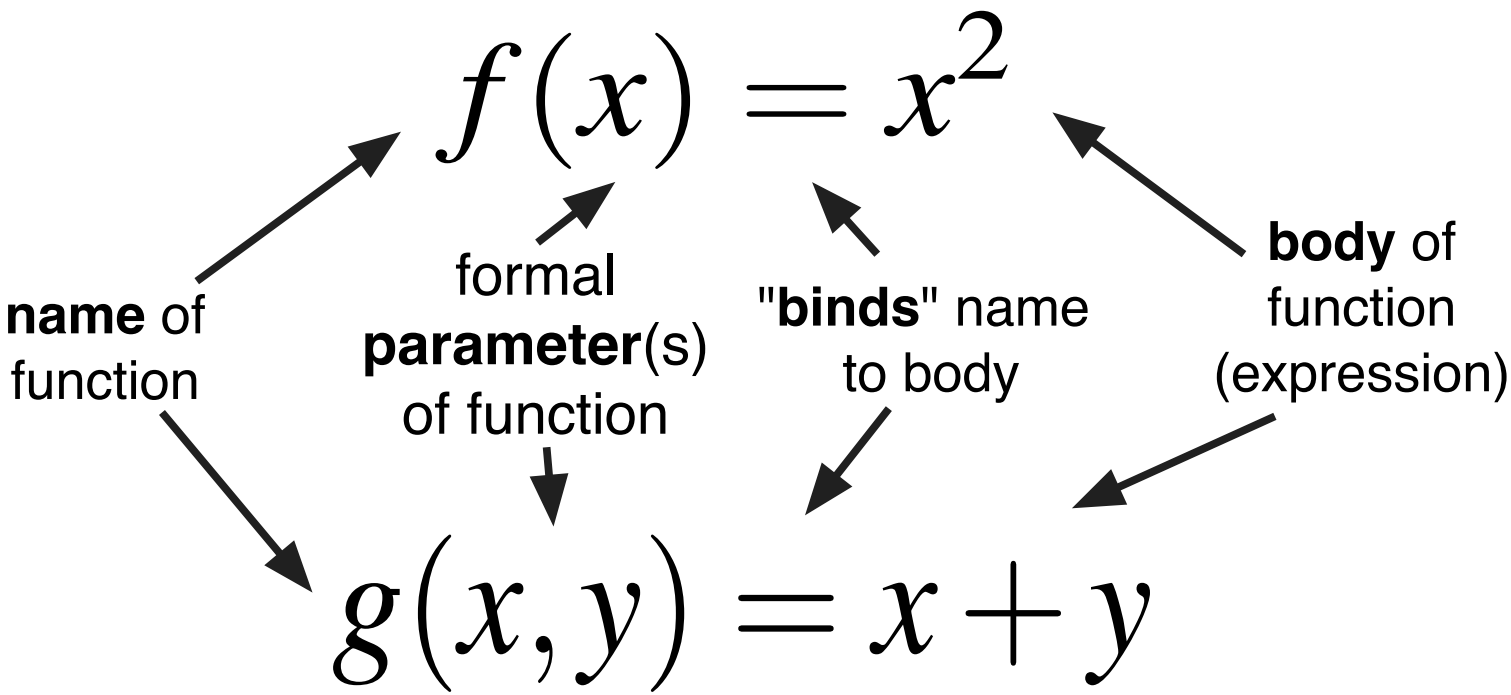
1: Introduction to Racket

Defining Functions in Mathematics



Defining functions in mathematics

Let $f(x) = x^2$,
$g(x, y) = x + y$

An **application** of a function:
$g(1, 3)$

Let $f(x) = x^2$,
$g(x, y) = x + y$

An **application** of a function:
$g(1, 3)$

$$g(x, y) = x + y$$

parameters body

An application supplies **arguments** for the parameters, which are **substituted** into the body expression.

Let $f(x) = x^2$,
$g(x, y) = x + y$

An **application** of a function:
$g(1, 3)$

$$g(1,3) = 1 + 3$$
$$= 4$$

$$g(x,y) = x + y$$

parameters body

An application supplies **arguments** for the parameters, which are **substituted** into the body expression.

Let $f(x) = x^2$,
$g(x, y) = x + y$

An **application** of a function:
$g(1, 3)$

$$g(1, 3) = 1 + 3$$
$$= 4$$

$$g(x, y) = x + y$$

parameters body

An application supplies **arguments** for the parameters, which are **substituted** into the body expression.

