

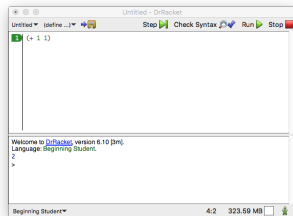
Not!

August 28, 2019

Dr. Racket

I encourage the use of Dr.
Racket:

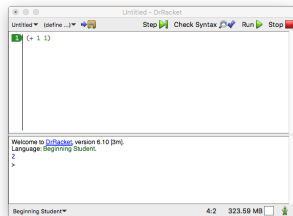
- The definitions window (top) is for writing programs



Dr. Racket

I encourage the use of Dr.
Racket:

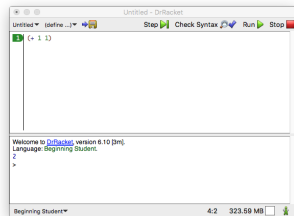
- The definitions window (top) is for writing programs
- The interactions window (bottom) is for interaction



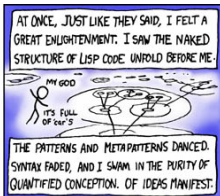
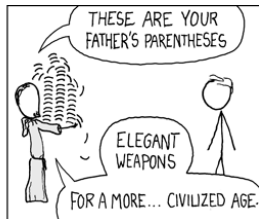
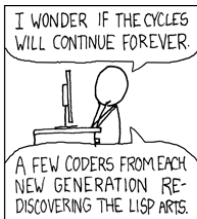
Dr. Racket

I encourage the use of Dr.
Racket:

- The definitions window (top) is for writing programs
- The interactions window (bottom) is for interaction
- Let me demonstrate



Relevant XKCDs



Syntax Primer

Functional Programming is about *functions*

- To call a function f , with argument 1 we write:

$(f\ 1)$

Syntax Primer

Functional Programming is about *functions*

- To call a function f , with argument 1 we write:

$(f\ 1)$

- Instead of $1+2$ we write:

$(+ 1\ 2)$

Syntax Primer

Functional Programming is about *functions*

- To call a function f , with argument 1 we write:

$(f\ 1)$

- Instead of $1+2$ we write:

$(+ 1\ 2)$

- There is no operator precedence. $1 + 2*3$ becomes:

$(+ 1\ (* 2\ 3))$

Syntax Primer

Functional Programming is about *functions*

- To call a function f , with argument 1 we write:

```
(f 1)
```

- Instead of $1+2$ we write:

```
(+ 1 2)
```

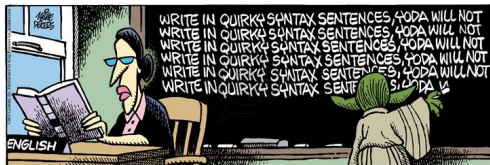
- There is no operator precedence. $1 + 2*3$ becomes:

```
(+ 1 (* 2 3))
```

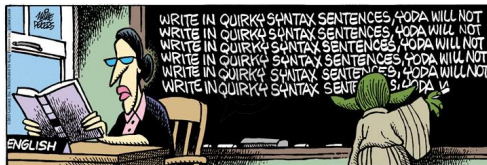
- To define a function that doubles numbers:

```
(define (double x) (* x 2))
```

Ewww Gross Syntax!

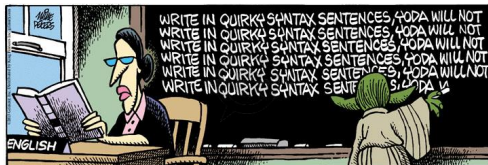


Ewww Gross Syntax!



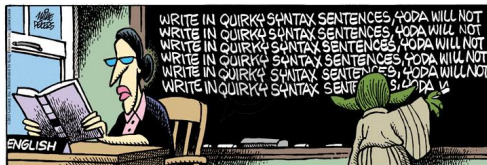
- Racket (Lisps in general) use prefix (polish) notation

Ewww Gross Syntax!



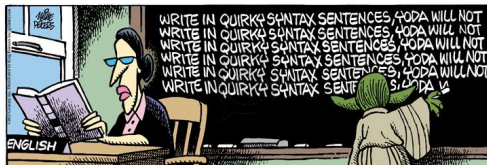
- Racket (Lisps in general) use prefix (polish) notation
- Tons of advantages:

Ewww Gross Syntax!



