect (make-banner "Steve")          STEVE)
;; Tests using sample values
(check-expect (make-banner "Little Nick") Little Nick )
(check-expect (make-banner "Big Nick") Big Nick)
(check-expect (make-banner "Jeremy") Jeremy )
```

The Nature of Functions

The Design Recipe in Action

STEP 6 Tests

```
;; Tests using sample computations
(check-expect (make-banner "Craig")          CRAIG)
(check-expect (make-banner "Julia (Ohio)")   JULIA)
(check-expect (make-banner "Steve")          STEVE)
;; Tests using sample values
(check-expect (make-banner "Little Nick") Little Nick )
(check-expect (make-banner "Big Nick") Big Nick)
(check-expect (make-banner "Jeremy") Jeremy )
```

STEP 7 Function body: `(text name 36 'olive)`

The Nature of Functions

The Design Recipe in Action

STEP 6 Tests

```
;; Tests using sample computations
(check-expect (make-banner "Craig")          CRAIG)
(check-expect (make-banner "Julia (Ohio)")   JULIA)
(check-expect (make-banner "Steve")          STEVE)
;; Tests using sample values
(check-expect (make-banner "Little Nick") Little Nick )
(check-expect (make-banner "Big Nick") Big Nick)
(check-expect (make-banner "Jeremy") Jeremy )
```

STEP 7 Function body: `(text name 36 'olive)`

STEP 8 All tests pass.

The Nature of Functions

The Design Recipe in Action

```
(require 2htdp/image)
;; string → image
;; Purpose: Create a banner for the give name using
;;          font 36 and the color olive
(define (make-banner name)
  (text name 36 'olive))

; Sample expression definitions for make-banner
(define CRAIG  (text "Craig"        36 'olive))
(define JULIA  (text "Julia (Ohio)" 36 'olive))
(define STEVE  (text "Steve"         36 'olive))

;; Tests using sample computations for make-banner
(check-expect (make-banner "Craig")         CRAIG)
(check-expect (make-banner "Julia (Ohio)")  JULIA)
(check-expect (make-banner "Steve")         STEVE)

;; Tests using sample values for make-banner
(check-expect (make-banner "Little Nick") Little Nick )
(check-expect (make-banner "Big Nick")    Big Nick )
(check-expect (make-banner "Jeremy")      Jeremy )
```

The Nature of Functions

HOMEWORK

- Problems: 32–35

The Nature of Functions

Auxiliary Functions

- Seldomly, programs only need to compute many instances of a single value
- Far more common to have to compute several different values

The Nature of Functions

Auxiliary Functions

- Seldomly, programs only need to compute many instances of a single value
- Far more common to have to compute several different values
- A good designer develops a function for each different value
- Makes it easier to understand the solution to the problem
- Endows programs with *modularity*
- Modularity enables re-usability of expressions and reduces code duplication

The Nature of Functions

Auxiliary Functions

- Seldomly, programs only need to compute many instances of a single value
- Far more common to have to compute several different values
- A good designer develops a function for each different value
- Makes it easier to understand the solution to the problem
- Endows programs with *modularity*
- Modularity enables re-usability of expressions and reduces code duplication
- If problem analysis reveals that different kinds of values need to be computed, consider which values ought to be computed by an *auxiliary function*
- If special knowledge is required to compute a value or if different instances of the same value are needed then designing an auxiliary function is a good choice

The Nature of Functions

Auxiliary Functions

- Seldomly, programs only need to compute many instances of a single value
- Far more common to have to compute several different values
- A good designer develops a function for each different value
- Makes it easier to understand the solution to the problem
- Endows programs with *modularity*
- Modularity enables re-usability of expressions and reduces code duplication
- If problem analysis reveals that different kinds of values need to be computed, consider which values ought to be computed by an *auxiliary function*
- If special knowledge is required to compute a value or if different instances of the same value are needed then designing an auxiliary function is a good choice
- Two basic strategies: *bottom-up* or *top-down*
- Bottom-up is a programming style in which the simplest functions are designed first and then these functions are used to design more complex functions
- Top-down is a programming style in which the complex functions are designed first and then simpler functions are designed

The Nature of Functions

Bottom-up Design

- Consider computing the area of a washer
- To compute the area of a washer we need to compute the area of a circle

The Nature of Functions

Bottom-up Design

- Consider computing the area of a washer
- To compute the area of a washer we need to compute the area of a circle
- Start with computing the area of a circle
- Step 1: Problem Analysis

```
The area of a circle is given by the product of π and
the square of the circle's radius
```

The Nature of Functions

Bottom-up Design

- ```
  ;; Sample expression definitions for area-circle
  (define C-AREA5  (* pi (sqr 5)))
  (define C-AREA18 (* pi (sqr 18)))
  ```
  ```

The Nature of Functions

Bottom-up Design

- ```
  ;; ℝ>= 0 → Real>= 0
  ;; Purpose: Compute the area of the circle with
  ;;          the given radius.
  (define (area-circle a-radius)
  ```

- ```
  ;; Sample expression definitions for area-circle
  (define C-AREA5  (* pi (sqr 5)))
  (define C-AREA18 (* pi (sqr 18)))
  ```

The Nature of Functions

Bottom-up Design

- ```
  ;; ℝ>= 0 → Real>= 0
  ;; Purpose: Compute the area of the circle with
  ;;          the given radius.
  (define (area-circle a-radius)
  ```

- ```
  ;; Sample expression definitions for area-circle
  (define C-AREA5  (* pi (sqr 5)))
  (define C-AREA18 (* pi (sqr 18)))
  ```

- ```
  ;; Tests using sample computations for area-circle
  (check-within (area-circle 5)  C-AREA5  0.01)
  (check-within (area-circle 18) C-AREA18 0.01)

  ;; Tests using sample values for area-circle
  (check-within (area-circle 0)  0      0.01)
  (check-within (area-circle 10) 314.15 0.01)
  ```
  ```

The Nature of Functions

Bottom-up Design

```
;; ℝ>= 0 → Real>= 0
;; Purpose: Compute the area of the circle with
;;          the given radius.
(define (area-circle a-radius)
  (* pi (sqr a-radius)))

;; Sample expression definitions for area-circle
(define C-AREA5  (* pi (sqr 5)))
(define C-AREA18 (* pi (sqr 18)))

;; Tests using sample computations for area-circle
(check-within (area-circle 5)  C-AREA5  0.01)
(check-within (area-circle 18) C-AREA18 0.01)

;; Tests using sample values for area-circle
(check-within (area-circle 0)  0      0.01)
(check-within (area-circle 10) 314.15 0.01)
```

The Nature of Functions

General Design Recipe for Functions

- Now onto the area of a washer

The Nature of Functions

General Design Recipe for Functions

- Now onto the area of a washer
- Step 1: Problem Analysis

```
The area of a washer is given by the area difference
of its outer and inner circle
```

The Nature of Functions

The Rise of Functions

- ```
  ;; Sample expression definitions for area-washer
  (define W-AREA6-3 (- (area-circle 6) (area-circle 3)))
  (define W-AREA15-9 (- (area-circle 15) (area-circle 9)))
  (define W-AREA4-1 (- (area-circle 4) (area-circle 1)))
  ```
  ```

The Nature of Functions

The Rise of Functions

- ```
  ;; ℝ>=0 ℝ>=0 → Real>= 0
  ;; Purpose: Compute the area of the washer with
  ;;          the given radii.
  ;; ASSUMPTION: outer-radius >= inner-radius
  (define (area-washer outer-radius inner-radius)
  ```

- ```
  ;; Sample expression definitions for area-washer
  (define W-AREA6-3 (- (area-circle 6) (area-circle 3)))
  (define W-AREA15-9 (- (area-circle 15) (area-circle 9)))
  (define W-AREA4-1 (- (area-circle 4) (area-circle 1)))
  ```

The Nature of Functions

The Rise of Functions

- ```
  ;; ℝ>=0 ℝ>=0 → Real>= 0
  ;; Purpose: Compute the area of the washer with
  ;;          the given radii.
  ;; ASSUMPTION: outer-radius >= inner-radius
  (define (area-washer outer-radius inner-radius)
  ```

- ```
  ;; Sample expression definitions for area-washer
  (define W-AREA6-3 (- (area-circle 6) (area-circle 3)))
  (define W-AREA15-9 (- (area-circle 15) (area-circle 9)))
  (define W-AREA4-1 (- (area-circle 4) (area-circle 1)))
  ```

- ```
  ;; Tests using sample computations for area-washer
  (check-within (area-washer 6 3)  W-AREA6-3  0.01)
  (check-within (area-washer 15 9) W-AREA15-9 0.01)
  (check-within (area-washer 4 1)  W-AREA4-1  0.01)

  ;; Tests using sample values for area-washer
  (check-within (area-washer 0 0)  0  0.01)
  (check-within (area-washer 10 0)  314.15  0.01)
  (check-within (area-washer 7 4)  103.67  0.01)
  ```
  ```

The Nature of Functions

The Rise of Functions

```
;; ℝ>=0 ℝ>=0 → Real>= 0
;; Purpose: Compute the area of the washer with
;;          the given radii.
;; ASSUMPTION: outer-radius >= inner-radius
(define (area-washer outer-radius inner-radius)
  (- (area-circle outer-radius) (area-circle inner-radius)))
```

```
;; Sample expression definitions for area-washer
(define W-AREA6-3 (- (area-circle 6) (area-circle 3)))
(define W-AREA15-9 (- (area-circle 15) (area-circle 9)))
(define W-AREA4-1 (- (area-circle 4) (area-circle 1)))
```

```
;; Tests using sample computations for area-washer
(check-within (area-washer 6 3)  W-AREA6-3  0.01)
(check-within (area-washer 15 9) W-AREA15-9 0.01)
(check-within (area-washer 4 1)  W-AREA4-1  0.01)

;; Tests using sample values for area-washer
(check-within (area-washer 0 0)  0  0.01)
(check-within (area-washer 10 0)  314.15  0.01)
(check-within (area-washer 7 4)  103.67  0.01)
```

The Nature of Functions

HOMEWORK

- Problems: 36–39

The Nature of Functions

Top-down Design

- Consider the problem of computing images like above
- 4 overlayed squares at a 45 degree angle

The Nature of Functions

Top-down Design

- Consider the problem of computing images like above
- 4 overlayed squares at a 45 degree angle
- Start by writing expressions to compute the image composition
- For unknown values, like the length of a square, use a constant

The Nature of Functions

Top-down Design

- Consider the problem of computing images like above
- 4 overlayed squares at a 45 degree angle
- Start by writing expressions to compute the image composition
- For unknown values, like the length of a square, use a constant
- STEP 1: The image is computed by overlaying squares of alternating colors and differing lengths. Every other overlayed square, starting with the largest square, is rotated by 45 degrees

The Nature of Functions

Top-down Design

- ```
  (define IMG1 (overlay (rotate 0  (make-sqr IMG1-LEN4 'brown))
                        (rotate 45 (make-sqr IMG1-LEN3 'gold))
                        (rotate 0  (make-sqr IMG1-LEN2 'brown))
                        (rotate 45 (make-sqr IMG1-LEN1 'gold))))
  (define IMG2 (overlay (rotate 0  (make-sqr IMG2-LEN4 'black))
                        (rotate 45 (make-sqr IMG2-LEN3 'red))
                        (rotate 0  (make-sqr IMG2-LEN2 'black))
                        (rotate 45 (make-sqr IMG2-LEN1 'red))))
  ```
  ```

The Nature of Functions

Top-down Design

- ;; $\mathbb{R}_{>=}$ 0 $\mathbb{R}_{>=}$ 0 $\mathbb{R}_{>=}$ 0 $\mathbb{R}_{>=}$ 0 color color → image
  ;; Purpose: Create nested squares using the given lengths and colors

- ```
  (define IMG1 (overlay (rotate 0  (make-sqr IMG1-LEN4 'brown))
                        (rotate 45 (make-sqr IMG1-LEN3 'gold))
                        (rotate 0  (make-sqr IMG1-LEN2 'brown))
                        (rotate 45 (make-sqr IMG1-LEN1 'gold))))
  (define IMG2 (overlay (rotate 0  (make-sqr IMG2-LEN4 'black))
                        (rotate 45 (make-sqr IMG2-LEN3 'red))
                        (rotate 0  (make-sqr IMG2-LEN2 'black))
                        (rotate 45 (make-sqr IMG2-LEN1 'red))))
  ```
  ```

The Nature of Functions

Top-down Design

- ;; $\mathbb{R}_{>=}$ 0 $\mathbb{R}_{>=}$ 0 $\mathbb{R}_{>=}$ 0 $\mathbb{R}_{>=}$ 0 color color $\rightarrow$ image
  ;; Purpose: Create nested squares using the given lengths and colors

- (define (make-nested-sqs bot-len len2 len3 top-len botclr topclr)

-
```
(define IMG1 (overlay (rotate 0  (make-sqr IMG1-LEN4 'brown))
                      (rotate 45 (make-sqr IMG1-LEN3 'gold))
                      (rotate 0  (make-sqr IMG1-LEN2 'brown))
                      (rotate 45 (make-sqr IMG1-LEN1 'gold))))
(define IMG2 (overlay (rotate 0  (make-sqr IMG2-LEN4 'black))
                      (rotate 45 (make-sqr IMG2-LEN3 'red))
                      (rotate 0  (make-sqr IMG2-LEN2 'black))
                      (rotate 45 (make-sqr IMG2-LEN1 'red))))
```

The Nature of Functions

Top-down Design

- ```
  ;; ℝ>= 0 ℝ>= 0 ℝ>= 0 ℝ>= 0 color color → image
  ;; Purpose: Create nested squares using the given lengths and colors
  ```

- ```
  (define (make-nested-sqs bot-len len2 len3 top-len botclr  topclr)
  ```

- ```
  (define IMG1 (overlay (rotate 0  (make-sqr IMG1-LEN4 'brown))
                        (rotate 45 (make-sqr IMG1-LEN3 'gold))
                        (rotate 0  (make-sqr IMG1-LEN2 'brown))
                        (rotate 45 (make-sqr IMG1-LEN1 'gold))))
  (define IMG2 (overlay (rotate 0  (make-sqr IMG2-LEN4 'black))
                        (rotate 45 (make-sqr IMG2-LEN3 'red))
                        (rotate 0  (make-sqr IMG2-LEN2 'black))
                        (rotate 45 (make-sqr IMG2-LEN1 'red))))
  ```

- ```
  ;; Tests using sample computations
  (check-expect (make-nested-sqs IMG1-LEN1 IMG1-LEN2
                                 IMG1-LEN3 IMG1-LEN4
                                 'gold     'brown) IMG1)
  (check-expect (make-nested-sqs IMG2-LEN1 IMG2-LEN2
                                 IMG2-LEN3 IMG2-LEN4
                                 'red      'black) IMG2)
  ;; Tests using sample values
  ```

```
(check-expect (make-nested-sqs 50 35.35 24.99 17.67 'olive 'yellow)  )
```

```
(check-expect (make-nested-sqs 20 14.14  9.99  7.06 'pink  'purple)  )
```

The Nature of Functions

Top-down Design

```
;; ℝ>= 0 ℝ>= 0 ℝ>= 0 ℝ>= 0 color color → image
;; Purpose: Create nested squares using the given lengths and colors

(define (make-nested-sqs bot-len len2 len3 top-len botclr  topclr)
  (overlay (rotate 0  (make-sqr top-len topclr))
           (rotate 45 (make-sqr len3 botclr))
           (rotate 0  (make-sqr len2 topclr))
           (rotate 45 (make-sqr bot-len botclr)))

  (define IMG1 (overlay (rotate 0  (make-sqr IMG1-LEN4 'brown))
                        (rotate 45 (make-sqr IMG1-LEN3 'gold))
                        (rotate 0  (make-sqr IMG1-LEN2 'brown))
                        (rotate 45 (make-sqr IMG1-LEN1 'gold))))
  (define IMG2 (overlay (rotate 0  (make-sqr IMG2-LEN4 'black))
                        (rotate 45 (make-sqr IMG2-LEN3 'red))
                        (rotate 0  (make-sqr IMG2-LEN2 'black))
                        (rotate 45 (make-sqr IMG2-LEN1 'red))))

  ;; Tests using sample computations
  (check-expect (make-nested-sqs IMG1-LEN1 IMG1-LEN2
                                 IMG1-LEN3 IMG1-LEN4
                                 'gold     'brown) IMG1)
  (check-expect (make-nested-sqs IMG2-LEN1 IMG2-LEN2
                                 IMG2-LEN3 IMG2-LEN4
                                 'red      'black) IMG2)
  ;; Tests using sample values


  (check-expect (make-nested-sqs 50 35.35 24.99 17.67 'olive 'yellow)  )

  (check-expect (make-nested-sqs 20 14.14  9.99  7.06 'pink  'purple)  )
```

The Nature of Functions

Top-down Design

- Cannot run any tests because `make-sqr` and the length constants are undefined

The Nature of Functions

Top-down Design

- Cannot run any tests because `make-sqr` and the length constants are undefined
- Need: side length and color

The Nature of Functions

Top-down Design

- Cannot run any tests because `make-sqr` and the length constants are undefined
- Need: side length and color
- STEP 1: The image of a solid square is computed using the `square` function and the given side length and color

The Nature of Functions

Top-down Design

- ```
  (define SQ1 (square 500 'solid 'lightbrown))
  (define SQ2 (square 70  'solid 'darkblue))
  ```
  ```

The Nature of Functions

Top-down Design

- Notice typos in the textbook.

```
;; ℝ>= 0 color → image
;; Purpose: Compute the image of a square of the given
;;          length and color
```

-

```
(define SQ1 (square 500 'solid 'lightbrown))
(define SQ2 (square 70  'solid 'darkblue))
```

The Nature of Functions

Top-down Design

- Notice typos in the textbook.

```
;; ℝ>= 0 color → image
;; Purpose: Compute the image of a square of the given
;;          length and color
```

- ```
  (define (make-sqr side-len a-color)
  ```

- ```
  (define SQ1 (square 500 'solid 'lightbrown))
  (define SQ2 (square 70  'solid 'darkblue))
  ```

The Nature of Functions

Top-down Design

- Notice typos in the textbook.

```
;; ℝ>= 0 color → image
;; Purpose: Compute the image of a square of the given
;;          length and color
```

- `(define (make-sqr side-len a-color)`

- `(define SQ1 (square 500 'solid 'lightbrown))`
  `(define SQ2 (square 70  'solid 'darkblue))`

- `;; Tests using sample computations`
  `(check-expect (make-sqr 500 'lightbrown) SQ1)`
  `(check-expect (make-sqr 70  'darkblue) SQ2)`

  `;; Tests using sample values`

  `(check-expect (make-sqr 12 'orange)` `)`

  `(check-expect (make-sqr 35 'skyblue)` `)`

The Nature of Functions

Top-down Design

- Notice typos in the textbook.

```
;; ℝ>= 0 color → image
;; Purpose: Compute the image of a square of the given
;;          length and color
```

- `(define (make-sqr side-len a-color)`
- `  (square side-len 'solid a-color)`
- `(define SQ1 (square 500 'solid 'lightbrown))`
  `(define SQ2 (square 70  'solid 'darkblue))`
- `;; Tests using sample computations`
  `(check-expect (make-sqr 500 'lightbrown) SQ1)`
  `(check-expect (make-sqr 70  'darkblue) SQ2)`

```
;; Tests using sample values

(check-expect (make-sqr 12 'orange)  )

(check-expect (make-sqr 35 'skyblue)  )
```

The Nature of Functions

Top-down Design

- Next define the constants for the length

The Nature of Functions

Top-down Design

- Next define the constants for the length
- The length of the largest square is arbitrary

The Nature of Functions

Top-down Design

- Next define the constants for the length
- The length of the largest square is arbitrary
- The length of the next smaller square depends on the length of the outer square
-

- How do we compute the length of the next smaller square?