Evaluating Expressions

In a larger expression, the arguments of an application may themselves be applications.

Evaluating Expressions

In a larger expression, the arguments of an application may themselves be applications.

Let $f(x) = x^2$,
$g(x, y) = x + y$

Evaluating Expressions

In a larger expression, the arguments of an application may themselves be applications.

Let $f(x) = x^2$,
$g(x, y) = x + y$

$$\begin{aligned} g(g(1,3), f(2)) &= g(4, f(2)) \\ &= g(4,4) \\ &= 4 + 4 \\ &= 8 \end{aligned}$$

Evaluating Expressions

In a larger expression, the arguments of an application may themselves be applications.

Let $f(x) = x^2$,
$g(x, y) = x + y$

$$\begin{aligned} g(g(1,3), f(2)) &= g(4, f(2)) \\ &= g(4,4) \\ &= 4 + 4 \\ &= 8 \end{aligned}$$

Evaluating Expressions

In a larger expression, the arguments of an application may themselves be applications.

Let $f(x) = x^2$,
$g(x, y) = x + y$

$$\begin{aligned} g(g(1,3),f(2)) &= g(4,f(2)) \\ &= g(4,4) \\ &= 4+4 \\ &= 8 \end{aligned}$$

Evaluating Expressions

In a larger expression, the arguments of an application may themselves be applications.

Let $f(x) = x^2$,
$g(x, y) = x + y$

$$\begin{aligned} g(g(1,3),f(2)) &= g(4,f(2)) \\ &= g(4,4) \\ &= 4+4 \\ &= 8 \end{aligned}$$

Evaluating Expressions

In a larger expression, the arguments of an application may themselves be applications.

Let $f(x) = x^2$,
$g(x, y) = x + y$

$$\begin{aligned} g(g(1,3),f(2)) &= g(4,f(2)) \\ &= g(4,4) \\ &= 4+4 \\ &= 8 \end{aligned}$$

Although different orderings are possible, we will evaluate these functions left-to-right, choosing to evaluate the innermost function application first.

This will be enforced!

Correct vs. Incorrect Evaluations

Mathematically, we could instead substitute the unsimplified arguments into the body expression.

Mathematically, we could instead substitute the unsimplified arguments into the body expression.

Comparison

Working from the outside in:

$$\begin{aligned} g(g(1,3),f(2)) &= g(1,3)+f(2) \\ &= 4+f(2) \\ &= 4+4 \\ &= 8 \end{aligned}$$

Working from the inside out:

$$\begin{aligned} g(g(1,3),f(2)) &= g(4,f(2)) \\ &= g(4,4) \\ &= 4+4 \\ &= 8 \end{aligned}$$

Correct vs. Incorrect Evaluations

Mathematically, we could instead substitute the unsimplified arguments into the body expression.

Comparison

Working from the outside in:

$$\begin{aligned} g(g(1,3),f(2)) &= g(1,3)+f(2) \\ &= 4+f(2) \\ &= 4+4 \\ &= 8 \end{aligned}$$

Working from the inside out:

$$\begin{aligned} g(g(1,3),f(2)) &= g(4,f(2)) \\ &= g(4,4) \\ &= 4+4 \\ &= 8 \end{aligned}$$

We now disallow the “outside in” kind of substitution.

Mathematically, we could instead substitute the unsimplified arguments into the body expression.

Comparison

Working from the outside in:

$$\begin{aligned} g(g(1,3),f(2)) &= g(1,3)+f(2) \\ &= 4+f(2) \\ &= 4+4 \\ &= 8 \end{aligned}$$

Working from the inside out:

$$\begin{aligned} g(g(1,3),f(2)) &= g(4,f(2)) \\ &= g(4,4) \\ &= 4+4 \\ &= 8 \end{aligned}$$

We now disallow the “outside in” kind of substitution.

We insist that we only substitute, in a function application, when all of its arguments are values (not more general expressions).

Function application in Racket

The **syntax** of function application in Racket is different from that of mathematics.

In Racket, the name of the function follows the open parenthesis, and arguments are separated by spaces.

$g(1,3)$ becomes, in Racket, `(g 1 3)`.

$g(g(1,3),f(2))$ becomes `(g (g 1 3) (f 2))`.

Operators in mathematics

In arithmetic expressions, we often place operators between their operands.