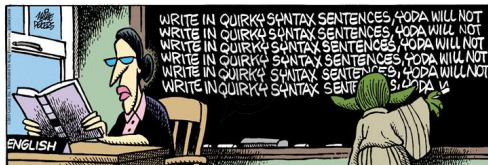
- Racket (Lisps in general) use prefix (polish) notation
- Tons of advantages:
 1. Easy to fit the grammar in your head (unlike C++)

Ewww Gross Syntax!



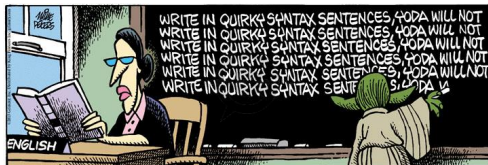
- Racket (Lisps in general) use prefix (polish) notation
- Tons of advantages:
 1. Easy to fit the grammar in your head (unlike C++)
 2. Racket is *homoiconic*

Ewww Gross Syntax!



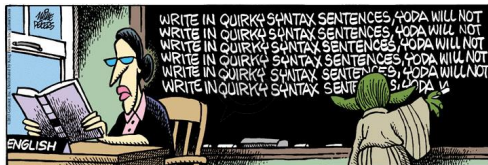
- Racket (Lisps in general) use prefix (polish) notation
- Tons of advantages:
 1. Easy to fit the grammar in your head (unlike C++)
 2. Racket is *homoiconic*
 3. Makes the language extensible

Ewww Gross Syntax!



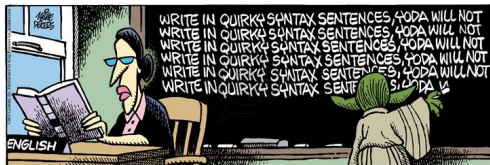
- Racket (Lisps in general) use prefix (polish) notation
- Tons of advantages:
 1. Easy to fit the grammar in your head (unlike C++)
 2. Racket is *homoiconic*
 3. Makes the language extensible
 4. Allows for structured editing

Ewww Gross Syntax!



- Racket (Lisps in general) use prefix (polish) notation
- Tons of advantages:
 1. Easy to fit the grammar in your head (unlike C++)
 2. Racket is *homoiconic*
 3. Makes the language extensible
 4. Allows for structured editing
- However one large disadvantage is that it is *different*

Ewww Gross Syntax!



- Racket (Lisps in general) use prefix (polish) notation
- Tons of advantages:
 1. Easy to fit the grammar in your head (unlike C++)
 2. Racket is *homoiconic*
 3. Makes the language extensible
 4. Allows for structured editing
- However one large disadvantage is that it is *different*
- You'll get used to it

Let's Write a Program!

We're going to write a program to make a rocket land, SpaceX style.



Let's Write a Program!

We're going to write a program to make a rocket land, SpaceX style.



- First, let's post an image of a rocket into Dr. Racket

Let's Write a Program!

We're going to write a program to make a rocket land, SpaceX style.



- First, let's post an image of a rocket into Dr. Racket



Let's Write a Program!

We're going to write a program to make a rocket land, SpaceX style.



- First, let's post an image of a rocket into Dr. Racket



Welcome to [DrRacket](#), version 7.2 [3m].

Language: *No language chosen*; memory limit: 128 MB.

DrRacket cannot process programs until you choose a programming language.

Either select the "Choose Language..." item in the "Language" menu, or get [guidance](#).




Messing With Images

Let's play around with our rocket:

- `(define rocket`  `)`

Messing With Images

Let's play around with our rocket:

- `(define rocket`  `)`
- Now let's create a function to calculate its size:

Messing With Images

Let's play around with our rocket:

- `(define rocket`  `)`

- Now let's create a function to calculate its size:

```
(define rocket-size  
  (* (image-width rocket) (image-height rocket)))
```

Messing With Images

Let's play around with our rocket:



- `(define rocket)`

- Now let's create a function to calculate its size:

```
(define rocket-size  
  (* (image-width rocket) (image-height rocket)))
```

- We can define circles: `(circle 20 "solid" "red")`
- Or rectangles: `(rectangle 30 20 "outline" "blue")`

Drawing a Scene

We'll start by drawing an empty scene.

- `(empty-scene 100 100)`

Drawing a Scene

We'll start by drawing an empty scene.

- `(empty-scene 100 100)`
- *Boring!* Let's add something to it:

Drawing a Scene

We'll start by drawing an empty scene.