The Nature of Functions

Top-down Design

- Next define the constants for the length
- The length of the largest square is arbitrary
- The length of the next smaller square depends on the length of the outer square
-

- How do we compute the length of the next smaller square?
- The length of the inner square is the length of the hypotenuse of the right triangle:

$$x^2 = (\frac{w}{2})^2 + (\frac{w}{2})^2$$

$$x = \sqrt{(\frac{w}{2})^2 + (\frac{w}{2})^2}$$

$$= \sqrt{\frac{w^2}{2}}$$

The Nature of Functions

Top-down Design

- Next define the constants for the length
- The length of the largest square is arbitrary
- The length of the next smaller square depends on the length of the outer square
-

- How do we compute the length of the next smaller square?
- The length of the inner square is the length of the hypotenuse of the right triangle:

$$x^2 = (\frac{w}{2})^2 + (\frac{w}{2})^2$$

$$x = \sqrt{(\frac{w}{2})^2 + (\frac{w}{2})^2}$$

$$= \sqrt{\frac{w^2}{2}}$$

- A function can be defined to compute the length of inner squares
- The only difference in computing one inner square length from another is the length of the outer square

The Nature of Functions

Top-down Design

- The length of the next smaller square is given by $\sqrt{\frac{w^2}{2}}$

The Nature of Functions

Top-down Design

- The length of the next smaller square is given by $\sqrt{\frac{w^2}{2}}$

- ```
  (define IMG1-LEN1 100)
  (define IMG1-LEN2 (sqrt (/ (sqr IMG1-LEN1) 2)))
  (define IMG1-LEN3 (sqrt (/ (sqr IMG1-LEN2) 2)))
  (define IMG1-LEN4 (sqrt (/ (sqr IMG1-LEN3) 2)))
  ```
  ```

The Nature of Functions

Top-down Design

- The length of the next smaller square is given by $\sqrt{\frac{w^2}{2}}$
- ```
  ;; ℝ>= 0 → ℝ>= 0
  ;; Purpose: Compute the length of the next smallest square
  (define (compute-new-sqr-len side-len)
  ```
- ```
  (define IMG1-LEN1 100)
  (define IMG1-LEN2 (sqrt (/ (sqr IMG1-LEN1) 2)))
  (define IMG1-LEN3 (sqrt (/ (sqr IMG1-LEN2) 2)))
  (define IMG1-LEN4 (sqrt (/ (sqr IMG1-LEN3) 2)))
  ```

The Nature of Functions

Top-down Design

- The length of the next smaller square is given by $\sqrt{\frac{w^2}{2}}$
- ;; $\mathbb{R}_{>=\ 0} \rightarrow \mathbb{R}_{>=\ 0}$

```
;; Purpose: Compute the length of the next smallest square
(define (compute-new-sqr-len side-len)
```

-

```
(define IMG1-LEN1 100)
(define IMG1-LEN2 (sqrt (/ (sqr IMG1-LEN1) 2)))
(define IMG1-LEN3 (sqrt (/ (sqr IMG1-LEN2) 2)))
(define IMG1-LEN4 (sqrt (/ (sqr IMG1-LEN3) 2)))
```

-

```
;; Tests using sample computations
(check-within (compute-new-sqr-len 100) IMG1-LEN2 0.01)
(check-within (compute-new-sqr-len IMG1-LEN2) IMG1-LEN3 0.01)
(check-within (compute-new-sqr-len IMG1-LEN3) IMG1-LEN4 0.01)
;; Tests using sample values
(define IMG2-LEN1 150)
(define IMG2-LEN2 (compute-new-sqr-len IMG2-LEN1))
(define IMG2-LEN3 (compute-new-sqr-len IMG2-LEN2))
(define IMG2-LEN4 (compute-new-sqr-len IMG2-LEN3))
(check-within IMG2-LEN2 106.06 0.01)
(check-within IMG2-LEN3 75 0.01)
(check-within IMG2-LEN4 53.03 0.01)
(check-within (compute-new-sqr-len 0) 0    0.01)
(check-within (compute-new-sqr-len 8) 5.65 0.01)
```

The Nature of Functions

Top-down Design

- The length of the next smaller square is given by $\sqrt{\frac{w^2}{2}}$

```
;; ℝ>= 0 → ℝ>= 0
;; Purpose: Compute the length of the next smallest square
(define (compute-new-sqr-len side-len)
  (sqrt (/ (sqr side-len) 2))
(define IMG1-LEN1 100)
(define IMG1-LEN2 (sqrt (/ (sqr IMG1-LEN1) 2)))
(define IMG1-LEN3 (sqrt (/ (sqr IMG1-LEN2) 2)))
(define IMG1-LEN4 (sqrt (/ (sqr IMG1-LEN3) 2)))
;; Tests using sample computations
(check-within (compute-new-sqr-len 100) IMG1-LEN2 0.01)
(check-within (compute-new-sqr-len IMG1-LEN2) IMG1-LEN3 0.01)
(check-within (compute-new-sqr-len IMG1-LEN3) IMG1-LEN4 0.01)
;; Tests using sample values
(define IMG2-LEN1 150)
(define IMG2-LEN2 (compute-new-sqr-len IMG2-LEN1))
(define IMG2-LEN3 (compute-new-sqr-len IMG2-LEN2))
(define IMG2-LEN4 (compute-new-sqr-len IMG2-LEN3))
(check-within IMG2-LEN2 106.06 0.01)
(check-within IMG2-LEN3 75 0.01)
(check-within IMG2-LEN4 53.03 0.01)
(check-within (compute-new-sqr-len 0) 0    0.01)
(check-within (compute-new-sqr-len 8) 5.65 0.01)
```

The Nature of Functions

Top-down Design

- IMPORTANT:
- A defined function may be applied any time when it is part of a test
- All other times, however, a function must be defined before it is applied to any arguments
- Definition must appear in a program before its first use outside tests

The Nature of Functions

HOMEWORK

- Problems: 40–43

Aliens Attack Version 0

- Time to put your newly acquired skills to work
- Develop the first version of *Aliens Attack*
- Will not develop a full video game
- The video game is developed using the process of iterative refinement as you learn more about problem solving and about BSL syntax

Aliens Attack Version 0

- Time to put your newly acquired skills to work
- Develop the first version of *Aliens Attack*
- Will not develop a full video game
- The video game is developed using the process of iterative refinement as you learn more about problem solving and about BSL syntax
-
- Single player version of Aliens Attack

Aliens Attack Version 0

- Time to put your newly acquired skills to work
- Develop the first version of *Aliens Attack*
- Will not develop a full video game
- The video game is developed using the process of iterative refinement as you learn more about problem solving and about BSL syntax

- Single player version of Aliens Attack
- The development of Aliens Attack version 0 focuses on the creation and placing of images to render the video game as an image

Aliens Attack Version 0

The Scene for Aliens Attack

- How do we reason about this empty scene?

Aliens Attack Version 0

The Scene for Aliens Attack

- How do we reason about this empty scene?

(a) Pixel Perspective.

(b) Image Perspective.

-

- The blue square in is at pixel coordinate (15, 4)
- Can work well, but mismatches our goal to place images in a scene

Aliens Attack Version 0

The Scene for Aliens Attack

- How do we reason about this empty scene?

(a) Pixel Perspective.

(b) Image Perspective.

-

- The blue square in is at pixel coordinate (15, 4)
- Can work well, but mismatches our goal to place images in a scene
- If we choose a maximum size for each image in the video game, we can reason about the empty scene as boxes where an image fits
- The blue square in is at image coordinate (3,0)

Aliens Attack Version 0

The Scene for Aliens Attack

- Why should we care about the perspective we adopt?

Aliens Attack Version 0

The Scene for Aliens Attack

- Why should we care about the perspective we adopt?
- Data representation strongly influences how a problem is reasoned about.

Aliens Attack Version 0

The Scene for Aliens Attack

- Why should we care about the perspective we adopt?
- Data representation strongly influences how a problem is reasoned about.
- How is an alien hit by a shot detected using a pixel perspective?

Aliens Attack Version 0

The Scene for Aliens Attack

- Why should we care about the perspective we adopt?
- Data representation strongly influences how a problem is reasoned about.
- How is an alien hit by a shot detected using a pixel perspective?

-
```
(define HALF-ALIEN-IMG-WIDTH  (/ (image-width ALIEN-IMG) 2))
(define HALF-ALIEN-IMG-HEIGHT (/ (image-height ALIEN-IMG) 2))

(define (alien-hit? x-alien y-alien x-shot y-shot)
  (and (<= (abs (- x-alien x-shot)) HALF-ALIEN-IMG-WIDTH)
       (<= (abs (- y-alien y-shot)) HALF-ALIEN-IMG-HEIGHT)))
```

Aliens Attack Version 0

The Scene for Aliens Attack

- Consider solving the same problem using an image perspective of the game's scene

Aliens Attack Version 0

The Scene for Aliens Attack

- Consider solving the same problem using an image perspective of the game's scene
- Images can only be placed in a box with image coordinates `x` and `y`
- This means that an alien is hit by a shot if both have the same coordinates

Aliens Attack Version 0

The Scene for Aliens Attack

- Consider solving the same problem using an image perspective of the game's scene
- Images can only be placed in a box with image coordinates `x` and `y`
- This means that an alien is hit by a shot if both have the same coordinates
-
```
(define (alien-hit? x-alien y-alien x-shot y-shot)
  (and (= x-alien x-shot) (= y-alien y-shot)))
```
- Much simpler and easier to understand

Aliens Attack Version 0

The Scene for Aliens Attack

- Consider solving the same problem using an image perspective of the game's scene
- Images can only be placed in a box with image coordinates `x` and `y`
- This means that an alien is hit by a shot if both have the same coordinates
-
```
(define (alien-hit? x-alien y-alien x-shot y-shot)
  (and (= x-alien x-shot) (= y-alien y-shot)))
```
- Much simpler and easier to understand
- IMPORTANT LESSON: Different representations may lead to vastly different solutions/programs
- It is worth exploring different representations to determine if any lead to a simpler and easier to understand solution/program

Aliens Attack Version 0

The Scene for Aliens Attack

- We have carefully analyzed how to represent coordinates in the game's scene
- May be cautiously confident that using image coordinates leads to simpler problem solutions

Aliens Attack Version 0

The Scene for Aliens Attack

- We have carefully analyzed how to represent coordinates in the game's scene
- May be cautiously confident that using image coordinates leads to simpler problem solutions
- Need to be careful about defining the data representation in the program
- We need *data definitions*
- A data definition is written to describe a new type of data.

Aliens Attack Version 0

The Scene for Aliens Attack

- ```
  ;; Character Image Maximum Dimensions (in pixels)
  (define IMAGE-WIDTH 30)  (define IMAGE-HEIGHT 30)
  ```
  ```

Aliens Attack Version 0

The Scene for Aliens Attack

- ```
  ;; Character Image Maximum Dimensions (in pixels)
  (define IMAGE-WIDTH 30)  (define IMAGE-HEIGHT 30)
  ```
- ```
  ;; Video Game Scene Dimensions
  (define MAX-CHARS-HORIZONTAL 20)
  (define MAX-CHARS-VERTICAL 15)
  ```

Aliens Attack Version 0

The Scene for Aliens Attack

- ```
  ;; Character Image Maximum Dimensions (in pixels)
  (define IMAGE-WIDTH 30)  (define IMAGE-HEIGHT 30)
  ```
- ```
  ;; Video Game Scene Dimensions
  (define MAX-CHARS-HORIZONTAL 20)
  (define MAX-CHARS-VERTICAL 15)
  ```
- ```
  ;; Empty Scene Constants
  (define E-SCENE-COLOR 'pink) (define E-SCENE2-COLOR 'black)
  (define E-SCENE-W (* MAX-CHARS-HORIZONTAL IMAGE-WIDTH))
  (define E-SCENE-H (* MAX-CHARS-VERTICAL IMAGE-HEIGHT))
  ```
  ```

Aliens Attack Version 0

The Scene for Aliens Attack

- ```
  ;; Character Image Maximum Dimensions (in pixels)
  (define IMAGE-WIDTH 30)  (define IMAGE-HEIGHT 30)
  ```
- ```
  ;; Video Game Scene Dimensions
  (define MAX-CHARS-HORIZONTAL 20)
  (define MAX-CHARS-VERTICAL 15)
  ```
- ```
  ;; Empty Scene Constants
  (define E-SCENE-COLOR 'pink) (define E-SCENE2-COLOR 'black)
  (define E-SCENE-W (* MAX-CHARS-HORIZONTAL IMAGE-WIDTH))
  (define E-SCENE-H (* MAX-CHARS-VERTICAL IMAGE-HEIGHT))
  ```
- ```
  #|DATA DEFINITIONS
    A character image (ci) is an image which is at most
    IMAGE-WIDTH x IMAGE-HEIGHT pixels
    An image-x is an integer in [0..MAX-CHARS-HORIZONTAL-1]
    An image-y is an integer in [0..MAX-CHARS-VERTICAL-1]
    A scene is a E-SCENE-W x E-SCENE-H image
    A pixel-x is an integer in [0..E-SCENE-W-1]
    A pixel-y is an integer in [0..E-SCENE-H-1]        |#
  ```

Aliens Attack Version 0

The Scene for Aliens Attack

- ```
  ;; Character Image Maximum Dimensions (in pixels)
  (define IMAGE-WIDTH 30)  (define IMAGE-HEIGHT 30)
  ```
- ```
  ;; Video Game Scene Dimensions
  (define MAX-CHARS-HORIZONTAL 20)
  (define MAX-CHARS-VERTICAL 15)
  ```
- ```
  ;; Empty Scene Constants
  (define E-SCENE-COLOR 'pink) (define E-SCENE2-COLOR 'black)
  (define E-SCENE-W (* MAX-CHARS-HORIZONTAL IMAGE-WIDTH))
  (define E-SCENE-H (* MAX-CHARS-VERTICAL IMAGE-HEIGHT))
  ```
- ```
  #|DATA DEFINITIONS
    A character image (ci) is an image which is at most
    IMAGE-WIDTH x IMAGE-HEIGHT pixels
    An image-x is an integer in [0..MAX-CHARS-HORIZONTAL-1]
    An image-y is an integer in [0..MAX-CHARS-VERTICAL-1]
    A scene is a E-SCENE-W x E-SCENE-H image
    A pixel-x is an integer in [0..E-SCENE-W-1]
    A pixel-y is an integer in [0..E-SCENE-H-1]        |#
  ```
- ```
  ;; Sample image-x
  (define AN-IMG-X (/ MAX-CHARS-HORIZONTAL 2))
  (define MIN-IMG-X 0)
  (define MAX-IMG-X (sub1 MAX-CHARS-HORIZONTAL))
  ```
  ```

Aliens Attack Version 0

The Scene for Aliens Attack

- ```
  ;; Character Image Maximum Dimensions (in pixels)
  (define IMAGE-WIDTH 30)  (define IMAGE-HEIGHT 30)
  ```
- ```
  ;; Video Game Scene Dimensions
  (define MAX-CHARS-HORIZONTAL 20)
  (define MAX-CHARS-VERTICAL 15)
  ```
- ```
  ;; Empty Scene Constants
  (define E-SCENE-COLOR 'pink) (define E-SCENE2-COLOR 'black)
  (define E-SCENE-W (* MAX-CHARS-HORIZONTAL IMAGE-WIDTH))
  (define E-SCENE-H (* MAX-CHARS-VERTICAL IMAGE-HEIGHT))
  ```
- ```
  #|DATA DEFINITIONS
    A character image (ci) is an image which is at most
    IMAGE-WIDTH x IMAGE-HEIGHT pixels
    An image-x is an integer in [0..MAX-CHARS-HORIZONTAL-1]
    An image-y is an integer in [0..MAX-CHARS-VERTICAL-1]
    A scene is a E-SCENE-W x E-SCENE-H image
    A pixel-x is an integer in [0..E-SCENE-W-1]
    A pixel-y is an integer in [0..E-SCENE-H-1]        |#
  ```
- ```
  ;; Sample image-x
  (define AN-IMG-X (/ MAX-CHARS-HORIZONTAL 2))
  (define MIN-IMG-X 0)
  (define MAX-IMG-X (sub1 MAX-CHARS-HORIZONTAL))
  ```
- ```
  ;; Sample image-y
  (define AN-IMG-Y (/ MAX-CHARS-VERTICAL 2))
  (define MIN-IMG-Y 0)
  (define MAX-IMG-Y (sub1 MAX-CHARS-VERTICAL))
  ```

Aliens Attack Version 0

The Scene for Aliens Attack

- ```
  ;; Character Image Maximum Dimensions (in pixels)
  (define IMAGE-WIDTH 30)  (define IMAGE-HEIGHT 30)
  ```
- ```
  ;; Video Game Scene Dimensions
  (define MAX-CHARS-HORIZONTAL 20)
  (define MAX-CHARS-VERTICAL 15)
  ```
- ```
  ;; Empty Scene Constants
  (define E-SCENE-COLOR 'pink) (define E-SCENE2-COLOR 'black)
  (define E-SCENE-W (* MAX-CHARS-HORIZONTAL IMAGE-WIDTH))
  (define E-SCENE-H (* MAX-CHARS-VERTICAL IMAGE-HEIGHT))
  ```
- ```
  #|DATA DEFINITIONS
    A character image (ci) is an image which is at most
    IMAGE-WIDTH x IMAGE-HEIGHT pixels
    An image-x is an integer in [0..MAX-CHARS-HORIZONTAL-1]
    An image-y is an integer in [0..MAX-CHARS-VERTICAL-1]
    A scene is a E-SCENE-W x E-SCENE-H image
    A pixel-x is an integer in [0..E-SCENE-W-1]
    A pixel-y is an integer in [0..E-SCENE-H-1]        |#
  ```
- ```
  ;; Sample image-x
  (define AN-IMG-X (/ MAX-CHARS-HORIZONTAL 2))
  (define MIN-IMG-X 0)
  (define MAX-IMG-X (sub1 MAX-CHARS-HORIZONTAL))
  ```
- ```
  ;; Sample image-y
  (define AN-IMG-Y (/ MAX-CHARS-VERTICAL 2))
  (define MIN-IMG-Y 0)
  (define MAX-IMG-Y (sub1 MAX-CHARS-VERTICAL))
  ```
- ```
  ;; Sample empty scenes
  (define E-SCENE  (empty-scene E-SCENE-W E-SCENE-H E-SCENE-COLOR))
  (define E-SCENE2 (empty-scene E-SCENE-W E-SCENE-H E-SCENE2-COLOR))
  ```
  ```

Aliens Attack Version 0

Creating Aliens Attack Images

- Creating images is an opportunity to practice the steps of the design recipe
- It also provides us with the opportunity to write an *application programming interface* (API)
- An API defines the data formats, the conventions, and the functions that may be used to create and access data.

Aliens Attack Version 0

Creating Aliens Attack Images

- Creating images is an opportunity to practice the steps of the design recipe
- It also provides us with the opportunity to write an *application programming interface* (API)
- An API defines the data formats, the conventions, and the functions that may be used to create and access data.
- The Aliens Attack images API:
  - The dimensions of any character image must be 30 x 30 pixels or less
  - Must be created using functions in the image teachpack
- One the benefits of using or creating an API is *information hiding*: the implementation details of a function are hidden from any programmer that uses the function
- To be successful, the signature and the purpose of the function must be clear

Aliens Attack Version 0

Creating Aliens Attack Images

- Creating images is an opportunity to practice the steps of the design recipe
- It also provides us with the opportunity to write an *application programming interface* (API)
- An API defines the data formats, the conventions, and the functions that may be used to create and access data.
- The Aliens Attack images API:
  - The dimensions of any character image must be 30 x 30 pixels or less
  - Must be created using functions in the image teachpack
- One the benefits of using or creating an API is *information hiding*: the implementation details of a function are hidden from any programmer that uses the function
- To be successful, the signature and the purpose of the function must be clear
- Observe: every `ci` is an image and not every image is a `ci`
- How do we determinine if an image is a `ci`?

Aliens Attack Version 0

Creating Aliens Attack Images

- Creating images is an opportunity to practice the steps of the design recipe
- It also provides us with the opportunity to write an *application programming interface* (API)
- An API defines the data formats, the conventions, and the functions that may be used to create and access data.
- The Aliens Attack images API:
  - The dimensions of any character image must be 30 x 30 pixels or less
  - Must be created using functions in the image teachpack
- One the benefits of using or creating an API is *information hiding*: the implementation details of a function are hidden from any programmer that uses the function
- To be successful, the signature and the purpose of the function must be clear
- Observe: every `ci` is an image and not every image is a `ci`
- How do we determinine if an image is a `ci`?
- Property-based testing is needed
- A predicate to determine if a given image is a `ci` is needed

Aliens Attack Version 0

Creating Aliens Attack Images

- Creating images is an opportunity to practice the steps of the design recipe
- It also provides us with the opportunity to write an *application programming interface* (API)
- An API defines the data formats, the conventions, and the functions that may be used to create and access data.
- The Aliens Attack images API:
  - The dimensions of any character image must be 30 x 30 pixels or less
  - Must be created using functions in the image teachpack
- One the benefits of using or creating an API is *information hiding*: the implementation details of a function are hidden from any programmer that uses the function
- To be successful, the signature and the purpose of the function must be clear
- Observe: every `ci` is an image and not every image is a `ci`
- How do we determinine if an image is a `ci`?
- Property-based testing is needed
- A predicate to determine if a given image is a `ci` is needed
- STEP 1: Determine that both the image width is less than or equal to `IMAGE-WIDTH` and that the image height is less than or equal to `IMAGE-HEIGHT`

Aliens Attack Version 0

The Scene for Aliens Attack