- $3 - 2 + 4/5$

Operators in mathematics

In arithmetic expressions, we often place operators between their operands.

- $3 - 2 + 4/5$

We have some rules (division before addition, left to right) to specify order of operation. Sometimes these do not suffice, and parentheses are required for grouping.

- $(6 - 4)/(5 + 7)$

Operators in mathematics

In arithmetic expressions, we often place operators between their operands.

- $3 - 2 + 4/5$

We have some rules (division before addition, left to right) to specify order of operation. Sometimes these do not suffice, and parentheses are required for grouping.

- $(6 - 4)/(5 + 7)$

We could use function notation for operations such as $+$.

- $3 - 2 + 4/5$ becomes $+(-(3, 2), /(4, 5))$

Operators in mathematics

In arithmetic expressions, we often place operators between their operands.

- $3 - 2 + 4/5$

We have some rules (division before addition, left to right) to specify order of operation. Sometimes these do not suffice, and parentheses are required for grouping.

- $(6 - 4)/(5 + 7)$

We could use function notation for operations such as $+$.

- $3 - 2 + 4/5$ becomes $+(-(3, 2), /(4, 5))$

Now parentheses are only used to associate arguments with operations, as they are no longer needed for ordering.

Operators in mathematics

In arithmetic expressions, we often place operators between their operands.

- $3 - 2 + 4/5$

We have some rules (division before addition, left to right) to specify order of operation. Sometimes these do not suffice, and parentheses are required for grouping.

- $(6 - 4)/(5 + 7)$

We could use function notation for operations such as $+$.

- $3 - 2 + 4/5$ becomes $+(-(3, 2), /(4, 5))$

Now parentheses are only used to associate arguments with operations, as they are no longer needed for ordering.

This is what is done in Racket!

Expressions in Racket

Operators in mathematics become functions in Racket.

Parentheses are used only to associate arguments with operations. Ordering of operations is handled by nesting.

- $3 - 2 + 4/5$ becomes `(+ (- 3 2) (/ 4 5))`
- $(6 - 4)(3 + 2)$ becomes `(* (- 6 4) (+ 3 2))`

Any arithmetic expression can be converted into a Racket expression in this way.

Expressions in Racket

Operators in mathematics become functions in Racket.

Parentheses are used only to associate arguments with operations. Ordering of operations is handled by nesting.

- $3 - 2 + 4/5$ becomes `(+ (- 3 2) (/ 4 5))`
- $(6 - 4)(3 + 2)$ becomes `(* (- 6 4) (+ 3 2))`

Any arithmetic expression can be converted into a Racket expression in this way.

Don't add extra parentheses!

Expressions in Racket

Operators in mathematics become functions in Racket.

Parentheses are used only to associate arguments with operations. Ordering of operations is handled by nesting.

- $3 - 2 + 4/5$ becomes `(+ (- 3 2) (/ 4 5))`
- $(6 - 4)(3 + 2)$ becomes `(* (- 6 4) (+ 3 2))`

Any arithmetic expression can be converted into a Racket expression in this way.

Don't add extra parentheses!

Extra parentheses are harmless in arithmetic expressions, but they are harmful in Racket.

Expressions in Racket

Operators in mathematics become functions in Racket.

Parentheses are used only to associate arguments with operations. Ordering of operations is handled by nesting.

- $3 - 2 + 4/5$ becomes `(+ (- 3 2) (/ 4 5))`
- $(6 - 4)(3 + 2)$ becomes `(* (- 6 4) (+ 3 2))`

Any arithmetic expression can be converted into a Racket expression in this way.

Don't add extra parentheses!

Extra parentheses are harmless in arithmetic expressions, but they are harmful in Racket.

Only use parentheses when necessary (to signal a function application or some other Racket syntax).

The DrRacket environment

- Two windows: Interactions and Definitions
- Interactions window: a read-evaluate-print loop (REPL)
- Can type in expressions and have them evaluated

The DrRacket environment

- Two windows: Interactions and Definitions
- Interactions window: a read-evaluate-print loop (REPL)
- Can type in expressions and have them evaluated

[Demonstration]

The DrRacket environment

- Two windows: Interactions and Definitions
- Interactions window: a read-evaluate-print loop (REPL)
- Can type in expressions and have them evaluated

[Demonstration]

Numbers in DrRacket

- Integers in Racket are represented exactly (unbounded).
- Rational numbers are represented exactly.
- Evaluating some expressions produces an **inexact** value.
  Example: `(sqrt 2)` evaluates to `#i1.414213562370951`.
  We will not use inexact numbers much (if at all).