- `(empty-scene 100 100)`
- *Boring!* Let's add something to it:

```
(place-image (circle 10 "solid" "green")  
             50 80  
             (empty-scene 100 100))
```

Drawing a Scene

We'll start by drawing an empty scene.

- `(empty-scene 100 100)`
- *Boring!* Let's add something to it:

```
(place-image (circle 10 "solid" "green")  
             50 80  
             (empty-scene 100 100))
```

- Here's what our new scene looks like:



Rocket Fun

Let's consider placing the rocket in the scene:

Rocket Fun

Let's consider placing the rocket in the scene:

```
(place-image rocket 50 20 (empty-scene 100 60))
```


Rocket Fun

Let's consider placing the rocket in the scene:

```
(place-image rocket 50 20 (empty-scene 100 60))
```



Rocket Fun

Let's consider placing the rocket in the scene:

```
(place-image rocket 50 20 (empty-scene 100 60))
```




- Let's open Dr. Racket and try moving the rocket up and down.

Rocket Fun

Let's consider placing the rocket in the scene:

```
(place-image rocket 50 20 (empty-scene 100 60))
```

- 
- Let's open Dr. Racket and try moving the rocket up and down.
- As you can see, higher numbers for y move it lower

Baby's First Animation

Here's a first attempt at a function that places the rocket at a height.

```
(define (picture-of-rocket height)
  (place-image 50 height (empty-scene 100 60)))
```

Baby's First Animation

Here's a first attempt at a function that places the rocket at a height.

```
(define (picture-of-rocket height)
  (place-image 50 height (empty-scene 100 60)))
```

-
- Now let's mess with the *animate* function

Baby's First Animation

Here's a first attempt at a function that places the rocket at a height.

```
(define (picture-of-rocket height)
  (place-image 50 height (empty-scene 100 60)))
```

-
- Now let's mess with the *animate* function
- (animate picture-of-rocket)

Baby's First Animation

Here's a first attempt at a function that places the rocket at a height.

```
(define (picture-of-rocket height)
  (place-image 50 height (empty-scene 100 60)))
```

-
- Now let's mess with the *animate* function
- (animate picture-of-rocket)
- Anything weird about this expression?

Baby's First Animation

Here's a first attempt at a function that places the rocket at a height.

```
(define (picture-of-rocket height)
  (place-image 50 height (empty-scene 100 60)))
```

-
- Now let's mess with the *animate* function
- (animate picture-of-rocket)
- Anything weird about this expression?
- This is our first example of using *higher-order* function, a function that takes another function as input or produces another function as output.

Baby's First Animation

Here's a first attempt at a function that places the rocket at a height.

```
(define (picture-of-rocket height)
  (place-image 50 height (empty-scene 100 60)))
```

-
- Now let's mess with the *animate* function
- (animate picture-of-rocket)
- Anything weird about this expression?
- This is our first example of using *higher-order* function, a function that takes another function as input or produces another function as output.
- One example of a famous higher order function is $\frac{d}{dx}$

Was Our Animation Good?

- It wasn't so nice that our rocket flew through the ground

Was Our Animation Good?

- It wasn't so nice that our rocket flew through the ground
- So let's fix that.

Was Our Animation Good?

- It wasn't so nice that our rocket flew through the ground
- So let's fix that.
- First, we need to introduce conditional expressions:

Was Our Animation Good?

- It wasn't so nice that our rocket flew through the ground
- So let's fix that.
- First, we need to introduce conditional expressions:

```
(define (sign x)
  (cond
    [(< x 0) -1]
    [(> x 0) 1]
    [else 0]))
```



Was Our Animation Good?

- It wasn't so nice that our rocket flew through the ground
- So let's fix that.
- First, we need to introduce conditional expressions:

```
(define (sign x)
  (cond
    [(< x 0) -1]
    [(> x 0) 1]
    [else 0]))
```

- ```
public int sign(int x) {
 if (x < 0)
 return -1
 else if (x > 0)
 return 1
 else
 return 0
}
```

## Fix the Landing

Let's try out a new version of the function:

## Fix the Landing

Let's try out a new version of the function:

```
(define (picture-of-rocket.v2 height)
 (cond
 [(<= height 60)
 (place-image rocket 50 height (empty-scene 100 60))]
 [else
 (place-image rocket 50 60 (empty-scene 100 60))]))
```





## Fix the Landing

Let's try out a new version of the function:

```
(define (picture-of-rocket.v2 height)
 (cond
 [(<= height 60)
 (place-image rocket 50 height (empty-scene 100 60))]
 [else
 (place-image rocket 50 60 (empty-scene 100 60))]))
```

- 
- Is there anything wrong with this code?