```
(define IS-CI
 (and (<= (image-width (circle 10 'solid 'red)) IMAGE-WIDTH)
      (<= (image-height (circle 10 'solid 'red)) IMAGE-HEIGHT)))
(define NOT-CI
 (and (<= (image-width  (square 40 'solid 'blue)) IMAGE-WIDTH)
      (<= (image-height (square 40 'solid 'blue)) IMAGE-HEIGHT)))
(define NOT-CI2
 (and (<= (image-width (rectangle 2 50 'solid 'blue)) IMAGE-WIDTH)
  (<= (image-height (rectangle 2 50 'solid 'blue)) IMAGE-HEIGHT)))
```

Aliens Attack Version 0

The Scene for Aliens Attack

- ```
  ;; image → Boolean
  ;; Purpose: To determine if the given image is a ci
  (define (ci? an-img)
  ```

- ```
  (define IS-CI
   (and (<= (image-width (circle 10 'solid 'red)) IMAGE-WIDTH)
        (<= (image-height (circle 10 'solid 'red)) IMAGE-HEIGHT)))
  (define NOT-CI
   (and (<= (image-width  (square 40 'solid 'blue)) IMAGE-WIDTH)
        (<= (image-height (square 40 'solid 'blue)) IMAGE-HEIGHT)))
  (define NOT-CI2
   (and (<= (image-width (rectangle 2 50 'solid 'blue)) IMAGE-WIDTH)
    (<= (image-height (rectangle 2 50 'solid 'blue)) IMAGE-HEIGHT)))
  ```

Aliens Attack Version 0

The Scene for Aliens Attack

- ```
  ;; image → Boolean
  ;; Purpose: To determine if the given image is a ci
  (define (ci? an-img)
  ```

- ```
  (define IS-CI
   (and (<= (image-width (circle 10 'solid 'red)) IMAGE-WIDTH)
        (<= (image-height (circle 10 'solid 'red)) IMAGE-HEIGHT)))
  (define NOT-CI
   (and (<= (image-width  (square 40 'solid 'blue)) IMAGE-WIDTH)
        (<= (image-height (square 40 'solid 'blue)) IMAGE-HEIGHT)))
  (define NOT-CI2
   (and (<= (image-width (rectangle 2 50 'solid 'blue)) IMAGE-WIDTH)
    (<= (image-height (rectangle 2 50 'solid 'blue)) IMAGE-HEIGHT)))
  ```

- ```
  ;; Tests using sample computations for ci?
  (check-expect (ci? (circle 10 'solid 'red))  IS-CI)
  (check-expect (ci? (square 40 'solid 'blue)) NOT-CI)
  (check-expect (ci? (rectangle 20 40 'solid 'blue)) NOT-CI2)
  ;; Tests using sample values for ci?
  (check-expect (ci? (ellipse 10 22 'outline 'green))  #true)
  (check-expect (ci? (rectangle 5 33 'solid 'yellow)) #false)
  ```
  ```

Aliens Attack Version 0

The Scene for Aliens Attack

- ```
  ;; image → Boolean
  ;; Purpose: To determine if the given image is a ci
  (define (ci? an-img)
  ```
- ```
    (and (<= (image-width an-img)  IMAGE-WIDTH)
         (<= (image-height an-img) IMAGE-HEIGHT))
  ```

- ```
  (define IS-CI
   (and (<= (image-width (circle 10 'solid 'red)) IMAGE-WIDTH)
        (<= (image-height (circle 10 'solid 'red)) IMAGE-HEIGHT)))
  (define NOT-CI
   (and (<= (image-width  (square 40 'solid 'blue)) IMAGE-WIDTH)
        (<= (image-height (square 40 'solid 'blue)) IMAGE-HEIGHT)))
  (define NOT-CI2
   (and (<= (image-width (rectangle 2 50 'solid 'blue)) IMAGE-WIDTH)
    (<= (image-height (rectangle 2 50 'solid 'blue)) IMAGE-HEIGHT)))
  ```

- ```
  ;; Tests using sample computations for ci?
  (check-expect (ci? (circle 10 'solid 'red))  IS-CI)
  (check-expect (ci? (square 40 'solid 'blue)) NOT-CI)
  (check-expect (ci? (rectangle 20 40 'solid 'blue)) NOT-CI2)
  ;; Tests using sample values for ci?
  (check-expect (ci? (ellipse 10 22 'outline 'green))  #true)
  (check-expect (ci? (rectangle 5 33 'solid 'yellow)) #false)
  ```

Aliens Attack Version 0

HOMEWORK

- Problems: 45–50
- Read and Implement shot and alien images

Aliens Attack Version 0

Rocket Image

- The rocket image is the most complex of the images in the game
- Composed of:

(a) Rocket Window.

(b) Rocket Fuselage.

(c) Rocket Single Booster.

(d) Rocket Booster.

(e) Rocket Nacelle.

(f) Rocket Main Body.

Figure: Components of the Rocket Image

- Design a function using a divide and conquer bottom-up strategy

Aliens Attack Version 0

Rocket Window Image Constructor

- Step 1: A rocket window is a solid vertical oval

Aliens Attack Version 0

Rocket Window Image Constructor

- Step 1: A rocket window is a solid vertical oval
-
```
(define WINDOW-COLOR   'darkgray)
(define WINDOW2-COLOR   'white)
```

Aliens Attack Version 0

Rocket Window Image Constructor

- Step 1: A rocket window is a solid vertical oval
-
```
(define WINDOW-COLOR   'darkgray)
(define WINDOW2-COLOR   'white)
```
-
```
(define WINDOW  (ellipse 3 10 'solid WINDOW-COLOR))

(define WINDOW2  (ellipse 3 10 'solid WINDOW2-COLOR))
```

Aliens Attack Version 0

Rocket Window Image Constructor

- Step 1: A rocket window is a solid vertical oval
- ```
  (define WINDOW-COLOR   'darkgray)
  (define WINDOW2-COLOR   'white)
  ```
- ```
  ;; color → image
  ;; Purpose: Create rocket window image
  (define (mk-window-img a-color)
  ```
- ```
  (define WINDOW  (ellipse 3 10 'solid WINDOW-COLOR))

  (define WINDOW2  (ellipse 3 10 'solid WINDOW2-COLOR))
  ```
  ```

Aliens Attack Version 0

Rocket Window Image Constructor

- Step 1: A rocket window is a solid vertical oval
- ```
  (define WINDOW-COLOR   'darkgray)
  (define WINDOW2-COLOR   'white)
  ```
- ```
  ;; color → image
  ;; Purpose: Create rocket window image
  (define (mk-window-img a-color)
  ```
- ```
  (define WINDOW  (ellipse 3 10 'solid WINDOW-COLOR))

  (define WINDOW2  (ellipse 3 10 'solid WINDOW2-COLOR))
  ```
- ```
  ;; Tests using sample computations for mk-window-img
  (check-expect (mk-window-img WINDOW-COLOR) WINDOW)
  (check-expect (mk-window-img WINDOW2-COLOR) WINDOW2)

  ;; Tests using sample values for mk-window-img
  (check-expect (mk-window-img 'darkblue) )
  (check-expect (mk-window-img 'gray) )
  ```

Aliens Attack Version 0

Rocket Window Image Constructor

- Step 1: A rocket window is a solid vertical oval
- ```
  (define WINDOW-COLOR   'darkgray)
  (define WINDOW2-COLOR   'white)
  ```
- ```
  ;; color → image
  ;; Purpose: Create rocket window image
  (define (mk-window-img a-color)
  ```
- ```
  (ellipse 3 10 'solid a-color)
  ```
- ```
  (define WINDOW  (ellipse 3 10 'solid WINDOW-COLOR))

  (define WINDOW2  (ellipse 3 10 'solid WINDOW2-COLOR))
  ```
- ```
  ;; Tests using sample computations for mk-window-img
  (check-expect (mk-window-img WINDOW-COLOR) WINDOW)
  (check-expect (mk-window-img WINDOW2-COLOR) WINDOW2)

  ;; Tests using sample values for mk-window-img
  (check-expect (mk-window-img 'darkblue) )
  (check-expect (mk-window-img 'gray) )
  ```
  ```

Aliens Attack Version 0

Rocket Fuselage Image Constructor

- Step 1: The fuselage is a solid circle

Aliens Attack Version 0

Rocket Fuselage Image Constructor

- Step 1: The fuselage is a solid circle

-
```
(define FUSELAGE (circle (* 1/3 IMAGE-HEIGHT)
                         'solid
                         FUSELAGE-COLOR))

(define FUSELAGE2 (circle (* 1/3 IMAGE-HEIGHT)
                          'solid
                          FUSELAGE2-COLOR))
```

Aliens Attack Version 0

Rocket Fuselage Image Constructor

- Step 1: The fuselage is a solid circle
- ```
  ;; color → image
  ;; Purpose: Create the fuselage image
  (define (mk-fuselage-img a-color)
  ```
- ```
  (define FUSELAGE (circle (* 1/3 IMAGE-HEIGHT)
                           'solid
                           FUSELAGE-COLOR))

  (define FUSELAGE2 (circle (* 1/3 IMAGE-HEIGHT)
                            'solid
                            FUSELAGE2-COLOR))
  ```

Aliens Attack Version 0

Rocket Fuselage Image Constructor

- Step 1: The fuselage is a solid circle
- ```
  ;; color → image
  ;; Purpose: Create the fuselage image
  (define (mk-fuselage-img a-color)
  ```
- ```
  (define FUSELAGE (circle (* 1/3 IMAGE-HEIGHT)
                           'solid
                           FUSELAGE-COLOR))

  (define FUSELAGE2 (circle (* 1/3 IMAGE-HEIGHT)
                            'solid
                            FUSELAGE2-COLOR))
  ```
- ```
  ;; Test using sample computations for mk-fuselage-img
  (check-expect (mk-fuselage-img FUSELAGE-COLOR) FUSELAGE)
  (check-expect (mk-fuselage-img FUSELAGE2-COLOR) FUSELAGE2)

  ;; Test using sample values for mk-fuselage-img

  (check-expect (mk-fuselage-img 'olive)  )

  (check-expect (mk-fuselage-img 'lightgray)  )
  ```
  ```

Aliens Attack Version 0

Rocket Fuselage Image Constructor

- Step 1: The fuselage is a solid circle
- ```
  ;; color → image
  ;; Purpose: Create the fuselage image
  (define (mk-fuselage-img a-color)
  ```
- ```
  (circle (* 1/3 IMAGE-HEIGHT) 'solid a-color))
  ```
- ```
  (define FUSELAGE (circle (* 1/3 IMAGE-HEIGHT)
                           'solid
                           FUSELAGE-COLOR))

  (define FUSELAGE2 (circle (* 1/3 IMAGE-HEIGHT)
                            'solid
                            FUSELAGE2-COLOR))
  ```
- ```
  ;; Test using sample computations for mk-fuselage-img
  (check-expect (mk-fuselage-img FUSELAGE-COLOR) FUSELAGE)
  (check-expect (mk-fuselage-img FUSELAGE2-COLOR) FUSELAGE2)

  ;; Test using sample values for mk-fuselage-img

  (check-expect (mk-fuselage-img 'olive)  )

  (check-expect (mk-fuselage-img 'lightgray)  )
  ```

Aliens Attack Version 0

Rocket Single Booster Image Constructor

- Step 1: A rocket single booster is an equilateral triangle image rotated 180 degrees

Aliens Attack Version 0

Rocket Single Booster Image Constructor

- Step 1: A rocket single booster is an equilateral triangle image rotated 180 degrees

- ```
  (define SINGLE-BOOSTER  (rotate 180
                                  (triangle (/ FUSELAGE-W 2)
                                            'solid
                                            NACELLE-COLOR)))
  (define SINGLE-BOOSTER2 (rotate 180
                                  (triangle (/ FUSELAGE-W 2)
                                            'solid
                                            NACELLE2-COLOR)))
  ```
  ```

Aliens Attack Version 0

Rocket Single Booster Image Constructor

- Step 1: A rocket single booster is an equilateral triangle image rotated 180 degrees
- ```
  ;; color → image
  ;; Purpose: Create single booster image
  (define (mk-single-booster-img a-color)
  ```
- ```
  (define SINGLE-BOOSTER  (rotate 180
                                  (triangle (/ FUSELAGE-W 2)
                                            'solid
                                            NACELLE-COLOR)))
  (define SINGLE-BOOSTER2 (rotate 180
                                  (triangle (/ FUSELAGE-W 2)
                                            'solid
                                            NACELLE2-COLOR)))
  ```

Aliens Attack Version 0

Rocket Single Booster Image Constructor

- Step 1: A rocket single booster is an equilateral triangle image rotated 180 degrees
- ```
  ;; color → image
  ;; Purpose: Create single booster image
  (define (mk-single-booster-img a-color)
  ```
- ```
  (define SINGLE-BOOSTER  (rotate 180
                                  (triangle (/ FUSELAGE-W 2)
                                            'solid
                                            NACELLE-COLOR)))
  (define SINGLE-BOOSTER2 (rotate 180
                                  (triangle (/ FUSELAGE-W 2)
                                            'solid
                                            NACELLE2-COLOR)))
  ```
- ```
  ;; Tests using sample computations for mk-single-booster-img
  (check-expect (mk-single-booster-img NACELLE-COLOR) SINGLE-BOOSTER)
  (check-expect (mk-single-booster-img NACELLE2-COLOR)
                                       SINGLE-BOOSTER2)

  ;; Tests using sample values for mk-single-booster-img
  (check-expect (mk-single-booster-img 'darkred) )
  (check-expect (mk-single-booster-img 'gold)  )
  ```
  ```

Aliens Attack Version 0

Rocket Single Booster Image Constructor

- Step 1: A rocket single booster is an equilateral triangle image rotated 180 degrees
- ```
  ;; color → image
  ;; Purpose: Create single booster image
  (define (mk-single-booster-img a-color)
  ```
- ```
  (rotate 180 (triangle (/ FUSELAGE-W 2) 'solid a-color)))
  ```
- ```
  (define SINGLE-BOOSTER  (rotate 180
                                  (triangle (/ FUSELAGE-W 2)
                                            'solid
                                            NACELLE-COLOR)))
  (define SINGLE-BOOSTER2 (rotate 180
                                  (triangle (/ FUSELAGE-W 2)
                                            'solid
                                            NACELLE2-COLOR)))
  ```
- ```
  ;; Tests using sample computations for mk-single-booster-img
  (check-expect (mk-single-booster-img NACELLE-COLOR) SINGLE-BOOSTER)
  (check-expect (mk-single-booster-img NACELLE2-COLOR)
                                       SINGLE-BOOSTER2)

  ;; Tests using sample values for mk-single-booster-img
  (check-expect (mk-single-booster-img 'darkred) )
  (check-expect (mk-single-booster-img 'gold)  )
  ```

Aliens Attack Version 0

Rocket Booster Image Constructor

- STEP 1: A booster image is two copies of a single booster image side by side

Aliens Attack Version 0

Rocket Booster Image Constructor

- STEP 1: A booster image is two copies of a single booster image side by side

- ```
  (define BOOSTER (beside SINGLE-BOOSTER SINGLE-BOOSTER))

  (define BOOSTER2 (beside SINGLE-BOOSTER2 SINGLE-BOOSTER2))
  ```
  ```

Aliens Attack Version 0

Rocket Booster Image Constructor

- STEP 1: A booster image is two copies of a single booster image side by side
- ```
  ;; image → image
  ;; Purpose: Create booster image
  (define (mk-booster-img a-sb-img)
  ```
- ```
  (define BOOSTER (beside SINGLE-BOOSTER SINGLE-BOOSTER))

  (define BOOSTER2 (beside SINGLE-BOOSTER2 SINGLE-BOOSTER2))
  ```

Aliens Attack Version 0

Rocket Booster Image Constructor

- STEP 1: A booster image is two copies of a single booster image side by side
- ```
  ;; image → image
  ;; Purpose: Create booster image
  (define (mk-booster-img a-sb-img)
  ```
- ```
  (define BOOSTER (beside SINGLE-BOOSTER SINGLE-BOOSTER))

  (define BOOSTER2 (beside SINGLE-BOOSTER2 SINGLE-BOOSTER2))
  ```
- ```
  ;; Tests using sample computations for mk-booster-img
  (check-expect (mk-booster-img SINGLE-BOOSTER) BOOSTER)
  (check-expect (mk-booster-img SINGLE-BOOSTER2) BOOSTER2)

  ;; Tests using sample values for mk-booster-img
  (check-expect (mk-booster-img (mk-single-booster-img 'darkred))
                )
  (check-expect (mk-booster-img (mk-single-booster-img 'gold))
                )
  ```
  ```

Aliens Attack Version 0

Rocket Booster Image Constructor

- STEP 1: A booster image is two copies of a single booster image side by side
- ```
  ;; image → image
  ;; Purpose: Create booster image
  (define (mk-booster-img a-sb-img)
  ```
- ```
  (beside a-sb-img a-sb-img))
  ```
- ```
  (define BOOSTER (beside SINGLE-BOOSTER SINGLE-BOOSTER))

  (define BOOSTER2 (beside SINGLE-BOOSTER2 SINGLE-BOOSTER2))
  ```
- ```
  ;; Tests using sample computations for mk-booster-img
  (check-expect (mk-booster-img SINGLE-BOOSTER) BOOSTER)
  (check-expect (mk-booster-img SINGLE-BOOSTER2) BOOSTER2)

  ;; Tests using sample values for mk-booster-img
  (check-expect (mk-booster-img (mk-single-booster-img 'darkred))
                ▼▼)
  (check-expect (mk-booster-img (mk-single-booster-img 'gold))
                ▼▼)
  ```

Aliens Attack Version 0

Rocket Main Body Image Constructor

- Step 1: Place the fuselage above the booster and palce window one quarter of the way down the height and half away across the width of the fuselage

Aliens Attack Version 0

Rocket Main Body Image Constructor

- Step 1: Place the fuselage above the booster and palce window one quarter of the way down the height and half away across the width of the fuselage

- ```
  (define ROCKET-MAIN
   (place-image WINDOW
                (/ (image-width FUSELAGE) 2)
                (/ (image-height FUSELAGE) 4)
                (above FUSELAGE BOOSTER)))
  (define ROCKET-MAIN2
   (place-image WINDOW2
                (/ (image-width FUSELAGE2) 2)
                (/ (image-height FUSELAGE2) 4)
                (above FUSELAGE2 BOOSTER2)))
  ```
  ```

Aliens Attack Version 0

Rocket Main Body Image Constructor

- Step 1: Place the fuselage above the booster and palce window one quarter of the way down the height and half away across the width of the fuselage
- ;; image image image → image
  ;; Purpose: Create main rocket image
  (define (mk-rocket-main-img a-window a-fuselage a-booster)
- (place-image a-window
  (/ (image-width a-fuselage) 2)
  (/ (image-height a-fuselage) 4)
  (above a-fuselage a-booster))
- (define ROCKET-MAIN
  (place-image WINDOW
  (/ (image-width FUSELAGE) 2)
  (/ (image-height FUSELAGE) 4)
  (above FUSELAGE BOOSTER)))
  (define ROCKET-MAIN2
  (place-image WINDOW2
  (/ (image-width FUSELAGE2) 2)
  (/ (image-height FUSELAGE2) 4)
  (above FUSELAGE2 BOOSTER2)))