Errors in expressions

What is wrong with each of the following? How might it be corrected?

- `(* (5) 3)`

Errors in expressions

What is wrong with each of the following? How might it be corrected?

- `(* (5) 3)`
- `(+ (* 2 4)`

Errors in expressions

What is wrong with each of the following? How might it be corrected?

- `(* (5) 3)`
- `(+ (* 2 4)`
- `(5 * 14)`

Errors in expressions

What is wrong with each of the following? How might it be corrected?

- `(* (5) 3)`
- `(+ (* 2 4)`
- `(5 * 14)`
- `(* + 3 5 2)`

Errors in expressions

What is wrong with each of the following? How might it be corrected?

- `(* (5) 3)`
- `(+ (* 2 4)`
- `(5 * 14)`
- `(* + 3 5 2)`
- `(/ 25 0)`

Defining Functions in Racket

$f(x) = x^2$ becomes `(define (f x) (* x x))`.

$g(x, y) = x + y$ becomes `(define (g x y) (+ x y))`

`define` is a **form**
(looks like a Racket function application, but treated differently).

Defining Functions in Racket

$f(x) = x^2$ becomes `(define (f x) (* x x))`.

$g(x, y) = x + y$ becomes `(define (g x y) (+ x y))`

`define` is a **form**
(looks like a Racket function application, but treated differently).

`define` **binds** a name to an expression
(which uses the parameters that follow the name).

Defining Functions in Racket

In general, a Racket function definition consists of:

- a name for the function,
- a list of parameters,
- a single “body” expression.

Defining Functions in Racket

In general, a Racket function definition consists of:

- a name for the function,
- a list of parameters,
- a single “body” expression.

The body expression typically uses the parameters together with other built-in and user-defined functions.

Defining Functions in Racket

In general, a Racket function definition consists of:

- a name for the function,
- a list of parameters,
- a single “body” expression.

The body expression typically uses the parameters together with other built-in and user-defined functions.

An application of a user-defined Racket function substitutes argument values for the corresponding parameters in the definition’s body expression.

Evaluating a Racket expression by hand

We use a process of substitution.

Evaluating a Racket expression by hand

We use a process of substitution.

Each step of the process is indicated using the “yields” symbol, written as $\Rightarrow$.

Evaluating a Racket expression by hand

We use a process of substitution.

Each step of the process is indicated using the “yields” symbol, written as $\Rightarrow$.

```
(* (- 6 4) (+ 3 2))
```

Evaluating a Racket expression by hand

We use a process of substitution.

Each step of the process is indicated using the “yields” symbol, written as $\Rightarrow$.

```
(* (- 6 4) (+ 3 2))
⇒  (* 2 (+ 3 2))
```

Evaluating a Racket expression by hand

We use a process of substitution.

Each step of the process is indicated using the “yields” symbol, written as $\Rightarrow$.

```
(* (- 6 4) (+ 3 2))
⇒  (* 2 (+ 3 2))
⇒  (* 2 5)
```

Evaluating a Racket expression by hand

We use a process of substitution.

Each step of the process is indicated using the “yields” symbol, written as $\Rightarrow$.

```
(* (- 6 4) (+ 3 2))
⇒  (* 2 (+ 3 2))
⇒  (* 2 5)
⇒  10
```

Evaluating a Racket expression by hand

We use a process of substitution.

Each step of the process is indicated using the “yields” symbol, written as $\Rightarrow$.

```
(* (- 6 4) (+ 3 2))
⇒  (* 2 (+ 3 2))
⇒  (* 2 5)
⇒  10
```

This mirrors how we work with mathematical expressions.