## Fix the Landing

Let's try out a new version of the function:

```
(define (picture-of-rocket.v2 height)
 (cond
 [(<= height 60)
 (place-image rocket 50 height (empty-scene 100 60))]
 [else
 (place-image rocket 50 60 (empty-scene 100 60))]))
```

- 
- Is there anything wrong with this code?
  1. Magic numbers

## Fix the Landing

Let's try out a new version of the function:

```
(define (picture-of-rocket.v2 height)
 (cond
 [(<= height 60)
 (place-image rocket 50 height (empty-scene 100 60))]
 [else
 (place-image rocket 50 60 (empty-scene 100 60))]))
```

- 
- Is there anything wrong with this code?
  1. Magic numbers
  2. Repeated empty-scene call

## Fix the Landing

Let's try out a new version of the function:

```
(define (picture-of-rocket.v2 height)
 (cond
 [(<= height 60)
 (place-image rocket 50 height (empty-scene 100 60))]
 [else
 (place-image rocket 50 60 (empty-scene 100 60))]))
```

- 
- Is there anything wrong with this code?
  1. Magic numbers
  2. Repeated empty-scene call
- What about the landing?

## Fix the Landing

Let's try out a new version of the function:

```
(define (picture-of-rocket.v2 height)
 (cond
 [(<= height 60)
 (place-image rocket 50 height (empty-scene 100 60))]
 [else
 (place-image rocket 50 60 (empty-scene 100 60))]))
```

- 
- Is there anything wrong with this code?
  1. Magic numbers
  2. Repeated empty-scene call
- What about the landing?
- Let's clean it up a bit.

## Fix the Landing (cont)

The landing should take into account the midpoint of the rocket and the canvas-height.

- No more magic numbers!

## Fix the Landing (cont)

The landing should take into account the midpoint of the rocket and the canvas-height.

- No more magic numbers!
- `(define canvas-height 60)`

## Fix the Landing (cont)

The landing should take into account the midpoint of the rocket and the canvas-height.

- No more magic numbers!
- (`define` canvas-height 60)
- Now let's get the center point:



## Fix the Landing (cont)

The landing should take into account the midpoint of the rocket and the canvas-height.

- No more magic numbers!
- `(define canvas-height 60)`
- Now let's get the center point:

```
(define rocket-center-to-top
 (- canvas-height (/ rocket-height 2)))
```



## Fix the Landing (cont)

The landing should take into account the midpoint of the rocket and the canvas-height.

- No more magic numbers!
- `(define canvas-height 60)`
- Now let's get the center point:

```
(define rocket-center-to-top
 (- canvas-height (/ rocket-height 2)))
```

- 
- Let's look at our new version.

## Good Now?

```
(define (picture-of-rocket.v3 height)
 (cond
 [(<= height rocket-center-to-top)
 (place-image rocket canvas-mid-width height canvas)]
 [else
 (place-image rocket canvas-mid-width rocket-center-to-top canvas)]
```

- Looks a lot better to me, in animation and code.

## Good Now?

```
(define (picture-of-rocket.v3 height)
 (cond
 [(<= height rocket-center-to-top)
 (place-image rocket canvas-mid-width height canvas)]
 [else
 (place-image rocket canvas-mid-width rocket-center-to-top canvas)]
```

- Looks a lot better to me, in animation and code.
- It's a bit weird that it lands on “nothing”, so let's add a rock bed and a blue background.

## Good Now?

```
(define (picture-of-rocket.v3 height)
 (cond
 [(<= height rocket-center-to-top)
 (place-image rocket canvas-mid-width height canvas)]
 [else
 (place-image rocket canvas-mid-width rocket-center-to-top canvas)]
```

- Looks a lot better to me, in animation and code.
- It's a bit weird that it lands on “nothing”, so let's add a rock bed and a blue background.

```
(define rock-bed
 (rectangle canvas-width 10 "solid" "grey"))
```

-

## Good Now?

```
(define (picture-of-rocket.v3 height)
 (cond
 [(<= height rocket-center-to-top)
 (place-image rocket canvas-mid-width height canvas)]
 [else
 (place-image rocket canvas-mid-width rocket-center-to-top canvas)]
```