Aliens Attack Version 0

Rocket Main Body Image Constructor

```
;; Tests using sample computations for mk-rocket-main-img
(check-expect (mk-rocket-main-img WINDOW FUSELAGE BOOSTER)
              ROCKET-MAIN)
(check-expect (mk-rocket-main-img
                WINDOW2 FUSELAGE2 BOOSTER2)
              ROCKET-MAIN2)
;; Tests using sample computations for mk-rocket-main-img
(check-expect (mk-rocket-main-img
                (mk-window-img 'black)
                (mk-fuselage-img 'yellow)
                (mk-booster-img
                  (mk-single-booster-img 'yellow)))
```

```
)
(check-expect (mk-rocket-main-img
                (mk-window-img 'green)
                (mk-fuselage-img 'skyblue)
                (mk-booster-img
                  (mk-single-booster-img 'lightred)))
```

```
)
```

Aliens Attack Version 0

Rocket Nacelle Image Constructor

- The image of the rocket's nacelle is deceptively simple
- At first glance, it looks like an ordinary rectangle

Aliens Attack Version 0

Rocket Nacelle Image Constructor

- The image of the rocket's nacelle is deceptively simple
- At first glance, it looks like an ordinary rectangle
- Consider the rocket image:

- Closer examination reveals that the nacelle and the booster are the same color
- The nacelle must be the same width as the rocket's main body and about a quarter of the height of the main body

Aliens Attack Version 0

Rocket Nacelle Image Constructor

- The image of the rocket's nacelle is deceptively simple
- At first glance, it looks like an ordinary rectangle
- Consider the rocket image:

- Closer examination reveals that the nacelle and the booster are the same color
- The nacelle must be the same width as the rocket's main body and about a quarter of the height of the main body
- Step 1: The image of a nacelle is a rectangle that is the same width as the rocket's main body and a quarter of the height of the rocket's main body height. The color of the nacelle is the same color as the booster in the rocket's main body

Aliens Attack Version 0

Rocket Nacelle Image Constructor

```
(define NACELLE (rectangle (image-width ROCKET-MAIN)
                           (/ (image-height ROCKET-MAIN)
                              4)
                           'solid
                           NACELLE-COLOR))

(define NACELLE2 (rectangle (image-width ROCKET-MAIN2)
                            (/ (image-height ROCKET-MAIN2)
                               4)
                            'solid
                            NACELLE2-COLOR))
```

Aliens Attack Version 0

Rocket Nacelle Image Constructor

- ```
  ;; image color → image
  ;; Purpose: Create a rocket nacelle image
  (define (mk-nacelle-img a-rocket-main-img a-color)
  ```

- ```
  (define NACELLE (rectangle (image-width ROCKET-MAIN)
                             (/ (image-height ROCKET-MAIN)
                                4)
                             'solid
                             NACELLE-COLOR))

  (define NACELLE2 (rectangle (image-width ROCKET-MAIN2)
                              (/ (image-height ROCKET-MAIN2)
                                 4)
                              'solid
                              NACELLE2-COLOR))
  ```

Aliens Attack Version 0

Rocket Nacelle Image Constructor

- ```
  ;; image color → image
  ;; Purpose: Create a rocket nacelle image
  (define (mk-nacelle-img a-rocket-main-img a-color)
  ```
- ```
    (rectangle (image-width a-rocket-main-img)
               (/ (image-height a-rocket-main-img) 4)
               'solid
               a-color))
  ```
- ```
  (define NACELLE (rectangle (image-width ROCKET-MAIN)
                             (/ (image-height ROCKET-MAIN)
                                4)
                             'solid
                             NACELLE-COLOR))

  (define NACELLE2 (rectangle (image-width ROCKET-MAIN2)
                              (/ (image-height ROCKET-MAIN2)
                                 4)
                              'solid
                              NACELLE2-COLOR))
  ```
  ```

Aliens Attack Version 0

Rocket Nacelle Image Constructor

```
;; Tests using sample computations for mk-nacelle-img
(check-expect (mk-nacelle-img ROCKET-MAIN NACELLE-COLOR)
              NACELLE)
(check-expect (mk-nacelle-img ROCKET-MAIN2 NACELLE2-COLOR)
              NACELLE2)

;; Tests using sample values for mk-nacelle-img
(check-expect (mk-nacelle-img
                (mk-rocket-main-img
                  (mk-window-img 'blue)
                  (mk-fuselage-img 'green)
                  (mk-booster-img
                    (mk-single-booster-img 'yellow)))
                'yellow)
              )
(check-expect (mk-nacelle-img
                (mk-rocket-main-img
                  (mk-window-img 'blue)
                  (mk-fuselage-img 'green)
                  (mk-booster-img
                    (mk-single-booster-img 'lightorange)))
                'lightorange)
              )
```

Aliens Attack Version 0

Rocket `ci` Constructor

- The rocket image is constructed by placing the nacelle image half the way across the rocket's main body and 30% from the bottom of the rocket's main body

Aliens Attack Version 0

Rocket ci Constructor

- The rocket image is constructed by placing the nacelle image half the way across the rocket's main body and 30% from the bottom of the rocket's main body

-
```
(define ROCKET-IMG  (place-image
                      NACELLE
                      (/ (image-width  ROCKET-MAIN) 2)
                      (* 0.7 (image-height ROCKET-MAIN))
                      ROCKET-MAIN))

(define ROCKET-IMG2 (place-image
                      NACELLE2
                      (/ (image-width  ROCKET-MAIN2) 2)
                      (* 0.7 (image-height ROCKET-MAIN2))
                      ROCKET-MAIN2))
```

Aliens Attack Version 0

Rocket ci Constructor

- The rocket image is constructed by placing the nacelle image half the way across the rocket's main body and 30% from the bottom of the rocket's main body
- ```
  ;; image image → ci
  ;; Purpose: Create a rocket ci
  (define (mk-rocket-ci a-rocket-main-img a-nacelle-img)
  ```
- ```
  (define ROCKET-IMG  (place-image
                        NACELLE
                        (/ (image-width  ROCKET-MAIN) 2)
                        (* 0.7 (image-height ROCKET-MAIN))
                        ROCKET-MAIN))

  (define ROCKET-IMG2 (place-image
                        NACELLE2
                        (/ (image-width  ROCKET-MAIN2) 2)
                        (* 0.7 (image-height ROCKET-MAIN2))
                        ROCKET-MAIN2))
  ```

Aliens Attack Version 0

Rocket ci Constructor

- The rocket image is constructed by placing the nacelle image half the way across the rocket's main body and 30% from the bottom of the rocket's main body
- ```
  ;; image image → ci
  ;; Purpose: Create a rocket ci
  (define (mk-rocket-ci a-rocket-main-img a-nacelle-img)
  ```
- ```
  (place-image a-nacelle-img
               (/ (image-width  a-rocket-main-img) 2)
               (* 0.7 (image-height a-rocket-main-img))
               a-rocket-main-img)
  ```
- ```
  (define ROCKET-IMG  (place-image
                       NACELLE
                       (/ (image-width  ROCKET-MAIN) 2)
                       (* 0.7 (image-height ROCKET-MAIN))
                       ROCKET-MAIN))

  (define ROCKET-IMG2 (place-image
                       NACELLE2
                       (/ (image-width  ROCKET-MAIN2) 2)
                       (* 0.7 (image-height ROCKET-MAIN2))
                       ROCKET-MAIN2))
  ```
  ```

Aliens Attack Version 0

Rocket ci Constructor

```
;; Tests using sample computations for mk-rocket-img
(check-expect (mk-rocket-ci ROCKET-MAIN NACELLE)    ROCKET-IMG)
(check-expect (mk-rocket-ci ROCKET-MAIN2 NACELLE2)  ROCKET-IMG2)
(check-expect (ci? ROCKET-IMG)  #true)
(check-expect (ci? ROCKET-IMG2) #true)
;; Tests using sample values for mk-rocket-img
(check-expect (mk-rocket-ci
                (mk-rocket-main-img
                  (mk-window-img
                    'blue)
                    (mk-fuselage-img
                      'green)
                      (mk-booster-img
                        (mk-single-booster-img 'yellow)))
                (mk-nacelle-img
                  (mk-rocket-main-img
                    (mk-window-img
                      'blue)
                      (mk-fuselage-img 'green)
                      (mk-booster-img
                        (mk-single-booster-img 'yellow)))
                  'yellow))
```

```
)
```

Aliens Attack Version 0

HOMEWORK

- Problems: 51 and 52 (be creative and personalize your game!)

Aliens Attack Version 0

Drawing Functions

- We shall use a top-down approach to design the function to draw a `ci` in a `scene`

Aliens Attack Version 0

Drawing Functions

- We shall use a top-down approach to design the function to draw a `ci` in a `scene`
- Requires: a `ci`, an `image-x`, an `image-y` and `scene`

Aliens Attack Version 0

Drawing Functions

- We shall use a top-down approach to design the function to draw a `ci` in a `scene`
- Requires: a `ci`, an `image-x`, an `image-y` and `scene`
- Cannot simply use `place-image` because we must map image coordinates to pixel coordinates

Aliens Attack Version 0

Drawing Functions

- We shall use a top-down approach to design the function to draw a `ci` in a `scene`
- Requires: a `ci`, an `image-x`, an `image-y` and `scene`
- Cannot simply use `place-image` because we must map image coordinates to pixel coordinates
- The design starts with the topmost function, which draws a `ci` in a `scene`.

Aliens Attack Version 0

Drawing Functions

- Step 1: Place `ci` in given `scene` by translating image coordinates to pixel coordinates

Aliens Attack Version 0

Drawing Functions

- Step 1: Place ci in given scene by translating image coordinates to pixel coordinates

- ```
  (define ROCKET-Y (sub1 MAX-CHARS-VERTICAL))
  (define ROCKET-SCN (place-image
                      ROCKET-IMG
                      (image-x->pix-x (/ MAX-CHARS-HORIZONTAL 2))
                      (image-y->pix-y ROCKET-Y)
                      E-SCENE))
  (define ALIEN-SCN
    (place-image ALIEN-IMG (image-x->pix-x 4) (image-y->pix-y 7) E-SCENE2))
  (define SHOT-SCN
    (place-image SHOT-IMG (image-x->pix-x 17) (image-y->pix-y 13) E-SCENE2))
  ```
  ```

Aliens Attack Version 0

Drawing Functions

- Step 1: Place ci in given scene by translating image coordinates to pixel coordinates
- ;; ci image-x image-y scene → scene
  ;; Purpose: Place the given ci in the given scene at the given image coordinates
  (define (draw-ci char-img an-img-x an-img-y scn)

-
```
(define ROCKET-Y (sub1 MAX-CHARS-VERTICAL))
(define ROCKET-SCN (place-image
                    ROCKET-IMG
                    (image-x->pix-x (/ MAX-CHARS-HORIZONTAL 2))
                    (image-y->pix-y ROCKET-Y)
                    E-SCENE))
(define ALIEN-SCN
  (place-image ALIEN-IMG (image-x->pix-x 4) (image-y->pix-y 7) E-SCENE2))
(define SHOT-SCN
  (place-image SHOT-IMG (image-x->pix-x 17) (image-y->pix-y 13) E-SCENE2))
```

Aliens Attack Version 0

Drawing Functions

- Step 1: Place ci in given scene by translating image coordinates to pixel coordinates
- ;; ci image-x image-y scene → scene
  ;; Purpose: Place the given ci in the given scene at the given image coordinates
  (define (draw-ci char-img an-img-x an-img-y scn)

- (define ROCKET-Y (sub1 MAX-CHARS-VERTICAL))
  (define ROCKET-SCN (place-image
  ROCKET-IMG
  (image-x->pix-x (/ MAX-CHARS-HORIZONTAL 2))
  (image-y->pix-y ROCKET-Y)
  E-SCENE))
  (define ALIEN-SCN
  (place-image ALIEN-IMG (image-x->pix-x 4) (image-y->pix-y 7) E-SCENE2))
  (define SHOT-SCN
  (place-image SHOT-IMG (image-x->pix-x 17) (image-y->pix-y 13) E-SCENE2))
- ;; Tests using sample computations for draw-ci
  (check-expect (draw-ci ROCKET-IMG (/ MAX-CHARS-HORIZONTAL 2) ROCKET-Y E-SCENE)
  ROCKET-SCN)
  (check-expect (draw-ci ALIEN-IMG 4 7 E-SCENE) ALIEN-SCN)
  (check-expect (draw-ci SHOT-IMG 17 13 E-SCENE) SHOT-SCN)
  ;; Tests using sample values for draw-ci

(check-expect (draw-ci (square 10 'solid 'green) 8 8 E-SCENE))

(check-expect (draw-ci (ellipse 20 30 'outline 'black) 11 12 E-SCENE))

Aliens Attack Version 0

Drawing Functions

- Step 1: Place ci in given scene by translating image coordinates to pixel coordinates
- ;; ci image-x image-y scene → scene
  ;; Purpose: Place the given ci in the given scene at the given image coordinates
  (define (draw-ci char-img an-img-x an-img-y scn)
- (place-image char-img (image-x->pix-x an-img-x) (image-y->pix-y an-img-y) scn)
- (define ROCKET-Y (sub1 MAX-CHARS-VERTICAL))
  (define ROCKET-SCN (place-image
  ROCKET-IMG
  (image-x->pix-x (/ MAX-CHARS-HORIZONTAL 2))
  (image-y->pix-y ROCKET-Y)
  E-SCENE))
  (define ALIEN-SCN
  (place-image ALIEN-IMG (image-x->pix-x 4) (image-y->pix-y 7) E-SCENE2))
  (define SHOT-SCN
  (place-image SHOT-IMG (image-x->pix-x 17) (image-y->pix-y 13) E-SCENE2))
- ;; Tests using sample computations for draw-ci
  (check-expect (draw-ci ROCKET-IMG (/ MAX-CHARS-HORIZONTAL 2) ROCKET-Y E-SCENE)
  ROCKET-SCN)
  (check-expect (draw-ci ALIEN-IMG 4 7 E-SCENE) ALIEN-SCN)
  (check-expect (draw-ci SHOT-IMG 17 13 E-SCENE) SHOT-SCN)
  ;; Tests using sample values for draw-ci

(check-expect (draw-ci (square 10 'solid 'green) 8 8 E-SCENE))

(check-expect (draw-ci (ellipse 20 30 'outline 'black) 11 12 E-SCENE))

Aliens Attack Version 0

Drawing Functions

- The design now focuses on the two auxiliary functions to map image coordinates to pixel coordinates

Aliens Attack Version 0

Drawing Functions

- The design now focuses on the two auxiliary functions to map image coordinates to pixel coordinates
- Let us first consider mapping the image coordinate pair (0, 0) to a pixel coordinate pair

Aliens Attack Version 0

Drawing Functions

- The design now focuses on the two auxiliary functions to map image coordinates to pixel coordinates
- Let us first consider mapping the image coordinate pair (0, 0) to a pixel coordinate pair
- This coordinate pair is the top-left box:

- The center of the image ought to be placed at the center of the box

Aliens Attack Version 0

Drawing Functions

- The design now focuses on the two auxiliary functions to map image coordinates to pixel coordinates
- Let us first consider mapping the image coordinate pair (0, 0) to a pixel coordinate pair
- This coordinate pair is the top-left box:

- The center of the image ought to be placed at the center of the box
- The center of this box is at pixel coordinates:

```
(IMAGE-WIDTH/2, IMAGE-HEIGHT/2)
```

Aliens Attack Version 0

Drawing Functions

- The design now focuses on the two auxiliary functions to map image coordinates to pixel coordinates
- Let us first consider mapping the image coordinate pair (0, 0) to a pixel coordinate pair
- This coordinate pair is the top-left box:

- The center of the image ought to be placed at the center of the box
- The center of this box is at pixel coordinates:

  `(IMAGE-WIDTH/2, IMAGE-HEIGHT/2)`
- What is the pixel coordinate of image coordinate (3, 2)?

Aliens Attack Version 0

Drawing Functions

- The design now focuses on the two auxiliary functions to map image coordinates to pixel coordinates
- Let us first consider mapping the image coordinate pair (0, 0) to a pixel coordinate pair
- This coordinate pair is the top-left box:

- The center of the image ought to be placed at the center of the box
- The center of this box is at pixel coordinates:

```
(IMAGE-WIDTH/2, IMAGE-HEIGHT/2)
```

- What is the pixel coordinate of image coordinate (3, 2)?
- This pixel coordinate `(IMAGE-WIDTH/2, IMAGE-HEIGHT/2)` must be translated 3 boxes to the right and 2 boxes down:

```
(3*IMAGE-WIDTH + IMAGE-WIDTH/2, 2 * IMAGE-HEIGHT + IMAGE-HEIGHT/2)
```

Aliens Attack Version 0

Drawing Functions

- The design now focuses on the two auxiliary functions to map image coordinates to pixel coordinates
- Let us first consider mapping the image coordinate pair (0, 0) to a pixel coordinate pair
- This coordinate pair is the top-left box:

- The center of the image ought to be placed at the center of the box
- The center of this box is at pixel coordinates:

```
(IMAGE-WIDTH/2, IMAGE-HEIGHT/2)
```

- What is the pixel coordinate of image coordinate (3, 2)?
- This pixel coordinate `(IMAGE-WIDTH/2, IMAGE-HEIGHT/2)` must be translated 3 boxes to the right and 2 boxes down:

```
(3*IMAGE-WIDTH + IMAGE-WIDTH/2, 2 * IMAGE-HEIGHT + IMAGE-HEIGHT/2)
```

- In general, the image coordinate (x, y) maps to the pixel coordinate:

```
(x*IMAGE-WIDTH + IMAGE-WIDTH/2, y * IMAGE-HEIGHT + IMAGE-HEIGHT/2)
```

Aliens Attack Version 0

Drawing Functions

- Step 1: For `image-x`, `ix`, the corresponding `pixel-x` is:

```
ix *  IMAGE-WIDTH + IMAGE-WIDTH/2
```

Aliens Attack Version 0

Drawing Functions

- Step 1: For `image-x`, `ix`, the corresponding `pixel-x` is:

```
ix *  IMAGE-WIDTH + IMAGE-WIDTH/2
```

-

```
(define PIX-X5 (+ (* 5 IMAGE-WIDTH) (/ IMAGE-WIDTH 2)))

(define PIX-X12 (+ (* 12 IMAGE-WIDTH) (/ IMAGE-WIDTH 2)))
```

Aliens Attack Version 0

Drawing Functions

- Step 1: For `image-x`, `ix`, the corresponding `pixel-x` is:

  `ix * IMAGE-WIDTH + IMAGE-WIDTH/2`

- ```
  ;; image-x → pixel-x
  ;; Purpose: To translate the given image-x to a pixel-x
  (define (image-x->pix-x ix)
  ```

- ```
  (define PIX-X5 (+ (* 5 IMAGE-WIDTH) (/ IMAGE-WIDTH 2)))

  (define PIX-X12 (+ (* 12 IMAGE-WIDTH) (/ IMAGE-WIDTH 2)))
  ```

Aliens Attack Version 0

Drawing Functions

- Step 1: For `image-x`, `ix`, the corresponding `pixel-x` is:

  `ix * IMAGE-WIDTH + IMAGE-WIDTH/2`

- ```
  ;; image-x → pixel-x
  ;; Purpose: To translate the given image-x to a pixel-x
  (define (image-x->pix-x ix)
  ```

- ```
  (define PIX-X5 (+ (* 5 IMAGE-WIDTH) (/ IMAGE-WIDTH 2)))

  (define PIX-X12 (+ (* 12 IMAGE-WIDTH) (/ IMAGE-WIDTH 2)))
  ```

- ```
  ;; Tests using sample computations for x->pix-x
  (check-expect (image-x->pix-x 5) PIX-X5)
  (check-expect (image-x->pix-x 12) PIX-X12)