Evaluating a Racket expression by hand

The same process works with user-defined Racket functions, using substitution for function application as we described earlier.

```
(define (f x) (* x x))
(define (g x y) (+ x y))
(g (g 1 3) (f 2))
```

Evaluating a Racket expression by hand

The same process works with user-defined Racket functions, using substitution for function application as we described earlier.

```
(define (f x) (* x x))
(define (g x y) (+ x y))
(g (g 1 3) (f 2))
⇒  (g (+ 1 3) (f 2))
```

Evaluating a Racket expression by hand

The same process works with user-defined Racket functions, using substitution for function application as we described earlier.

```
(define (f x) (* x x))
(define (g x y) (+ x y))
(g (g 1 3) (f 2))
⇒  (g (+ 1 3) (f 2))
⇒  (g 4 (f 2))
```

Evaluating a Racket expression by hand

The same process works with user-defined Racket functions, using substitution for function application as we described earlier.

```
(define (f x) (* x x))
(define (g x y) (+ x y))
(g (g 1 3) (f 2))
⇒  (g (+ 1 3) (f 2))
⇒  (g 4 (f 2))
⇒  (g 4 (* 2 2))
```

Evaluating a Racket expression by hand

The same process works with user-defined Racket functions, using substitution for function application as we described earlier.

```
(define (f x) (* x x))
(define (g x y) (+ x y))
(g (g 1 3) (f 2))
⇒  (g (+ 1 3) (f 2))
⇒  (g 4 (f 2))
⇒  (g 4 (* 2 2))
⇒  (g 4 4)
```

Evaluating a Racket expression by hand

The same process works with user-defined Racket functions, using substitution for function application as we described earlier.

```
(define (f x) (* x x))
(define (g x y) (+ x y))
(g (g 1 3) (f 2))
⇒  (g (+ 1 3) (f 2))
⇒  (g 4 (f 2))
⇒  (g 4 (* 2 2))
⇒  (g 4 4)
⇒  (+ 4 4)
```

Evaluating a Racket expression by hand

The same process works with user-defined Racket functions, using substitution for function application as we described earlier.

```
(define (f x) (* x x))
(define (g x y) (+ x y))
(g (g 1 3) (f 2))
⇒ (g (+ 1 3) (f 2))
⇒ (g 4 (f 2))
⇒ (g 4 (* 2 2))
⇒ (g 4 4)
⇒ (+ 4 4)
⇒ 8
```

Parameters in Racket

Each parameter name has meaning only within the body of its function.

```
(define (f x y)
  (+ x y))

(define (g z x)
  (* x z))
```

The two uses of `x` are independent.

Defining constants in Racket

$k = 3$ becomes `(define k 3)`

$p = k^2$ becomes `(define p (* k k))`

Defining constants in Racket

$k = 3$ becomes `(define k 3)`

$p = k^2$ becomes `(define p (* k k))`

The effect of `(define k 3)` is to **bind** the name `k` to the value `3`.

Defining constants in Racket

$k = 3$ becomes `(define k 3)`

$p = k^2$ becomes `(define p (* k k))`

The effect of `(define k 3)` is to **bind** the name `k` to the value `3`.

In `(define p (* k k))`, the expression `(* k k)` is first evaluated to give `9`, and then `p` is bound to that value.

```
(define p (* k k))
```

Defining constants in Racket

$k = 3$ becomes `(define k 3)`

$p = k^2$ becomes `(define p (* k k))`

The effect of `(define k 3)` is to **bind** the name `k` to the value `3`.

In `(define p (* k k))`, the expression `(* k k)` is first evaluated to give `9`, and then `p` is bound to that value.

```
(define p (* k k))
⇒  (define p (* 3 k))
```

Defining constants in Racket

$k = 3$ becomes `(define k 3)`

$p = k^2$ becomes `(define p (* k k))`

The effect of `(define k 3)` is to **bind** the name `k` to the value `3`.

In `(define p (* k k))`, the expression `(* k k)` is first evaluated to give `9`, and then `p` is bound to that value.

```
(define p (* k k))
⇒  (define p (* 3 k))
⇒  (define p (* 3 3))
```

Defining constants in Racket

$k = 3$ becomes `(define k 3)`

$p = k^2$ becomes `(define p (* k k))`

The effect of `(define k 3)` is to **bind** the name `k` to the value `3`.