- Looks a lot better to me, in animation and code.
- It's a bit weird that it lands on “nothing”, so let's add a rock bed and a blue background.

```
(define rock-bed
 (rectangle canvas-width 10 "solid" "grey"))
```

- 

```
(define blue-canvas
 (empty-scene canvas-width canvas-height "blue"))
```

-

## Rockbedding and Rolling

```
(define (picture-of-rocket.v4 height)
 (cond
 [(<= height rocket-center-to-top)
 (place-image rocket canvas-mid-width height
 (place-image rock-bed canvas-mid-width canvas-height blue-canvas))
]
 [else
 (place-image rocket canvas-mid-width rocket-center-to-top
 (place-image rock-bed canvas-mid-width canvas-height blue-canvas))
]
)
```

## Rockbedding and Rolling

```
(define (picture-of-rocket.v4 height)
 (cond
 [(<= height rocket-center-to-top)
 (place-image rocket canvas-mid-width height
 (place-image rock-bed canvas-mid-width canvas-height blue-canvas))
 [else
 (place-image rocket canvas-mid-width rocket-center-to-top
 (place-image rock-bed canvas-mid-width canvas-height blue-canvas))])
```





## But do We Understand Animate?

Let's look up what exactly it does in the online documentation.

- `https://docs.racket-lang.org/teachpack/2htdpuniverse.html`

## But do We Understand Animate?

Let's look up what exactly it does in the online documentation.

- `https://docs.racket-lang.org/teachpack/2htdpuniverse.html`
- So, `animate` applies the argument function with a *time* that is used to create an image.

## But do We Understand Animate?

Let's look up what exactly it does in the online documentation.

- `https://docs.racket-lang.org/teachpack/2htdpuniverse.html`
- So, `animate` applies the argument function with a *time* that is used to create an image.
- Does our code have a big problem then?

## But do We Understand Animate?

Let's look up what exactly it does in the online documentation.

- `https://docs.racket-lang.org/teachpack/2htdpuniverse.html`
- So, `animate` applies the argument function with a *time* that is used to create an image.
- Does our code have a big problem then?
- We are using time as distance!

## But do We Understand Animate?

Let's look up what exactly it does in the online documentation.

- `https://docs.racket-lang.org/teachpack/2htdpuniverse.html`
- So, `animate` applies the argument function with a *time* that is used to create an image.
- Does our code have a big problem then?
- We are using time as distance!
- We need one last version of the program.

# Last Fix

Fix:

1. (`define` velocity 3)

## Last Fix

Fix:

1. `(define velocity 3)`

2. `(define (distance time) (* time velocity))`

```
(define (picture-of-rocket.v5 time)
 (cond
 ((<= (distance time) rocket-center-to-top)
 (place-image rocket canvas-mid-width (distance time)
 (place-image rock-bed canvas-mid-width canvas-height blue-canvas)))
 (else
 (place-image rocket canvas-mid-width rocket-center-to-top
 (place-image rock-bed canvas-mid-width canvas-height blue-canvas))))
```

# Congratulations!

You're a functional programmer now! ...



# Congratulations!

You're a functional programmer now! ...

- Right?

# Congratulations!

You're a functional programmer now! ...

- Right?
- Of course not!

# Congratulations!

You're a functional programmer now! ...

- Right?
- Of course not!
- Many other courses might try to continue introducing features in this manner...

# Congratulations!

You're a functional programmer now! ...

- Right?
- Of course not!
- Many other courses might try to continue introducing features in this manner...
- But this lecture was messy and barely motivated the need for novel things like higher order functions, that I glossed over.

# Congratulations!

You're a functional programmer now! ...

- Right?
- Of course not!
- Many other courses might try to continue introducing features in this manner...
- But this lecture was messy and barely motivated the need for novel things like higher order functions, that I glossed over.
- So, next lecture we're going to *slow down* and take a systematic approach.

# A Glimpse of the Future

## 1. From Problem Analysis to Data Definitions

Identify the information that must be represented and how it is represented in the chosen programming language. Formulate data definitions and illustrate them with examples.

## 2. Signature, Purpose Statement, Header

State what kind of data the desired function consumes and produces. Formulate a concise answer to the question what the function computes. Define a stub that lives up to the signature.

## 3. Functional Examples

Work through examples that illustrate the function's purpose.

## 4. Function Template

Translate the data definitions into an outline of the function.