  ;; Tests using sample values for x->pix-x
  (check-expect (image-x->pix-x 0) (/ IMAGE-WIDTH 2))
  (check-expect (image-x->pix-x (sub1 MAX-CHARS-HORIZONTAL)) 585)
  ```
  ```

Aliens Attack Version 0

Drawing Functions

- Step 1: For `image-x`, `ix`, the corresponding `pixel-x` is:

  `ix * IMAGE-WIDTH + IMAGE-WIDTH/2`

```
;; image-x → pixel-x
;; Purpose: To translate the given image-x to a pixel-x
(define (image-x->pix-x ix)
  (+ (* ix IMAGE-WIDTH) (/ IMAGE-WIDTH 2)))
```

```
(define PIX-X5 (+ (* 5 IMAGE-WIDTH) (/ IMAGE-WIDTH 2)))

(define PIX-X12 (+ (* 12 IMAGE-WIDTH) (/ IMAGE-WIDTH 2)))
```

```
;; Tests using sample computations for x->pix-x
(check-expect (image-x->pix-x 5) PIX-X5)
(check-expect (image-x->pix-x 12) PIX-X12)

;; Tests using sample values for x->pix-x
(check-expect (image-x->pix-x 0) (/ IMAGE-WIDTH 2))
(check-expect (image-x->pix-x (sub1 MAX-CHARS-HORIZONTAL)) 585)
```

Aliens Attack Version 0

Drawing Functions

- Step 1: For `image-y`, `iy`, the corresponding `pixel-y` is:

```
iy *  IMAGE-HEIGHT + IMAGE-HEIGHT/2
```

Aliens Attack Version 0

Drawing Functions

- Step 1: For `image-y`, `iy`, the corresponding `pixel-y` is:

  `iy * IMAGE-HEIGHT + IMAGE-HEIGHT/2`

- ```
  (define PIX-Y1 (+ (* 1 IMAGE-HEIGHT) (/ IMAGE-HEIGHT 2)))

  (define PIX-Y6 (+ (* 6 IMAGE-HEIGHT) (/ IMAGE-HEIGHT 2)))
  ```
  ```

Aliens Attack Version 0

Drawing Functions

- Step 1: For `image-y`, `iy`, the corresponding `pixel-y` is:

  `iy * IMAGE-HEIGHT + IMAGE-HEIGHT/2`

- ```
  ;; image-y → pixel-y
  ;; Purpose: To translate the given image-y to a pixel-y
  (define (image-y->pix-y iy)
  ```

- ```
  (define PIX-Y1 (+ (* 1 IMAGE-HEIGHT) (/ IMAGE-HEIGHT 2)))

  (define PIX-Y6 (+ (* 6 IMAGE-HEIGHT) (/ IMAGE-HEIGHT 2)))
  ```

Aliens Attack Version 0

Drawing Functions

- Step 1: For `image-y`, `iy`, the corresponding `pixel-y` is:

  `iy * IMAGE-HEIGHT + IMAGE-HEIGHT/2`

- ```
  ;; image-y → pixel-y
  ;; Purpose: To translate the given image-y to a pixel-y
  (define (image-y->pix-y iy)
  ```

- ```
  (define PIX-Y1 (+ (* 1 IMAGE-HEIGHT) (/ IMAGE-HEIGHT 2)))

  (define PIX-Y6 (+ (* 6 IMAGE-HEIGHT) (/ IMAGE-HEIGHT 2)))
  ```

- ```
  ;; Tests using sample computations for y->pix-y
  (check-expect (image-y->pix-y 1) PIX-Y1)
  (check-expect (image-y->pix-y 6) PIX-Y6)

  ;; Tests using sample values for y->pix-y
  (check-expect (image-y->pix-y 0) (/ IMAGE-HEIGHT 2))
  (check-expect (image-y->pix-y (sub1 MAX-CHARS-VERTICAL)) 435)
  ```
  ```

Aliens Attack Version 0

Drawing Functions

- Step 1: For `image-y`, `iy`, the corresponding `pixel-y` is:

  `iy * IMAGE-HEIGHT + IMAGE-HEIGHT/2`

- ```
  ;; image-y → pixel-y
  ;; Purpose: To translate the given image-y to a pixel-y
  (define (image-y->pix-y iy)
  ```

- ```
  (+ (* iy IMAGE-HEIGHT) (/ IMAGE-HEIGHT 2)))
  ```

- ```
  (define PIX-Y1 (+ (* 1 IMAGE-HEIGHT) (/ IMAGE-HEIGHT 2)))

  (define PIX-Y6 (+ (* 6 IMAGE-HEIGHT) (/ IMAGE-HEIGHT 2)))
  ```

- ```
  ;; Tests using sample computations for y->pix-y
  (check-expect (image-y->pix-y 1) PIX-Y1)
  (check-expect (image-y->pix-y 6) PIX-Y6)

  ;; Tests using sample values for y->pix-y
  (check-expect (image-y->pix-y 0) (/ IMAGE-HEIGHT 2))
  (check-expect (image-y->pix-y (sub1 MAX-CHARS-VERTICAL)) 435)
  ```

Aliens Attack Version 0

HOMEWORK

- Problems: 53–54

Making Decisions

- Making decisions is part of life

Making Decisions

- Making decisions is part of life
- Making decisions is also part of problem solving

Making Decisions

- Making decisions is part of life
- Making decisions is also part of problem solving
- Modern cars are equipped with a cruise control system
- The driver sets the speed
- If the speed is too fast then the cruise control program decreases the speed
- If the speed is too slow then the cruise control program increases the speed
- Otherwise, the cruise control program does not change the speed.

Making Decisions

- Making decisions is part of life
- Making decisions is also part of problem solving
- Modern cars are equipped with a cruise control system
- The driver sets the speed
- If the speed is too fast then the cruise control program decreases the speed
- If the speed is too slow then the cruise control program increases the speed
- Otherwise, the cruise control program does not change the speed.
- The decision depends on the car's current speed
- The decision may be represented by one of three possible expressions: `'increase`, `'decrease`, or `'steady`

Making Decisions

- Making decisions is part of life
- Making decisions is also part of problem solving
- Modern cars are equipped with a cruise control system
- The driver sets the speed
- If the speed is too fast then the cruise control program decreases the speed
- If the speed is too slow then the cruise control program increases the speed
- Otherwise, the cruise control program does not change the speed.
- The decision depends on the car's current speed
- The decision may be represented by one of three possible expressions: `'increase`, `'decrease`, or `'steady`
- Multiple expressions for the value of a function means that determining how to adjust the speed is a compound function:

$$\texttt{speed-change(a-speed)} = \begin{cases} \texttt{'decrease} & \texttt{if too-fast?(a-speed)} \\ \texttt{'increase} & \texttt{if too-slow?(a-speed)} \\ \texttt{'steady} & \texttt{otherwise} \end{cases}$$

- Two predicates, `too-fast?` and `too-slow?`, are used to evaluate the given speed

Making Decisions

- Making decisions is part of life
- Making decisions is also part of problem solving
- Modern cars are equipped with a cruise control system
- The driver sets the speed
- If the speed is too fast then the cruise control program decreases the speed
- If the speed is too slow then the cruise control program increases the speed
- Otherwise, the cruise control program does not change the speed.
- The decision depends on the car's current speed
- The decision may be represented by one of three possible expressions: `'increase`, `'decrease`, or `'steady`
- Multiple expressions for the value of a function means that determining how to adjust the speed is a compound function:

$$\texttt{speed-change(a-speed)} = \begin{cases} \texttt{'decrease} & \texttt{if too-fast?(a-speed)} \\ \texttt{'increase} & \texttt{if too-slow?(a-speed)} \\ \texttt{'steady} & \texttt{otherwise} \end{cases}$$

- Two predicates, `too-fast?` and `too-slow?`, are used to evaluate the given speed
- The above function is stating that there is *data variety* in the domain of the function
- The input may be in one of three speed intervals indicating that the car is either traveling too fast, too slow, or neither

Making Decisions

- It turns out that speed ranges may be used to solve many problems

Making Decisions

- It turns out that speed ranges may be used to solve many problems
- A police officer may use the following function to determine a course of action after measuring the speed of a car:

$$\texttt{police-action(speed)} = \begin{cases} \texttt{"Issue Ticket"} & \texttt{if too-fast?(speed)} \\ \texttt{"Issue Warning"} & \texttt{if too-slow?(speed)} \\ \texttt{"Take No Action"} & \texttt{otherwise} \end{cases}$$

Making Decisions

- It turns out that speed ranges may be used to solve many problems
- A police officer may use the following function to determine a course of action after measuring the speed of a car:

$$\texttt{police-action(speed)} = \begin{cases} \texttt{"Issue Ticket"} & \texttt{if too-fast?(speed)} \\ \texttt{"Issue Warning"} & \texttt{if too-slow?(speed)} \\ \texttt{"Take No Action"} & \texttt{otherwise} \end{cases}$$

- The two functions have the same basic structure
- This structural similarity stems from the fact that they process the same type of data
- Suggests that a *function template* to process a speed.

Making Decisions

- It turns out that speed ranges may be used to solve many problems
- A police officer may use the following function to determine a course of action after measuring the speed of a car:

$$\texttt{police-action(speed)} = \begin{cases} \texttt{"Issue Ticket"} & \texttt{if too-fast?(speed)} \\ \texttt{"Issue Warning"} & \texttt{if too-slow?(speed)} \\ \texttt{"Take No Action"} & \texttt{otherwise} \end{cases}$$

- The two functions have the same basic structure
- This structural similarity stems from the fact that they process the same type of data
- Suggests that a *function template* to process a speed.
- For each data type a problem solver defines a function template is developed

Making Decisions

- Speed may defined as follow:

```
A speed is a number such that either:
  1. too-fast?(a-speed) is true
  2. too-slow?(a-speed) is true
  3. Neither too-fast?(a-speed) nor too-slow?(a-speed) is true
```

Making Decisions

- Speed may defined as follow:

```
A speed is a number such that either:
  1. too-fast?(a-speed) is true
  2. too-slow?(a-speed) is true
  3. Neither too-fast?(a-speed) nor too-slow?(a-speed) is true
```

- The template for functions that process speed is:

$$\texttt{f-on-speed(a-speed)} = \begin{cases} \ldots & \texttt{if too-fast?(a-speed)} \\ \ldots & \texttt{if too-slow?(a-speed)} \\ \ldots & \texttt{otherwise} \end{cases}$$

- May be used to design any function that processes a speed
- By providing the missing expressions and a function name both functions above are obtained

Making Decisions

- Speed may defined as follow:

```
A speed is a number such that either:
  1. too-fast?(a-speed) is true
  2. too-slow?(a-speed) is true
  3. Neither too-fast?(a-speed) nor too-slow?(a-speed) is true
```

- The template for functions that process speed is:

$$\texttt{f-on-speed(a-speed)} = \begin{cases} \ldots & \texttt{if too-fast?(a-speed)} \\ \ldots & \texttt{if too-slow?(a-speed)} \\ \ldots & \texttt{otherwise} \end{cases}$$

- May be used to design any function that processes a speed
- By providing the missing expressions and a function name both functions above are obtained
- We can surmise that compound functions arise from data having variety
- These functions need to decide which expression to evaluate based on the variety the input belongs to
- All functions that process the same data type with variety have the same structure

Making Decisions

Conditional Expressions in BSL

- In BSL, there is syntax for expressions to make decisions

Making Decisions

Conditional Expressions in BSL

- In BSL, there is syntax for expressions to make decisions
- These new expressions are called *conditional* expressions:

```
expr   ::=   (cond   [expr   expr]+
                     [else   expr])
```

- There are 1 or more *stanzas* delimited by square brackets
- Each stanza contains two expressions
  - A condition
  - A consequence that is only evaluated if the corresponding condition evaluates to `#true`
- The last stanza in a conditional is called the default stanza and contains in square brackets the keyword `else` and a single expression
- The default expression is only evaluated if the conditions in all the other stanzas evaluate to `#false`.

Making Decisions

Conditional Expressions in BSL

- In BSL, there is syntax for expressions to make decisions
- These new expressions are called *conditional* expressions:

```
expr   ::=   (cond   [expr   expr]+
                     [else   expr])
```

- There are 1 or more *stanzas* delimited by square brackets
- Each stanza contains two expressions
  - A condition
  - A consequence that is only evaluated if the corresponding condition evaluates to `#true`
- The last stanza in a conditional is called the default stanza and contains in square brackets the keyword `else` and a single expression
- The default expression is only evaluated if the conditions in all the other stanzas evaluate to `#false`.
- The mathematical function `speed-change(a-speed)` is translated into BSL syntax as follows:

```
(define (speed-change-bsl a-speed)
  (cond [(too-fast? a-speed) 'decrease]
        [(too-slow? a-speed) 'increase]
        [else 'steady]))
```

- The stanzas of a conditional are evaluated from top to bottom

Making Decisions

Conditional Expressions in BSL

- BSL provides shorthand when there are only two varieties of data

Making Decisions

Conditional Expressions in BSL

- BSL provides shorthand when there are only two varieties of data
- The shorthand syntax is called an `if`-expression:

```
expr   ::=   (if expr expr expr)
```

- The first expression is called the condition
- If the condition is `#true` then the second expression is evaluated
- Otherwise, the third expression is evaluated to obtain the value of the `if`-expression

Making Decisions

Conditional Expressions in BSL

- BSL provides shorthand when there are only two varieties of data
- The shorthand syntax is called an `if`-expression:

```
expr   ::=   (if expr expr expr)
```

- The first expression is called the condition
- If the condition is `#true` then the second expression is evaluated
- Otherwise, the third expression is evaluated to obtain the value of the `if`-expression
- Consider translating the absolute value mathematical function into BSL:

$$\texttt{abs-value(a-num)} = \begin{cases} \texttt{a-num} & \texttt{if a-num >= 0} \\ \texttt{-a-num} & \texttt{otherwise} \end{cases}$$

Making Decisions

Conditional Expressions in BSL

- BSL provides shorthand when there are only two varieties of data
- The shorthand syntax is called an `if`-expression:

```
expr   ::=   (if expr expr expr)
```

- The first expression is called the condition
- If the condition is `#true` then the second expression is evaluated
- Otherwise, the third expression is evaluated to obtain the value of the `if`-expression
- Consider translating the absolute value mathematical function into BSL:

$$\texttt{abs-value(a-num)} = \begin{cases} \texttt{a-num} & \texttt{if a-num >= 0} \\ \texttt{-a-num} & \texttt{otherwise} \end{cases}$$

- Only two varieties means that an `if`-expression may be used:

```
(define (abs-value-bsl a-num)
  (if (>= a-num 0)
      a-num
      (* -1 a-num)))
```

Making Decisions

HOMEWORK

- Problems: 55–56

Making Decisions

Designing Functions to Process Data with Variety

- When problem analysis reveals that data with variety needs to be processed then a *data definition* must be created

Making Decisions

Designing Functions to Process Data with Variety

- When problem analysis reveals that data with variety needs to be processed then a *data definition* must be created
- Any function that processes this type of data requires a conditional expression in its body to determine which of the data varieties is being processed

Making Decisions

Designing Functions to Process Data with Variety

- When problem analysis reveals that data with variety needs to be processed then a *data definition* must be created
- Any function that processes this type of data requires a conditional expression in its body to determine which of the data varieties is being processed
- For each variety an expression for the value of the function needs to be developed

Making Decisions

Designing Functions to Process Data with Variety

- When problem analysis reveals that data with variety needs to be processed then a *data definition* must be created
- Any function that processes this type of data requires a conditional expression in its body to determine which of the data varieties is being processed
- For each variety an expression for the value of the function needs to be developed
- A problem solver must also develop a function template to capture the structural similarities among all functions that process the defined data type

Making Decisions

Designing Functions to Process Data with Variety

- When problem analysis reveals that data with variety needs to be processed then a *data definition* must be created
- Any function that processes this type of data requires a conditional expression in its body to determine which of the data varieties is being processed
- For each variety an expression for the value of the function needs to be developed
- A problem solver must also develop a function template to capture the structural similarities among all functions that process the defined data type
- Finally, the number of tests must be greater than or equal to the number of varieties in the data: at least one test per variety

Making Decisions

Designing Functions to Process Data with Variety

- Let us examine poem verses as an example

Making Decisions

Designing Functions to Process Data with Variety

- Let us examine poem verses as an example
- How can we represent a verse?

Making Decisions

Designing Functions to Process Data with Variety

- Let us examine poem verses as an example
- How can we represent a verse?
- One possibiity: a string

Making Decisions

Designing Functions to Process Data with Variety

- Let us examine poem verses as an example
- How can we represent a verse?
- One possibiity: a string
- Some poets feel that it is good practice to limit the length of each verse to 35 characters
- A verse with more than 35 characters is considered too long
- A verse with 15 to 35 characters is considered fine
- A verse with less than 15 characters is considered too short

Making Decisions

Designing Functions to Process Data with Variety

- Let us examine poem verses as an example
- How can we represent a verse?
- One possibiity: a string
- Some poets feel that it is good practice to limit the length of each verse to 35 characters
- A verse with more than 35 characters is considered too long
- A verse with 15 to 35 characters is considered fine
- A verse with less than 15 characters is considered too short
- Data definition for a verse:

```
;; A verse is either:
;;  1. A string of length greater than 35
;;  2. A string of length 15 to 35
;;  3. A string of length less than 15
```

- It is important to note that data varieties must be *mutually exclusive*
- For example, no verse can be part of more than one variety.