In `(define p (* k k))`, the expression `(* k k)` is first evaluated to give `9`, and then `p` is bound to that value.

```
(define p (* k k))
⇒  (define p (* 3 k))
⇒  (define p (* 3 3))
⇒  (define p 9)
```

Defining constants in Racket

In the body of a function, a parameter name “shadows” a constant of the same name.

```
(define x 3)

(define (f x y)
  (- x y))
(+ x x) ⇒ 6
```

Defining constants in Racket

In the body of a function, a parameter name “shadows” a constant of the same name.

```
(define x 3)

(define (f x y)
  (- x y))
(+ x x) ⇒ 6
(f 7 6) ⇒ (- 7 6)
```

Defining constants in Racket

In the body of a function, a parameter name “shadows” a constant of the same name.

```
(define x 3)

(define (f x y)
  (- x y))
(+ x x) ⇒ 6
(f 7 6) ⇒ (- 7 6) ⇒ 1
```

Defining constants in Racket

In the body of a function, a parameter name “shadows” a constant of the same name.

```
(define x 3)

(define (f x y)
  (- x y))
```

```
(+ x x) ⇒ 6
(f 7 6) ⇒ (- 7 6) ⇒ 1
(f 5 x) ⇒ (f 5 3)
```

Defining constants in Racket

In the body of a function, a parameter name “shadows” a constant of the same name.

```
(define x 3)

(define (f x y)
  (- x y))
```

```
(+ x x) ⇒ 6
(f 7 6) ⇒ (- 7 6) ⇒ 1
(f 5 x) ⇒ (f 5 3) ⇒ (- 5 3)
```

Defining constants in Racket

In the body of a function, a parameter name “shadows” a constant of the same name.

```
(define x 3)

(define (f x y)
  (- x y))
```

```
(+ x x) ⇒ 6
(f 7 6) ⇒ (- 7 6) ⇒ 1
(f 5 x) ⇒ (f 5 3) ⇒ (- 5 3) ⇒ 2
```

Advantages of constants

- Makes programs easier to understand
- Can give meaningful names to useful values
  (e.g. `interest-rate`, `passing-grade`, and `starting-salary`).
- Reduces typing and errors when such values need to be changed

Advantages of constants

- Makes programs easier to understand
- Can give meaningful names to useful values
  (e.g. `interest-rate`, `passing-grade`, and `starting-salary`).
- Reduces typing and errors when such values need to be changed

Constants can be used in any expression, including the body of function definitions.

Advantages of constants

- Makes programs easier to understand
- Can give meaningful names to useful values
  (e.g. `interest-rate`, `passing-grade`, and `starting-salary`).
- Reduces typing and errors when such values need to be changed

Constants can be used in any expression, including the body of function definitions.

They are sometimes called variables, but their values cannot be changed.

DrRacket's Definitions window

- Can accumulate definitions and expressions

DrRacket's Definitions window

- Can accumulate definitions and expressions
- Can save and restore Definitions window

DrRacket's Definitions window

- Can accumulate definitions and expressions
- Can save and restore Definitions window
- (Demonstration)

DrRacket's Definitions window

- Can accumulate definitions and expressions
- Can save and restore Definitions window
- (Demonstration)
- Provides a Stepper to let one evaluate expressions step-by-step

DrRacket's Definitions window

- Can accumulate definitions and expressions
- Can save and restore Definitions window
- (Demonstration)
- Provides a Stepper to let one evaluate expressions step-by-step
- Features: subexpression highlighting, syntax checking, error highlighting, automatic indentation

Programs in Racket

A Racket program is a sequence of definitions and expressions.

Programs in Racket

A Racket program is a sequence of definitions and expressions.

When the program is run using the Run button, the expressions are evaluated, using substitution, to produce values. These values are printed in the Interactions window.

Programs in Racket

A Racket program is a sequence of definitions and expressions.

When the program is run using the Run button, the expressions are evaluated, using substitution, to produce values. These values are printed in the Interactions window.

Further expressions using the definitions may be evaluated in the Interactions window.

Goals of this module

1. You should understand the basic syntax of Racket, how to form expressions properly, and what DrRacket might do when given an expression causing an error.

Goals of this module

1. You should understand the basic syntax of Racket, how to form expressions properly, and what DrRacket might do when given an expression causing an error.
2. You should be comfortable with these terms: function, parameter, application, argument, constant, expression, form.

Goals of this module

1. You should understand the basic syntax of Racket, how to form expressions properly, and what DrRacket might do when given an expression causing an error.
2. You should be comfortable with these terms: function, parameter, application, argument, constant, expression, form.
3. You should be able to define and use functions that do simple arithmetic computations.

Goals of this module

1. You should understand the basic syntax of Racket, how to form expressions properly, and what DrRacket might do when given an expression causing an error.
2. You should be comfortable with these terms: function, parameter, application, argument, constant, expression, form.
3. You should be able to define and use functions that do simple arithmetic computations.
4. You should understand the purposes and uses of the Definitions and Interactions windows in DrRacket.