## 5. Function Definition

Fill in the gaps in the function template. Exploit the purpose statement and the examples.

## 6. Testing

Articulate the examples as tests and ensure that the function passes all. Doing so discovers mistakes. Tests also supplement examples in that they help others read and understand the definition when the need arises—and it will arise for any serious program.

Figure 1: The basic steps of a function design recipe

# A Glimpse of the Future

## 1. From Problem Analysis to Data Definitions

Identify the information that must be represented and how it is represented in the chosen programming language. Formulate data definitions and illustrate them with examples.

## 2. Signature, Purpose Statement, Header

State what kind of data the desired function consumes and produces. Formulate a concise answer to the question what the function computes. Define a stub that lives up to the signature.

## 3. Functional Examples

Work through examples that illustrate the function's purpose.

## 4. Function Template

Translate the data definitions into an outline of the function.

## 5. Function Definition

Fill in the gaps in the function template. Exploit the purpose statement and the examples.

## 6. Testing

Articulate the examples as tests and ensure that the function passes all. Doing so discovers mistakes. Tests also supplement examples in that they help others read and understand the definition when the need arises—and it will arise for any serious program.

Figure 1: The basic steps of a function design recipe

- We'll start with a slow paced introduction to the basic building blocks of programs.

# A Glimpse of the Future

## 1. From Problem Analysis to Data Definitions

Identify the information that must be represented and how it is represented in the chosen programming language. Formulate data definitions and illustrate them with examples.

## 2. Signature, Purpose Statement, Header

State what kind of data the desired function consumes and produces. Formulate a concise answer to the question what the function computes. Define a stub that lives up to the signature.

## 3. Functional Examples

Work through examples that illustrate the function's purpose.

## 4. Function Template

Translate the data definitions into an outline of the function.

## 5. Function Definition

Fill in the gaps in the function template. Exploit the purpose statement and the examples.

## 6. Testing

Articulate the examples as tests and ensure that the function passes all. Doing so discovers mistakes. Tests also supplement examples in that they help others read and understand the definition when the need arises—and it will arise for any serious program.

Figure 1: The basic steps of a function design recipe

- We'll start with a slow paced introduction to the basic building blocks of programs.
- Then we'll move onto how to create and call functions.



# A Glimpse of the Future

## 1. From Problem Analysis to Data Definitions

Identify the information that must be represented and how it is represented in the chosen programming language. Formulate data definitions and illustrate them with examples.

## 2. Signature, Purpose Statement, Header

State what kind of data the desired function consumes and produces. Formulate a concise answer to the question what the function computes. Define a stub that lives up to the signature.

## 3. Functional Examples

Work through examples that illustrate the function's purpose.

## 4. Function Template

Translate the data definitions into an outline of the function.

## 5. Function Definition

Fill in the gaps in the function template. Exploit the purpose statement and the examples.

## 6. Testing

Articulate the examples as tests and ensure that the function passes all. Doing so discovers mistakes. Tests also supplement examples in that they help others read and understand the definition when the need arises—and it will arise for any serious program.

Figure 1: The basic steps of a function design recipe

- We'll start with a slow paced introduction to the basic building blocks of programs.
- Then we'll move onto how to create and call functions.
- Then we'll discuss the two fundamental kinds of programs, batch programs (scripts) and big bang programs (event driven).

# A Glimpse of the Future

## 1. From Problem Analysis to Data Definitions

Identify the information that must be represented and how it is represented in the chosen programming language. Formulate data definitions and illustrate them with examples.

## 2. Signature, Purpose Statement, Header

State what kind of data the desired function consumes and produces. Formulate a concise answer to the question what the function computes. Define a stub that lives up to the signature.

## 3. Functional Examples

Work through examples that illustrate the function's purpose.

## 4. Function Template

Translate the data definitions into an outline of the function.

## 5. Function Definition

Fill in the gaps in the function template. Exploit the purpose statement and the examples.

## 6. Testing

Articulate the examples as tests and ensure that the function passes all. Doing so discovers mistakes. Tests also supplement examples in that they help others read and understand the definition when the need arises—and it will arise for any serious program.

Figure 1: The basic steps of a function design recipe

- We'll start with a slow paced introduction to the basic building blocks of programs.
- Then we'll move onto how to create and call functions.
- Then we'll discuss the two fundamental kinds of programs, batch programs (scripts) and big bang programs (event driven).
- Then finally, how to *design* these kinds of programs.