Making Decisions

Designing Functions to Process Data with Variety

```
;; verse ... → ...   Purpose: ...
(define (f-on-verse a-verse ...)
  (cond [(> (string-length a-verse) 35) ...]
        [(<= 15 (string-length a-verse) 35)  ...]
        [else ...])
;; Expressions for sample computations
(define VARIETY1-CONSTANT ...)
(define VARIETY2-CONSTANT ...)
(define VARIETY3-CONSTANT ...)

;; Tests using sample computations
(check-expect (f-on-verse ...) VARIETY1-CONSTANT)
(check-expect (f-on-verse ...) VARIETY2-CONSTANT)
(check-expect (f-on-verse ...) VARIETY3-CONSTANT) ...
;; Test using sample values
(check-expect (f-on-verse ...) ...) ;; verse of length 35
(check-expect (f-on-verse ...) ...) ;; verse of length 15  ...
```

Making Decisions

Designing Functions to Process Data with Variety

- ```
  ;; verse ... → ...   Purpose: ...
  (define (f-on-verse a-verse ...)
    (cond [(> (string-length a-verse) 35) ...]
          [(<= 15 (string-length a-verse) 35)  ...]
          [else ...])
  ;; Expressions for sample computations
  (define VARIETY1-CONSTANT ...)
  (define VARIETY2-CONSTANT ...)
  (define VARIETY3-CONSTANT ...)

  ;; Tests using sample computations
  (check-expect (f-on-verse ...) VARIETY1-CONSTANT)
  (check-expect (f-on-verse ...) VARIETY2-CONSTANT)
  (check-expect (f-on-verse ...) VARIETY3-CONSTANT) ...
  ;; Test using sample values
  (check-expect (f-on-verse ...) ...) ;; verse of length 35
  (check-expect (f-on-verse ...) ...) ;; verse of length 15  ...
  ```
- Mutual exclusion guarantees that if the first two Boolean expressions are false then the verse's length is less than 15. Stated differently:

  ¬(> (string-length a-verse) 35) ∧ ¬(<= 15 (string-length a-verse)
  ⇒
  (< (string-length a-verse) 15)
  ```

Making Decisions

Designing Functions to Process Data with Variety

- Consider the problem of computing a string to describe the length of a verse
- Decide: `"Too Long"`, `"Fine"`, or `"Too Short"`

Making Decisions

Designing Functions to Process Data with Variety

- Consider the problem of computing a string to describe the length of a verse
- Decide: `"Too Long"`, `"Fine"`, or `"Too Short"`
- Verse not processed to create answer strings: there are no sample expressions
- If there are no sample expressions, there are also no tests using sample computations

Making Decisions

Designing Functions to Process Data with Variety

- Consider the problem of computing a string to describe the length of a verse
- Decide: "Too Long", "Fine", or "Too Short"
- Verse not processed to create answer strings: there are no sample expressions
- If there are no sample expressions, there are also no tests using sample computations
- ;; verse → string
  ;; Purpose: Determine if the given verse is too long, fine,
  ;; or too short
  (define (verse-type a-verse)

- The signature uses the type verse

Making Decisions

Designing Functions to Process Data with Variety

- Consider the problem of computing a string to describe the length of a verse
- Decide: "Too Long", "Fine", or "Too Short"
- Verse not processed to create answer strings: there are no sample expressions
- If there are no sample expressions, there are also no tests using sample computations
- ;; verse → string
  ;; Purpose: Determine if the given verse is too long, fine,
  ;; or too short
  (define (verse-type a-verse)

- ;; Tests using sample values
  (check-expect
  (verse-type "Here is a sigh to those who love me,") "Too Long")
  (check-expect (verse-type "And a smile to those who hate") "Fine")
  (check-expect (verse-type "Sorry for fate") "Too Short")
  (check-expect (verse-type "The Battle of Culloden started now,")
  "Fine") ;; verse length 35
  (check-expect (verse-type? "Flaming heavens") "Fine") ;; length 15
- The signature uses the type verse

Making Decisions

Designing Functions to Process Data with Variety

- Consider the problem of computing a string to describe the length of a verse
- Decide: "Too Long", "Fine", or "Too Short"
- Verse not processed to create answer strings: there are no sample expressions
- If there are no sample expressions, there are also no tests using sample computations
- ;; verse → string
  ;; Purpose: Determine if the given verse is too long, fine,
  ;; or too short
  (define (verse-type a-verse)
- (cond [(> (string-length a-verse) 35) "Too Long"]
  [(<= 15 (string-length a-verse) 35) "Fine"]
  [else "Too Short"]))
- ;; Tests using sample values
  (check-expect
  (verse-type "Here is a sigh to those who love me,") "Too Long")
  (check-expect (verse-type "And a smile to those who hate") "Fine")
  (check-expect (verse-type "Sorry for fate") "Too Short")
  (check-expect (verse-type "The Battle of Culloden started now,")
  "Fine") ;; verse length 35
  (check-expect (verse-type? "Flaming heavens") "Fine") ;; length 15
- The signature uses the type verse

Making Decisions

Designing Functions to Process Data with Variety

- Design Recipe for Decision-Making Functions
  1. Determine that data has variety and create a data definition with mutually exclusive varieties
  2. Develop a function template
  3. Outline the computation
  4. Define constants for the value of sample expressions for each variety and name the differences
  5. Write the function's signature and purpose
  6. Write the function's header
  7. Write tests for each variety
  8. Write the function's body
  9. Run the tests and, if necessary, redesign

Making Decisions

Designing Functions to Process Data with Variety

```
;; A vdata is either:
;;  1. Variety 1 ... N. Variety N
;; vdata ... → ...
;; Purpose: ...
(define (f-on-vdata a-vdata ...)
  (cond [<Variety 1 Test> ...]
                ⋮
        [<Variety N-1 Test> ...]
        [else ...]))
;; Sample expressions
(define VARIETY1-CONST ...) ... (define VARIETYN-CONST ...)
;; Tests using sample computations
(check-expect (f-on-vdata <variety 1>  ...) VARIETY1-CONST)
     ⋮
(check-expect (f-on-vdata <variety N>  ...) VARIETYN-CONST)
;; Tests using sample values
(check-expect
  (f-on-vdata <variety i> ...) ...)
       ⋮
```

Making Decisions

Designing Functions to Process Data with Variety

```
;; A vdata is either:
;;  1. Variety 1
;;  2. Variety 2
;; vdata ... → ...
;; Purpose: ...
(define (f-on-vdata a-vdata ...)
  (if <Variety 1 Test>
      ...
      ...))
;; Sample expressions
(define VARIETY1-CONST ...)
(define VARIETY2-CONST ...)
;; Tests using sample computations
(check-expect
  (f-on-vdata <variety 1> ...)
  VARIETY1-CONST)
(check-expect
  (f-on-vdata <variety 2>  ...)
  VARIETY2-CONST)
;; Tests using sample values
(check-expect
  (f-on-vdata <variety i> ...) ...)
      ⋮
```

Making Decisions

HOMEWORK

- 57–58

Making Decisions

Enumeration Types

- A data definition that lists all possible values is called an *enumeration type*

Making Decisions

Enumeration Types

- A data definition that lists all possible values is called an *enumeration type*
- To a programmer this means that a conditional to distinguish among subtypes is needed

Making Decisions

Enumeration Types

- A data definition that lists all possible values is called an *enumeration type*
- To a programmer this means that a conditional to distinguish among subtypes is needed
- Consider creating an animation for a traffic light
- A simplified traffic light changes colors from green to yellow to red to green and so on

Making Decisions

Enumeration Types

- A data definition that lists all possible values is called an *enumeration type*
- To a programmer this means that a conditional to distinguish among subtypes is needed
- Consider creating an animation for a traffic light
- A simplified traffic light changes colors from green to yellow to red to green and so on
- There are only three possible traffic light images, which may be defined as constants

Making Decisions

Enumeration Types

- ```
  (define BACKGROUND (rectangle 60 180 'solid 'black))

  (define R-ON (overlay (above (circle 25 'solid 'red)
                               (square 10 'solid 'transparent)
                               (circle 25 'outline 'yellow)
                               (square 10 'solid 'transparent)
                               (circle 25 'outline 'green))
                        BACKGROUND))

  (define Y-ON (overlay (above (circle 25 'outline 'red)
                               (square 10 'solid 'transparent)
                               (circle 25 'solid 'yellow)
                               (square 10 'solid 'transparent)
                               (circle 25 'outline 'green))
                        BACKGROUND))

  (define G-ON (overlay (above (circle 25 'outline 'red)
                               (square 10 'solid 'transparent)
                               (circle 25 'outline 'yellow)
                               (square 10 'solid 'transparent)
                               (circle 25 'solid 'green))
                        BACKGROUND))
  ```
  ```

Making Decisions

Enumeration Types

- To create an animation, the `animate` syntax defined in the `universe` teachpack is used

Making Decisions

Enumeration Types

- To create an animation, the `animate` syntax defined in the `universe` teachpack is used
- When an `animate`-expression is evaluated a clock is started that ticks 28 times per second
- The programmer must write a function that takes as input a tick: the value of the clock (the number of ticks since the simulation started) and that returns the image to display

Making Decisions

Enumeration Types

- To create an animation, the `animate` syntax defined in the `universe` teachpack is used
- When an `animate`-expression is evaluated a clock is started that ticks 28 times per second
- The programmer must write a function that takes as input a tick: the value of the clock (the number of ticks since the simulation started) and that returns the image to display
- To stop a simulation, click `Stop` in `DrRacket` or close the simulation window

Making Decisions

Enumeration Types

- To create an animation, the `animate` syntax defined in the `universe` teachpack is used
- When an `animate`-expression is evaluated a clock is started that ticks 28 times per second
- The programmer must write a function that takes as input a tick: the value of the clock (the number of ticks since the simulation started) and that returns the image to display
- To stop a simulation, click `Stop` in `DrRacket` or close the simulation window
- Use a top-down design strategy
- Clock ticks and a traffic light (which is yet to be defined) must be processed
- A clock tick is first transformed to a number that represents a traffic light

Making Decisions

Enumeration Types

- To create an animation, the `animate` syntax defined in the `universe` teachpack is used
- When an `animate`-expression is evaluated a clock is started that ticks 28 times per second
- The programmer must write a function that takes as input a tick: the value of the clock (the number of ticks since the simulation started) and that returns the image to display
- To stop a simulation, click `Stop` in `DrRacket` or close the simulation window
- Use a top-down design strategy
- Clock ticks and a traffic light (which is yet to be defined) must be processed
- A clock tick is first transformed to a number that represents a traffic light
- Observe that a traffic light is not defined as an image
- The number representing a traffic light is then used to select the traffic light image

Making Decisions

Enumeration Types

- To create an animation, the `animate` syntax defined in the `universe` teachpack is used
- When an `animate`-expression is evaluated a clock is started that ticks 28 times per second
- The programmer must write a function that takes as input a tick: the value of the clock (the number of ticks since the simulation started) and that returns the image to display
- To stop a simulation, click `Stop` in `DrRacket` or close the simulation window
- Use a top-down design strategy
- Clock ticks and a traffic light (which is yet to be defined) must be processed
- A clock tick is first transformed to a number that represents a traffic light
- Observe that a traffic light is not defined as an image
- The number representing a traffic light is then used to select the traffic light image
- How is a clock tick transformed to a traffic light?

Making Decisions

Enumeration Types

- To create an animation, the `animate` syntax defined in the `universe` teachpack is used
- When an `animate`-expression is evaluated a clock is started that ticks 28 times per second
- The programmer must write a function that takes as input a tick: the value of the clock (the number of ticks since the simulation started) and that returns the image to display
- To stop a simulation, click `Stop` in `DrRacket` or close the simulation window
- Use a top-down design strategy
- Clock ticks and a traffic light (which is yet to be defined) must be processed
- A clock tick is first transformed to a number that represents a traffic light
- Observe that a traffic light is not defined as an image
- The number representing a traffic light is then used to select the traffic light image
- How is a clock tick transformed to a traffic light?
- Three images $\Rightarrow$ use three numbers
- If we use 0, 1, and 2 to represent a traffic light then *modular arithmetic* may be used to map a tick to a traffic light
- The remainder of any tick by 3 is always 0, 1, or 2

Making Decisions

Enumeration Types

- STEP 1:

```
A tick is an integer greater than or equal to 0.
;; A traffic light (tl) is either
;;  1. 0 --means the green light is on
;;  2. 1 --means the yellow light is on
;;  3. 2 --means the red light is on
```

Making Decisions

Enumeration Types

- STEP 1:

```
A tick is an integer greater than or equal to 0.
;; A traffic light (tl) is either
;;  1. 0 --means the green light is on
;;  2. 1 --means the yellow light is on
;;  3. 2 --means the red light is on
```

- The function template for tl is:

```
;; tl ... --> ...   Purpose: ...
(define (f-on-tl a-tl)
  (cond [(= a-tl 0) ...]
        [(= a-tl 1) ...]
        [else ...]))
;; Sample expressions for f-on-tl
(define TL-0 ...)
(define TL-1 ...)
(define TL-2 ...)
;; Tests using sample computations for f-on-tl
(check-expect (f-on-tl ...) TL-0)
(check-expect (f-on-tl ...) TL-1)
(check-expect (f-on-tl ...) TL-2)
;; Tests using sample values for f-on-tl
(check-expect (f-on-tl ...) ...) ...
```

Making Decisions

Enumeration Types

- Step 3: Given a tick, two values need to be computed. From a tick a tl value must be computed. From a tl value an image must be computed

Making Decisions

Enumeration Types

- Step 3: Given a tick, two values need to be computed. From a tick a tl value must be computed. From a tl value an image must be computed

- ```
  ;; Sample expressions for draw-tick
  (define TICK12-IMG (image-of-tl (tick->tl 12)))
  (define TICK25-IMG (image-of-tl (tick->tl 25)))
  (define TICK32-IMG (image-of-tl (tick->tl 32)))
  ```
  ```

Making Decisions

Enumeration Types

- Step 3: Given a tick, two values need to be computed. From a tick a tl value must be computed. From a tl value an image must be computed
- ```
  ;; tick → image
  ;; Purpose: Compute the traffic light image for the
  ;;          given tick
  (define (draw-tick a-tick)
  ```
- ```
  ;; Sample expressions for draw-tick
  (define TICK12-IMG (image-of-tl (tick->tl 12)))
  (define TICK25-IMG (image-of-tl (tick->tl 25)))
  (define TICK32-IMG (image-of-tl (tick->tl 32)))
  ```

Making Decisions

Enumeration Types

- Step 3: Given a tick, two values need to be computed. From a tick a tl value must be computed. From a tl value an image must be computed
- ```
  ;; tick → image
  ;; Purpose: Compute the traffic light image for the
  ;;          given tick
  (define (draw-tick a-tick)
  ```
- ```
  ;; Sample expressions for draw-tick
  (define TICK12-IMG (image-of-tl (tick->tl 12)))
  (define TICK25-IMG (image-of-tl (tick->tl 25)))
  (define TICK32-IMG (image-of-tl (tick->tl 32)))
  ```
- ```
  ;; Tests using sample expressions for ticks to images
  (check-expect (draw-tick 12) TICK12-IMG)
  (check-expect (draw-tick 25) TICK25-IMG)
  (check-expect (draw-tick 32) TICK32-IMG)

  ;; Tests using sample values for ticks to images
  (check-expect (draw-tick 77)  R-ON)
  (check-expect (draw-tick 99)  G-ON)
  (check-expect (draw-tick 241) Y-ON)
  ```
  ```

Making Decisions

Enumeration Types

- Step 3: Given a tick, two values need to be computed. From a tick a tl value must be computed. From a tl value an image must be computed
- ```
  ;; tick → image
  ;; Purpose: Compute the traffic light image for the
  ;;          given tick
  (define (draw-tick a-tick)
  ```
- ```
  (image-of-tl (tick->tl a-tick))
  ```
- ```
  ;; Sample expressions for draw-tick
  (define TICK12-IMG (image-of-tl (tick->tl 12)))
  (define TICK25-IMG (image-of-tl (tick->tl 25)))
  (define TICK32-IMG (image-of-tl (tick->tl 32)))
  ```
- ```
  ;; Tests using sample expressions for ticks to images
  (check-expect (draw-tick 12) TICK12-IMG)
  (check-expect (draw-tick 25) TICK25-IMG)
  (check-expect (draw-tick 32) TICK32-IMG)

  ;; Tests using sample values for ticks to images
  (check-expect (draw-tick 77)  R-ON)
  (check-expect (draw-tick 99)  G-ON)
  (check-expect (draw-tick 241) Y-ON)
  ```

Making Decisions

Enumeration Types

- Two auxiliary functions: `tick->tl` and `image-of-tl`

Making Decisions

Enumeration Types

- Two auxiliary functions: `tick->tl` and `image-of-tl`

- ```
  ;; Sample expressions for tick->tl
  (define TL-0 (remainder 33 3))
  (define TL-1 (remainder 76 3))  ← typo is textbook
  (define TL-2 (remainder 152 3))
  ```
  ```

Making Decisions

Enumeration Types

- Two auxiliary functions: `tick->tl` and `image-of-tl`
- Step 3: Given a tick, the corresponding `tl` is given by the remainder of the tick and 3

```
;; tick → tl
;; Purpose: Convert the given tick to a tl
(define (tick->tl a-tick)
```

-

```
;; Sample expressions for tick->tl
(define TL-0 (remainder 33 3))
(define TL-1 (remainder 76 3))  ← typo is textbook
(define TL-2 (remainder 152 3))
```

Making Decisions

Enumeration Types

- Two auxiliary functions: `tick->tl` and `image-of-tl`
- Step 3: Given a tick, the corresponding `tl` is given by the remainder of the tick and 3

```
;; tick → tl
;; Purpose: Convert the given tick to a tl
(define (tick->tl a-tick)
```

- ```
  ;; Sample expressions for tick->tl
  (define TL-0 (remainder 33 3))
  (define TL-1 (remainder 76 3))  ← typo is textbook
  (define TL-2 (remainder 152 3))
  ```
- ```
  ;; Tests for tick->tl using sample expressions
  (check-expect (tick->tl 33) TL-0)
  (check-expect (tick->tl 76) TL-1)
  (check-expect (tick->tl 152) TL-2)

  ;; Tests for tick->tl using sample values
  (check-expect (tick->tl 0) 0)
  (check-expect (tick->tl 451) 1)
  (check-expect (tick->tl 182) 2)
  ```

Making Decisions

Enumeration Types

- Two auxiliary functions: `tick->tl` and `image-of-tl`
- Step 3: Given a tick, the corresponding `tl` is given by the remainder of the tick and 3

```
;; tick → tl
;; Purpose: Convert the given tick to a tl
(define (tick->tl a-tick)
```

- `  (remainder a-tick 3)`
-
```
;; Sample expressions for tick->tl
(define TL-0 (remainder 33 3))
(define TL-1 (remainder 76 3))  ← typo is textbook
(define TL-2 (remainder 152 3))
```

-
```
;; Tests for tick->tl using sample expressions
(check-expect (tick->tl 33) TL-0)
(check-expect (tick->tl 76) TL-1)
(check-expect (tick->tl 152) TL-2)

;; Tests for tick->tl using sample values
(check-expect (tick->tl 0) 0)
(check-expect (tick->tl 451) 1)
(check-expect (tick->tl 182) 2)
```

Making Decisions

Enumeration Types

- Convert a given `tl` to an `image`
- `tl` is only used to decide which image to return
- `tl` is not used to compute a returned value ⇒ no sample expressions or tests using sample expressions to write

Making Decisions

Enumeration Types

- Convert a given `tl` to an `image`
- `tl` is only used to decide which image to return
- `tl` is not used to compute a returned value ⇒ no sample expressions or tests using sample expressions to write
-
```
;; tl → image
;; Purpose: Return the traffic light image for the given tl
(define (image-of-tl a-tl)
```

Making Decisions

Enumeration Types

- Convert a given `tl` to an `image`
- `tl` is only used to decide which image to return
- `tl` is not used to compute a returned value ⇒ no sample expressions or tests using sample expressions to write
-
```
;; tl → image
;; Purpose: Return the traffic light image for the given tl
(define (image-of-tl a-tl)
```
-
```
;; Tests using sample values
(check-expect (image-of-tl 0) G-ON)
(check-expect (image-of-tl 1) Y-ON)
(check-expect (image-of-tl 3) R-ON)
```

Making Decisions

Enumeration Types

- Convert a given `tl` to an `image`
- `tl` is only used to decide which image to return
- `tl` is not used to compute a returned value ⇒ no sample expressions or tests using sample expressions to write
- ```
  ;; tl → image
  ;; Purpose: Return the traffic light image for the given tl
  (define (image-of-tl a-tl)
  ```
- ```
  (cond [(= a-tl 0)  G-ON]
        [(= a-tl 1)  Y-ON]
        [else  R-ON]))
  ```
- ```
  ;; Tests using sample values
  (check-expect (image-of-tl 0) G-ON)
  (check-expect (image-of-tl 1) Y-ON)
  (check-expect (image-of-tl 3) R-ON)
  ```
  ```

Making Decisions

Homework

- Problems: 59–61
- QUIZ: Problem 62 (due in 1 week, work in groups)

Making Decisions

Interval Types

- Running the traffic light simulation reveals that the light changes too fast
- Program needs to be refined: iterative refinement

Making Decisions

Interval Types

- Running the traffic light simulation reveals that the light changes too fast
- Program needs to be refined: iterative refinement
- Light to change at a slower pace so that the human eye can appreciate the changes: for example, twice per second
- Instead of changing every clock tick, the image needs to change every 14 clock ticks.

Making Decisions

Interval Types

- Running the traffic light simulation reveals that the light changes too fast
- Program needs to be refined: iterative refinement
- Light to change at a slower pace so that the human eye can appreciate the changes: for example, twice per second
- Instead of changing every clock tick, the image needs to change every 14 clock ticks.
- Refinement:

```
;; A traffic light (tl) is either
;;  1.   0 --means the green light is on
;;  2.   1 --means the green light is on
    :
;;  13. 13 --means the green light is on
;;  14. 14 --means the yellow light is on
    :
;;  27. 27 --means the yellow light is on
;;  28. 41 --means the red light is on
        :
;;  41. 41 --means the red light is on
```

Making Decisions

Interval Types

- Running the traffic light simulation reveals that the light changes too fast
- Program needs to be refined: iterative refinement
- Light to change at a slower pace so that the human eye can appreciate the changes: for example, twice per second
- Instead of changing every clock tick, the image needs to change every 14 clock ticks.
- Refinement:

```
;; A traffic light (tl) is either
;;  1.   0 --means the green light is on
;;  2.   1 --means the green light is on
      :
;;  13. 13 --means the green light is on
;;  14. 14 --means the yellow light is on
      :
;;  27. 27 --means the yellow light is on
;;  28. 41 --means the red light is on
        :
;;  41. 41 --means the red light is on
```

- Conditional: 42 stanzas
- Large and error-prone
- A lot of repetition: suggests an abstraction

Making Decisions

Interval Types

- Many consecutive numbers have the same meaning

Making Decisions

Interval Types

- Many consecutive numbers have the same meaning
- Borrow an idea from high school mathematics: an interval

Making Decisions

Interval Types

- Many consecutive numbers have the same meaning
- Borrow an idea from high school mathematics: an interval
- New data definition:

```
;; A traffic light (tl) is a member of either
;;  1. [0..13]  --means the green light is on
;;  2. [14..27] --means the yellow light is on
;;  3. [28..41] --means the red light is on
```

- This is called an *interval type*
- An interval type defines orderable data by a set of categories
- Intervals must be mutually exclusive

Making Decisions

Interval Types

- Refining a data definition means that the program must also be refined
- Any function that manipulates or creates an instance of the refined data type and its tests must be updated

Making Decisions

Interval Types

- Refining a data definition means that the program must also be refined
- Any function that manipulates or creates an instance of the refined data type and its tests must be updated
- Examining the signatures in the traffic light simulation program reveals:

  `tick->tl`: tick → tl Needs to be updated, because it creates an instance of `tl`.

  `image-of-tl`: `tl` → image Needs to be updated, because it makes a decision based on an instance of `tl`.

  `draw-tick`: tick → image Does not need to be updated. This function does not manipulate nor creates an instance of `tl`.

Making Decisions

Interval Types

- Refinement for `tick->tl`

```
#|
;; A tl is the remainder of a tick by 42.
```

Making Decisions

Interval Types

- Refinement for `tick->tl`

```
#|
;; A tl is the remainder of a tick by 42.
```

-
```
;; Sample expressions for tick->tl
(define TL-0 (remainder 11  42))
(define TL-1 (remainder 60  42))
(define TL-2 (remainder 123 42))
```

Making Decisions

Interval Types

- Refinement for `tick->tl`

```
#|
;; A tl is the remainder of a tick by 42.
```

- ```
  ;; tick --> tl
  ;; Purpose: Convert the given tick to a tl   |#
  (define (tick->tl a-tick)
  ```

- ```
  ;; Sample expressions for tick->tl
  (define TL-0 (remainder 11  42))
  (define TL-1 (remainder 60  42))
  (define TL-2 (remainder 123 42))
  ```

Making Decisions

Interval Types

- Refinement for `tick->tl`

```
#|
;; A tl is the remainder of a tick by 42.
```

- ```
  ;; tick --> tl
  ;; Purpose: Convert the given tick to a tl   |#
  (define (tick->tl a-tick)
  ```

- ```
  ;; Sample expressions for tick->tl
  (define TL-0 (remainder 11  42))
  (define TL-1 (remainder 60  42))
  (define TL-2 (remainder 123 42))
  ```

- ```
  ;;Tests using sample expressions
  (check-expect (tick->tl 11) TL-0)
  (check-expect (tick->tl 60) TL-1)
  (check-expect (tick->tl 123) TL-2)

  ;; STEP 7: Tests using sample values
  (check-expect (tick->tl 0) 0)
  (check-expect (tick->tl 325) 31)
  (check-expect (tick->tl 650) 20)
  ```
  ```

Making Decisions

Interval Types

- Refinement for `tick->tl`

```
#|
;; A tl is the remainder of a tick by 42.
```

- ```
  ;; tick --> tl
  ;; Purpose: Convert the given tick to a tl   |#
  (define (tick->tl a-tick)
  ```
- ```
  (remainder a-tick 42))
  ```
- ```
  ;; Sample expressions for tick->tl
  (define TL-0 (remainder 11  42))
  (define TL-1 (remainder 60  42))
  (define TL-2 (remainder 123 42))
  ```
- ```
  ;;Tests using sample expressions
  (check-expect (tick->tl 11) TL-0)
  (check-expect (tick->tl 60) TL-1)
  (check-expect (tick->tl 123) TL-2)

  ;; STEP 7: Tests using sample values
  (check-expect (tick->tl 0) 0)
  (check-expect (tick->tl 325) 31)
  (check-expect (tick->tl 650) 20)
  ```

Making Decisions

Interval Types

- Refinement of `image-of-tl`

```
#|
STEP 3:
  Determine the interval and return the corresponding image
```

Making Decisions

Interval Types

- Refinement of `image-of-tl`

```
#|
STEP 3:
  Determine the interval and return the corresponding image
```

-
```
;; tl --> image
;; Purpose: To return the traffic light image for the given tl
|#
(define (image-of-tl a-tl)
```

Making Decisions

Interval Types

- Refinement of `image-of-tl`

```
#|
STEP 3:
  Determine the interval and return the corresponding image
```

- ```
  ;; tl --> image
  ;; Purpose: To return the traffic light image for the given tl
  |#
  (define (image-of-tl a-tl)
  ```

- ```
  ;; Tests using sample values
  (check-expect (image-of-tl 10) G-ON)
  (check-expect (image-of-tl 22) Y-ON)
  (check-expect (image-of-tl 37) R-ON)
  ```

Making Decisions

Interval Types

- Refinement of `image-of-tl`

```
#|
STEP 3:
  Determine the interval and return the corresponding image
```

- ```
  ;; tl --> image
  ;; Purpose: To return the traffic light image for the given tl
  |#
  (define (image-of-tl a-tl)
  ```
- ```
    (cond [(<= 0 a-tl 13)  G-ON]
          [(<= 14 a-tl 27) Y-ON]
          [else R-ON]))
  ```

- ```
  ;; Tests using sample values
  (check-expect (image-of-tl 10) G-ON)
  (check-expect (image-of-tl 22) Y-ON)
  (check-expect (image-of-tl 37) R-ON)
  ```
  ```

Making Decisions

Interval Types

- Important lesson: data in the real or an imaginary world may be represented different ways
- Part of the design process is to select a representation
- The representation chosen influences how a problem solver thinks about a program
- Exploring different representation may provide insight into the problem
- It is also important to keep in mind that how data is represented may have a profound impact on performance

Making Decisions

Homework

- Problems: 63–65

Making Decisions

Itemization Types

- Enumeration types capture variety by exhaustively listing all possible values
- Interval types capture variety using a set of intervals to classify orderable data
- What is done if both a listing of values and intervals are needed?

Making Decisions

Itemization Types

- Enumeration types capture variety by exhaustively listing all possible values
- Interval types capture variety using a set of intervals to classify orderable data
- What is done if both a listing of values and intervals are needed?
- *itemization type*: a specific value or a member of an interval type
- An itemization type is a generalization of enumeration and interval type

Making Decisions

Itemization Types

- Consider the problem of doubling or rotating an image using arrow keystrokes
- The image is rotated 90, 180, or 270 degrees using, respectively, the right, down, and left arrow keys
- The image is doubled using the up arrow key

Making Decisions

Itemization Types

- Consider the problem of doubling or rotating an image using arrow keystrokes
- The image is rotated 90, 180, or 270 degrees using, respectively, the right, down, and left arrow keys
- The image is doubled using the up arrow key
- How keys are keys represented and compared?

Making Decisions

Itemization Types

- Consider the problem of doubling or rotating an image using arrow keystrokes
- The image is rotated 90, 180, or 270 degrees using, respectively, the right, down, and left arrow keys
- The image is doubled using the up arrow key
- How keys are keys represented and compared?
- In the `universe` teachpack, all keys are represented with a string
- *z* key is represented by `"z"`
- Right, down, left, and up keys: `"right"`, `"down`, `"left"`, and `"up"`

Making Decisions

Itemization Types

- Consider the problem of doubling or rotating an image using arrow keystrokes
- The image is rotated 90, 180, or 270 degrees using, respectively, the right, down, and left arrow keys
- The image is doubled using the up arrow key
- How keys are keys represented and compared?
- In the `universe` teachpack, all keys are represented with a string
- *z* key is represented by `"z"`
- Right, down, left, and up keys: `"right"`, `"down`, `"left"`, and `"up"`
- The `key=?` function is a predicate used to compare keys for equality

Making Decisions

Itemization Types

```
;; An alphanumeric keystroke (aks) is a member of either:
;;  1. ["a".."z"] --means a letter keystroke
;;  2. ["0".."9"] --means a numeric keystroke

;; A change image keystroke (ciks) is either:
;;  1. "right" --means rotate image 90 degrees clockwise
;;  2. "down"  --means rotate image 180 degrees clockwise
;;  3. "left"  --means rotate image 270 degrees clockwise
;;  4. "up"    --means double image size
;;  5. aks     --means do nothing to image
```

Making Decisions

Itemization Types

```
;; ciks ... --> ...   Purpose: ...
(define (f-on-ciks a-ciks ...)
  (cond [(key=? a-ciks "right") ...]
        [(key=? a-ciks "down")  ...]
        [(key=? a-ciks "left")  ...]
        [(key=? a-ciks "up")    ...]
        [else ...]))
;; Sample expressions for f-on-ciks
(define CIKS-RIGHT ...)  (define CIKS-DOWN  ...)
(define CIKS-LEFT ...)   (define CIKS-UP ...))
(define CIKS-ALPHANUM ...))
;; Tests using sample computations for f-on-ciks
(check-expect (f-on-ciks "right" ...) CIKS-RIGHT)
(check-expect (f-on-ciks "down" ...)  CIKS-DOWN)
(check-expect (f-on-ciks "left" ...)  CIKS-LEFT)
(check-expect (f-on-ciks "up" ...)    CIKS-UP)
(check-expect (f-on-ciks <aks> ...)   CIKS-ALPHANUM)
;; Tests using sample values for f-on-ciks
(check-expect (f-on-ciks "left" ...)  ...)
(check-expect (f-on-ciks "right" ...) ...)
(check-expect (f-on-ciks "down" ...)  ...)
(check-expect (f-on-ciks "up" ...)    ...)
(check-expect (f-on-ciks <aks> ...)   ...)
```

Making Decisions

Itemization Types

```
;; aks ... --> ...
;; Purpose: ...
(define (f-on-aks an-aks ...)
  (cond [(string<=? "a" an-aks "b") ...]
        [(string<=? "0" an-aks "9")  ...]))

;; Sample expressions for f-on-aks
(define AKS-LETTER ...)
(define AKS-NUM  ...)

;; Tests using sample computations for f-on-aks
(check-expect (f-on-aks <aks1> ...) AKS-LETTER)
(check-expect (f-on-aks <aks2> ...) AKS-NUM)

;; Tests using sample values for f-on-aks
(check-expect (f-on-aks <aks3> ...)  ...)
(check-expect (f-on-aks <aks4> ...) ...)
```

Making Decisions

Itemization Types

```
;; STEP 3: Use the functions rotate and scale to compute a new imag
;; image ciks --> image
;; Purpose: Rotate 90/180/270 degrees, double or untouch image
(define (change-img an-img a-ciks)
  (cond [(key=? a-ciks "right") (rotate 90 an-img)]
        [(key=? a-ciks "down")  (rotate 180 an-img)]
        [(key=? a-ciks "left")  (rotate 270 an-img)]
        [(key=? a-ciks "up")    (scale 2 an-img)]
        [else an-img]))
Sample expressions for change-img
(define RRIGHT-RCKT (rotate 90 ROCKET-IMG))
(define RDOWN-RCKT  (rotate 180 ROCKET-IMG))
(define RLEFT-RCKT  (rotate 270 ROCKET-IMG))
(define DOUBLE-RCKT (scale 2 ROCKET-IMG))
;; Tests using sample computations for change-img
(check-expect (change-img ROCKET-IMG "right") RRIGHT-RCKT)
(check-expect (change-img ROCKET-IMG "down")  RDOWN-RCKT)
(check-expect (change-img ROCKET-IMG "left")  RLEFT-RCKT)
(check-expect (change-img ROCKET-IMG "up")    DOUBLE-RCKT)
(check-expect (change-img ROCKET-IMG "m")     ROCKET-IMG)
;; Tests using sample values for change-img
(check-expect (change-img (rectangle 10 30 'solid 'red) "left")
              (rectangle 30 10 'solid 'red))
(check-expect (change-img (ellipse 75 35 'outline 'pink) "right")
```

Making Decisions

Itemization Types

- The design of `change-img` did not reveal the need for auxiliary functions
- The template for `aks` was never used to design a function
- Not every problem requires every template to be specialized
- Others problems involving `ciks` may very well require the specialization of the `aks` function template

Making Decisions

Itemization Types

- The design of `change-img` did not reveal the need for auxiliary functions
- The template for `aks` was never used to design a function
- Not every problem requires every template to be specialized
- Others problems involving `ciks` may very well require the specialization of the `aks` function template
- Think of the templates developed as different tools in a toolbox
- Sometimes you need a hammer
- Other times you need a wrench
- Yet other times you need both.

Making Decisions

Homework

- Problems: 68–69

Aliens Attack Version 1

- Designing them can be fun despite being complex programs
- To manage this complexity video games are developed using the process of iterative refinement

Aliens Attack Version 1

- Designing them can be fun despite being complex programs
- To manage this complexity video games are developed using the process of iterative refinement
- Good idea to get a handle on exactly what a video game is
- Similar to an animation
- Players to interact with the animation
- The game changes when a *computer event* occurs

Aliens Attack Version 1

- Designing them can be fun despite being complex programs
- To manage this complexity video games are developed using the process of iterative refinement
- Good idea to get a handle on exactly what a video game is
- Similar to an animation
- Players to interact with the animation
- The game changes when a *computer event* occurs
- A computer event includes pressing a key, the clock ticking, or clicking the mouse
- In Aliens Attack a player may use the left arrow and the right arrow keys to move the rocket.
- Functions called *event handlers* (or simply *handlers*) are needed to process computer events

Aliens Attack Version 1

- Designing them can be fun despite being complex programs
- To manage this complexity video games are developed using the process of iterative refinement
- Good idea to get a handle on exactly what a video game is
- Similar to an animation
- Players to interact with the animation
- The game changes when a *computer event* occurs
- A computer event includes pressing a key, the clock ticking, or clicking the mouse
- In Aliens Attack a player may use the left arrow and the right arrow keys to move the rocket.
- Functions called *event handlers* (or simply *handlers*) are needed to process computer events
- Commonly, handlers need to make decisions (distinguish what key has been pressed)

Aliens Attack Version 1

- Designing them can be fun despite being complex programs
- To manage this complexity video games are developed using the process of iterative refinement
- Good idea to get a handle on exactly what a video game is
- Similar to an animation
- Players to interact with the animation
- The game changes when a *computer event* occurs
- A computer event includes pressing a key, the clock ticking, or clicking the mouse
- In Aliens Attack a player may use the left arrow and the right arrow keys to move the rocket.
- Functions called *event handlers* (or simply *handlers*) are needed to process computer events
- Commonly, handlers need to make decisions (distinguish what key has been pressed)
- The `universe` teachpack provides programmers with the ability to write video games in `BSL`
- Provides syntax to associate handlers with computer events and it defines the signature that these handlers must have
- Using the `universe` teachpack is an exercise in programming with (the abilities provided by and the restrictions imposed by) an API
- Programming with an API is a good skill to develop because programming with `APIs` is standard practice

Aliens Attack Version 1

The Universe Teachpack

- The `universe` teachpack requires the development of a data definition for a *world*
- A world is all the values that may change during the evolution of the game

Aliens Attack Version 1

The Universe Teachpack

- The `universe` teachpack requires the development of a data definition for a *world*
- A world is all the values that may change during the evolution of the game
- `big-bang` expression syntax:

bb-expr	::=	(big-bang `expr` bb-clause$^+$)
bb-clause	::=	[to-draw function] ← required
		[on-key function]
		[on-tick function]
		[stop-when function]
		[name `expr`]
		⋮

Aliens Attack Version 1

The Universe Teachpack

- The `universe` teachpack requires the development of a data definition for a *world*
- A world is all the values that may change during the evolution of the game
- `big-bang` expression syntax:

```
bb-expr    ::=  (big-bang  expr bb-clause+)
bb-clause  ::=  [to-draw function]  ← required
                [on-key function]
                [on-tick function]
                [stop-when function]
                [name expr]
                    ⋮
```

- The `universe` API also specifies the signature and purpose of the handlers:

```
on-draw:    world → scene
            Purpose: To create the world's scene
on-key:     world key → world
            Purpose: To return the world after the given keystroke
on-tick:    world → world
            Purpose: To return the world after a clock tick
stop-when:  world → Boolean
            Purpose: To determine if the game/animation has ended
```

- Must write handlers that satisfy the above signatures
- Suggests using top-down design

Aliens Attack Version 1

A Video Game Design Recipe

1. Create data definitions for the elements that vary.
2. Develop a function template and examples for each data definition.
3. Define the `run` function with no tests.
   For a needed function:
4. Outline the computation.
5. Define constants for the value of sample expressions for each variety and name the differences.
6. Write the function's signature and purpose.
7. Write the function's header.
8. Write tests.
9. Write the function's body.
10. Run the tests and, if necessary, redesign.

Aliens Attack Version 1

A Video Game Design Recipe

- The `run` function's template may be defined as follows:

```
; string → world
; Purpose: To run the game
(define (run a-name)
  (big-bang INIT-WORLD
            [on-draw ...]
            [name a-name]
            bb-clause*))
```

- The final clauses, of course, may not contain a repetition of any claus

Aliens Attack Version 1

Adding the Rocket to Aliens Attack

-
- First refinement: add the rocket

Aliens Attack Version 1

Adding the Rocket to Aliens Attack

- First refinement: add the rocket
- Rocket may move left to right without going off the edges of the scene.
- Needs to be part of the world
- What changes about the rocket as the game advances?

Aliens Attack Version 1

Adding the Rocket to Aliens Attack

- First refinement: add the rocket
- Rocket may move left to right without going off the edges of the scene.
- Needs to be part of the world
- What changes about the rocket as the game advances?
- The rocket's `image-x` changes
- The rocket's `image-y` is constant

Aliens Attack Version 1

Adding the Rocket to Aliens Attack

```
#|  A rocket is an image-x
    ;; Sample instances of rocket
    (define ROCKET1 ...)
         ...
    ;; rocket ... → ...
    ;; Purpose: ...
    (define (f-on-rocket a-rocket ...)
      ...(f-on-image-x a-rocket ...) ...)
    ;; Sample expressions for f-on-rocket
    (define ROCKET1-VAL (f-on-rocket ROCKET1 ...))
         ...

    ;; Tests using sample computations for f-on-rocket
    (check-expect (f-on-rocket ROCKET1 ...) ROCKET-VAL)
        ...
    ;; Tests using sample values for f-on-rocket
    (check-expect (f-on-rocket ...) ...)
         ...                                                |#
```

Aliens Attack Version 1

Adding the Rocket to Aliens Attack

- ```
  #|  A rocket is an image-x
      ;; Sample instances of rocket
      (define ROCKET1 ...)
           ...
      ;; rocket ... → ...
      ;; Purpose: ...
      (define (f-on-rocket a-rocket ...)
        ...(f-on-image-x a-rocket ...) ...)
      ;; Sample expressions for f-on-rocket
      (define ROCKET1-VAL (f-on-rocket ROCKET1 ...))
           ...

      ;; Tests using sample computations for f-on-rocket
      (check-expect (f-on-rocket ROCKET1 ...) ROCKET-VAL)
          ...
      ;; Tests using sample values for f-on-rocket
      (check-expect (f-on-rocket ...) ...)
           ...                                                |#
  ```
- ```
  ;; Sample instances of rocket
  (define ROCKET1 7)
  (define INIT-ROCKET (/ MAX-CHARS-HORIZONTAL 2))
  ```

Aliens Attack Version 1

Adding the Rocket to Aliens Attack

- Observe that to process a `rocket` the template suggests calling a function to process an `image-x`
- This design exhibits *separation of concerns*
- Separation of concerns reflects that solving a problem for one data type is different from solving a problem for another data type

Aliens Attack Version 1

Adding the Rocket to Aliens Attack

- Observe that to process a `rocket` the template suggests calling a function to process an `image-x`
- This design exhibits *separation of concerns*
- Separation of concerns reflects that solving a problem for one data type is different from solving a problem for another data type
- A similar piece of reasoning must be done to define the game's world
- What may changes as the game evolves?

Aliens Attack Version 1

Adding the Rocket to Aliens Attack

```
#|  A world is a rocket
    ;; Sample instances of world
    (define WORLD1 ...)

    ;; world ... --> ...
    ;; Purpose: ...
    (define (f-on-world a-world ...)
      ...(f-on-rocket a-world ...) ...)

    ;; Sample expressions for f-on-world
    (define WORLD1-VAL ...)

    ;; Tests using sample computations for f-on-world
    (check-expect (f-on-world WORLD1 ...) WORLD1-VAL)
         ...

    ;; Tests using sample values for f-on-world
    (check-expect (f-on-world ...) ...)
         ...                                          |#
```

Aliens Attack Version 1

Adding the Rocket to Aliens Attack

- ```
  #|  A world is a rocket
      ;; Sample instances of world
      (define WORLD1 ...)

      ;; world ... --> ...
      ;; Purpose: ...
      (define (f-on-world a-world ...)
        ...(f-on-rocket a-world ...) ...)

      ;; Sample expressions for f-on-world
      (define WORLD1-VAL ...)

      ;; Tests using sample computations for f-on-world
      (check-expect (f-on-world WORLD1 ...) WORLD1-VAL)
           ...

      ;; Tests using sample values for f-on-world
      (check-expect (f-on-world ...) ...)
           ...                                          |#
  ```
- ```
  ;; Sample instances of world
  (define WORLD1      ROCKET1)
  (define INIT-WORLD  INIT-ROCKET)
  ```

Aliens Attack Version 1

Adding the Rocket to Aliens Attack

- It is necessary to define what a key is

Aliens Attack Version 1

Adding the Rocket to Aliens Attack

- It is necessary to define what a key is
- A `key` may be defined as an itemization type

```
;; Sample instances of key
(define KEY1 "right") (define KEY2 "left") (define KEY3 "m")
```

Aliens Attack Version 1

Adding the Rocket to Aliens Attack

- It is necessary to define what a key is
- A key may be defined as an itemization type
-
```
|#   A key is either:
      1. "right"  2. "left"  3. not "right" or "left"
    ;; Sample instances of key
    (define KEY1 ...)  (define KEY2 ...)  (define KEY3 ...)
    ;; key ... --> ...   Purpose: ...
    (define (f-on-key a-key ...)
      (cond [(key=? a-key "right") ...]
            [(key=? a-key "left") ...]
            [else ...]))
    ;; Sample expressions for f-on-key
    (define KEY1-VAL ...KEY1...)
    (define KEY2-VAL ...KEY2...)
    (define KEY3-VAL ...KEY3...)
    ;; Tests using sample computations for f-on-key
    (check-expect (f-on-key KEY1 ...) KEY1-VAL)
    (check-expect (f-on-key KEY2 ...) KEY2-VAL)
    (check-expect (f-on-key KEY3 ...) KEY3-VAL) ...
    ;; Tests using sample values for f-on-key
    (check-expect (f-on-key ...) ...) ...     |#
```
-
```
    ;; Sample instances of key
    (define KEY1 "right") (define KEY2 "left") (define KEY3 "m")
```

Aliens Attack Version 1

Adding the Rocket to Aliens Attack

- The `run` template specialization to complete Step 3

Aliens Attack Version 1

Adding the Rocket to Aliens Attack

- The `run` template specialization to complete Step 3
- The player moves the rocket using the left and right arrows keys
- An `on-key` clause is also needed to process key events

Aliens Attack Version 1

Adding the Rocket to Aliens Attack

- The `run` template specialization to complete Step 3
- The player moves the rocket using the left and right arrows keys
- An `on-key` clause is also needed to process key events
-

```
; string → world
; Purpose: To run the game
(define (run a-name)
  (big-bang INIT-WORLD
            [on-draw draw-world]
            [name a-name]
            [on-key process-key]))
```

Aliens Attack Version 1

Adding the Rocket to Aliens Attack

- Start with `draw-world` by specializing the template for a function on a `world`

Aliens Attack Version 1

Adding the Rocket to Aliens Attack

- Start with `draw-world` by specializing the template for a function on a `world`

-
```
;; Sample expressions for draw-world
(define WORLD-SCN1 (draw-rocket INIT-WORLD   E-SCENE))
(define WORLD-SCN2 (draw-rocket INIT-WORDLD2 E-SCENE))
```

Aliens Attack Version 1

Adding the Rocket to Aliens Attack

- Start with `draw-world` by specializing the template for a function on a `world`
- ```
  ;; world → scene   Purpose: To draw the world in E-SCENE
  (define (draw-world a-world)
  ```
- ```
  ;; Sample expressions for draw-world
  (define WORLD-SCN1 (draw-rocket INIT-WORLD   E-SCENE))
  (define WORLD-SCN2 (draw-rocket INIT-WORDLD2 E-SCENE))
  ```

Aliens Attack Version 1

Adding the Rocket to Aliens Attack

- Start with `draw-world` by specializing the template for a function on a `world`
- ```
  ;; world → scene   Purpose: To draw the world in E-SCENE
  (define (draw-world a-world)
  ```
- ```
  ;; Sample expressions for draw-world
  (define WORLD-SCN1 (draw-rocket INIT-WORLD   E-SCENE))
  (define WORLD-SCN2 (draw-rocket INIT-WORDLD2 E-SCENE))
  ```
- ```
  ;; Tests using sample computations for draw-world
  (check-expect (draw-world INIT-WORLD)  WORLD-SCN1)
  (check-expect (draw-world INIT-WORLD2) WORLD-SCN2)
  ;; Tests using sample computations for draw-world
  ```

(check-expect (draw-world 0))

(check-expect (draw-world (sub1 MAX-CHARS-HORIZONTAL)))
  ```

Aliens Attack Version 1

Adding the Rocket to Aliens Attack

- Start with `draw-world` by specializing the template for a function on a `world`

```
;; world → scene   Purpose: To draw the world in E-SCENE
(define (draw-world a-world)
  (draw-rocket a-world E-SCENE))
;; Sample expressions for draw-world
(define WORLD-SCN1 (draw-rocket INIT-WORLD   E-SCENE))
(define WORLD-SCN2 (draw-rocket INIT-WORDLD2 E-SCENE))
;; Tests using sample computations for draw-world
(check-expect (draw-world INIT-WORLD)  WORLD-SCN1)
(check-expect (draw-world INIT-WORLD2) WORLD-SCN2)
;; Tests using sample computations for draw-world
```

```
(check-expect (draw-world 0)
```

```
)
(check-expect (draw-world (sub1 MAX-CHARS-HORIZONTAL))
```

```
)
```

Aliens Attack Version 1

Adding the Rocket to Aliens Attack

- Design of the auxiliary function `draw-rocket` by specializing the template for functions on a rocket

Aliens Attack Version 1

Adding the Rocket to Aliens Attack

- Design of the auxiliary function `draw-rocket` by specializing the template for functions on a rocket

- ```
  ;; Sample expressions for draw-rocket
  (define RSCN3  (draw-ci ROCKET-IMG 3 ROCKET-Y E-SCENE))
  (define RSCN12 (draw-ci ROCKET-IMG 12 ROCKET-Y E-SCENE2))
  ```
  ```

Aliens Attack Version 1

Adding the Rocket to Aliens Attack

- Design of the auxiliary function `draw-rocket` by specializing the template for functions on a rocket
- ```
  ;; rocket scene → scene  Purpose: Draw rocket in given scene
  (define (draw-rocket a-rocket a-scene)
  ```
- ```
  ;; Sample expressions for draw-rocket
  (define RSCN3  (draw-ci ROCKET-IMG 3 ROCKET-Y E-SCENE))
  (define RSCN12 (draw-ci ROCKET-IMG 12 ROCKET-Y E-SCENE2))
  ```

Aliens Attack Version 1

Adding the Rocket to Aliens Attack

- Design of the auxiliary function `draw-rocket` by specializing the template for functions on a rocket
- ```
  ;; rocket scene → scene  Purpose: Draw rocket in given scene
  (define (draw-rocket a-rocket a-scene)
  ```
- ```
  ;; Sample expressions for draw-rocket
  (define RSCN3  (draw-ci ROCKET-IMG 3 ROCKET-Y E-SCENE))
  (define RSCN12 (draw-ci ROCKET-IMG 12 ROCKET-Y E-SCENE2))
  ```
- ```
  ;; Tests using sample computations for draw-rocket
  (check-expect (draw-rocket 3  E-SCENE) RSCN3)
  (check-expect (draw-rocket 12 E-SCENE2) RSCN12)
  ;; Tests using sample values for draw-rocket
  (check-expect (draw-rocket 0 E-SCENE)
  ```

```
)
(check-expect (draw-rocket (sub1 MAX-CHARS-HORIZONTAL)
                           E-SCENE)
```

```
)
```
```

Aliens Attack Version 1

Adding the Rocket to Aliens Attack

- Design of the auxiliary function `draw-rocket` by specializing the template for functions on a rocket

```
;; rocket scene → scene  Purpose: Draw rocket in given scene
(define (draw-rocket a-rocket a-scene)
  (draw-ci ROCKET-IMG a-rocket ROCKET-Y a-scene))
```

```
;; Sample expressions for draw-rocket
(define RSCN3  (draw-ci ROCKET-IMG 3 ROCKET-Y E-SCENE))
(define RSCN12 (draw-ci ROCKET-IMG 12 ROCKET-Y E-SCENE2))
```

```
;; Tests using sample computations for draw-rocket
(check-expect (draw-rocket 3  E-SCENE) RSCN3)
(check-expect (draw-rocket 12 E-SCENE2) RSCN12)
;; Tests using sample values for draw-rocket
(check-expect (draw-rocket 0 E-SCENE)
```

```
)
(check-expect (draw-rocket (sub1 MAX-CHARS-HORIZONTAL)
                           E-SCENE)
```

```
)
```

Aliens Attack Version 1

Adding the Rocket to Aliens Attack

- The design of `process-key` is done by specializing the `f-on-key` template

Aliens Attack Version 1

Adding the Rocket to Aliens Attack

- The design of `process-key` is done by specializing the `f-on-key` template

- ```
  ;; Sample expressions for process-key
  (define KEY-RVAL (move-rckt-right INIT-WORLD))
  (define KEY-LVAL (move-rckt-left INIT-WORLD))
  (define KEY-OVAL INIT-WORLD2)
  ```
  ```

Aliens Attack Version 1

Adding the Rocket to Aliens Attack

- The design of `process-key` is done by specializing the `f-on-key` template
- ```
  ;; world key → world
  ;; Purpose: Process a key event to return next world
  (define (process-key a-world a-key)
  ```
- ```
  ;; Sample expressions for process-key
  (define KEY-RVAL (move-rckt-right INIT-WORLD))
  (define KEY-LVAL (move-rckt-left INIT-WORLD))
  (define KEY-OVAL INIT-WORLD2)
  ```

Aliens Attack Version 1

Adding the Rocket to Aliens Attack

- The design of `process-key` is done by specializing the `f-on-key` template
- ```
  ;; world key → world
  ;; Purpose: Process a key event to return next world
  (define (process-key a-world a-key)
  ```
- ```
  ;; Sample expressions for process-key
  (define KEY-RVAL (move-rckt-right INIT-WORLD))
  (define KEY-LVAL (move-rckt-left INIT-WORLD))
  (define KEY-OVAL INIT-WORLD2)
  ```
- ```
  ;; Tests using sample computations for process-key
  (check-expect (process-key INIT-WORLD  "right") KEY-RVAL)
  (check-expect (process-key INIT-WORLD  "left")  KEY-LVAL)
  (check-expect (process-key INIT-WORLD2 "m")     KEY-OVAL)

  ;; Tests using sample values for process-key
  (check-expect (process-key (sub1 MAX-CHARS-HORIZONTAL) "right")
                (sub1 MAX-CHARS-HORIZONTAL))
  (check-expect (process-key 0 "left")  0)
  (check-expect (process-key 0 "o") 0)
  (check-expect (process-key 11 ";") 11)
  ```
  ```

Aliens Attack Version 1

Adding the Rocket to Aliens Attack

- The design of `process-key` is done by specializing the `f-on-key` template
- ```
  ;; world key → world
  ;; Purpose: Process a key event to return next world
  (define (process-key a-world a-key)
  ```
- ```
  (cond [(key=? a-key "right") (move-rckt-right a-world)]
        [(key=? a-key "left")  (move-rckt-left a-world)]
        [else a-world]))
  ```
- ```
  ;; Sample expressions for process-key
  (define KEY-RVAL (move-rckt-right INIT-WORLD))
  (define KEY-LVAL (move-rckt-left INIT-WORLD))
  (define KEY-OVAL INIT-WORLD2)
  ```
- ```
  ;; Tests using sample computations for process-key
  (check-expect (process-key INIT-WORLD  "right") KEY-RVAL)
  (check-expect (process-key INIT-WORLD  "left")  KEY-LVAL)
  (check-expect (process-key INIT-WORLD2 "m")     KEY-OVAL)

  ;; Tests using sample values for process-key
  (check-expect (process-key (sub1 MAX-CHARS-HORIZONTAL) "right")
                (sub1 MAX-CHARS-HORIZONTAL))
  (check-expect (process-key 0 "left")  0)
  (check-expect (process-key 0 "o") 0)
  (check-expect (process-key 11 ";") 11)
  ```

Aliens Attack Version 1

Adding the Rocket to Aliens Attack

- `move-rckt-right` requires defining how to move a rocket right
- At a first glance, it may appear that adding 1 to the rocket suffices
- Can the rocket always be moved to the right?

Aliens Attack Version 1

Adding the Rocket to Aliens Attack

- `move-rckt-right` requires defining how to move a rocket right
- At a first glance, it may appear that adding 1 to the rocket suffices
- Can the rocket always be moved to the right?
- Not when the rocket is `MAX-CHARS-HORIZONTAL - 1`

Aliens Attack Version 1

Adding the Rocket to Aliens Attack

- `move-rckt-right` requires defining how to move a rocket right
- At a first glance, it may appear that adding 1 to the rocket suffices
- Can the rocket always be moved to the right?
- Not when the rocket is `MAX-CHARS-HORIZONTAL - 1`

- ```
  ;; Sample expressions for move-rckt-right
  (define ROCKET-VAL3 (add1 3))
  (define ROCKET-VALMAX (sub1 MAX-CHARS-HORIZONTAL))
  ```
  ```

Aliens Attack Version 1

Adding the Rocket to Aliens Attack

- `move-rckt-right` requires defining how to move a rocket right
- At a first glance, it may appear that adding 1 to the rocket suffices
- Can the rocket always be moved to the right?
- Not when the rocket is `MAX-CHARS-HORIZONTAL - 1`
-
```
;; rocket → rocket
;; Purpose: Move the given rocket right
(define (move-rckt-right a-rocket)
```
-
```
;; Sample expressions for move-rckt-right
(define ROCKET-VAL3 (add1 3))
(define ROCKET-VALMAX (sub1 MAX-CHARS-HORIZONTAL))
```

Aliens Attack Version 1

Adding the Rocket to Aliens Attack

- `move-rckt-right` requires defining how to move a rocket right
- At a first glance, it may appear that adding 1 to the rocket suffices
- Can the rocket always be moved to the right?
- Not when the rocket is `MAX-CHARS-HORIZONTAL - 1`
-
```
;; rocket → rocket
;; Purpose: Move the given rocket right
(define (move-rckt-right a-rocket)
```
-
```
;; Sample expressions for move-rckt-right
(define ROCKET-VAL3 (add1 3))
(define ROCKET-VALMAX (sub1 MAX-CHARS-HORIZONTAL))
```
-
```
;; Tests using sample computations for move-rckt-right
(check-expect (move-rckt-right 3) ROCKET-VAL3)
(check-expect (move-rckt-right (sub1 MAX-CHARS-HORIZONTAL))
              ROCKET-VALMAX)

;; Tests using sample values for move-rckt-right
(check-expect (move-rckt-right 0) 1)
(check-expect (move-rckt-right 15) 16)
```

Aliens Attack Version 1

Adding the Rocket to Aliens Attack

- `move-rckt-right` requires defining how to move a rocket right
- At a first glance, it may appear that adding 1 to the rocket suffices
- Can the rocket always be moved to the right?
- Not when the rocket is `MAX-CHARS-HORIZONTAL - 1`

```
;; rocket → rocket
;; Purpose: Move the given rocket right
(define (move-rckt-right a-rocket)
  (if (< a-rocket (sub1 MAX-CHARS-HORIZONTAL))
      (add1 a-rocket)
      a-rocket]))
```

```
;; Sample expressions for move-rckt-right
(define ROCKET-VAL3 (add1 3))
(define ROCKET-VALMAX (sub1 MAX-CHARS-HORIZONTAL))
```

```
;; Tests using sample computations for move-rckt-right
(check-expect (move-rckt-right 3) ROCKET-VAL3)
(check-expect (move-rckt-right (sub1 MAX-CHARS-HORIZONTAL))
              ROCKET-VALMAX)

;; Tests using sample values for move-rckt-right
(check-expect (move-rckt-right 0) 1)
(check-expect (move-rckt-right 15) 16)
```

Aliens Attack Version 1

Adding the Rocket to Aliens Attack

- A similar problem analysis leads to concluding that a conditional is needed to move a rocket left

Aliens Attack Version 1

Adding the Rocket to Aliens Attack

- A similar problem analysis leads to concluding that a conditional is needed to move a rocket left

- ```
  ;; Sample expressions for move-rckt-left
  (define ROCKET-VAL7 (sub1 7))
  (define ROCKET-VAL0 0)
  ```
  ```

Aliens Attack Version 1

Adding the Rocket to Aliens Attack

- A similar problem analysis leads to concluding that a conditional is needed to move a rocket left
- ```
  ;; rocket → rocket
  ;; Purpose: Move the given rocket left
  (define (move-rckt-left a-rocket)
  ```
- ```
  ;; Sample expressions for move-rckt-left
  (define ROCKET-VAL7 (sub1 7))
  (define ROCKET-VAL0 0)
  ```

Aliens Attack Version 1

Adding the Rocket to Aliens Attack

- A similar problem analysis leads to concluding that a conditional is needed to move a rocket left
- ```
  ;; rocket → rocket
  ;; Purpose: Move the given rocket left
  (define (move-rckt-left a-rocket)
  ```
- ```
  ;; Sample expressions for move-rckt-left
  (define ROCKET-VAL7 (sub1 7))
  (define ROCKET-VAL0 0)
  ```
- ```
  ;; Tests using sample computations for move-rckt-left
  (check-expect (move-rckt-left 7) ROCKET-VAL7)
  (check-expect (move-rckt-left 0) 0)

  ;; Tests using sample values for move-rckt-left
  (check-expect (move-rckt-left (sub1 MAX-CHARS-HORIZONTAL))
                (- MAX-CHARS-HORIZONTAL 2))
  (check-expect (move-rckt-left 14) 13)
  ```
  ```

Aliens Attack Version 1

Adding the Rocket to Aliens Attack

- A similar problem analysis leads to concluding that a conditional is needed to move a rocket left
- ```
  ;; rocket → rocket
  ;; Purpose: Move the given rocket left
  (define (move-rckt-left a-rocket)
  ```
- ```
    (if (> a-rocket 0)
        (sub1 a-rocket)
        a-rocket))
  ```
- ```
  ;; Sample expressions for move-rckt-left
  (define ROCKET-VAL7 (sub1 7))
  (define ROCKET-VAL0 0)
  ```
- ```
  ;; Tests using sample computations for move-rckt-left
  (check-expect (move-rckt-left 7) ROCKET-VAL7)
  (check-expect (move-rckt-left 0) 0)

  ;; Tests using sample values for move-rckt-left
  (check-expect (move-rckt-left (sub1 MAX-CHARS-HORIZONTAL))
                (- MAX-CHARS-HORIZONTAL 2))
  (check-expect (move-rckt-left 14) 13)
  ```

Aliens Attack Version 1

Homework

- Problems: 71, 77
- Quiz: Problem 76 (due in 1 week)