Contracts

Contracts tell us how to use a function. For example: `num-min :: (a :: Number, b :: Number) -> Number` tells us that the name of the function is `num-min`, it takes two inputs (both Numbers), and it evaluates to a `Number`. From the contract, we know `num-min(4, 6)` will evaluate to a `Number`. **Use the blank line under each contract for notes or sample code for that function!**

Name		Domain		Range
num-sqr	::	Number	->	Number
num-sqr(9)				
num-sqrt	::	Number	->	Number
num-sqrt(25)				
triangle	::	Number, String, String	->	Image
triangle(80, "solid", "darkgreen")				
circle	::		->	Image
star	::		->	
square	::		->	
rectangle	::		->	
text	::		->	
ellipse	::		->	

Contracts

Contracts tell us how to use a function. For example: `num-min :: (a :: Number, b :: Number) -> Number` tells us that the name of the function is `num-min`, it takes two inputs (both Numbers), and it evaluates to a `Number`. From the contract, we know `num-min(4, 6)` will evaluate to a `Number`. **Use the blank line under each contract for notes or sample code for that function!**

Name	Domain	Range
`;regular-polygon`	`::`	`->`
`#rhombus`	`::`	`->`
`#right-triangle`	`::`	`->`
`#isosceles-triangle`	`::`	`->`
`#radial-star`	`::`	`->`
`#star-polygon`	`::`	`->`
`overlay`	`::` `Image, Image`	`->` `Image`
`overlay(star(30, "solid", "gold"),circle(30, "solid", "blue"))`		
`beside`	`::` `Image, Image`	`->` `Image`
`beside(star(50, "solid", "orange"),circle(50, "solid", "green"))`		
`above`	`::` `Image, Image`	`->` `Image`
`above(triangle(30, "solid", "red"),square(30, "solid", "blue"))`		

Contracts

Contracts tell us how to use a function. For example: `num-min :: (a :: Number, b :: Number) -> Number` tells us that the name of the function is `num-min`, it takes two inputs (both Numbers), and it evaluates to a `Number`. From the contract, we know `num-min(4, 6)` will evaluate to a `Number`. **Use the blank line under each contract for notes or sample code for that function!**

Name		Domain		Range
put-image	::	Image, Number, Number, Image	->	Image
put-image(star(30, "solid", "red"), 50, 150, rectangle(300, 200, "outline", "black"))				
rotate	::	Number, Image	->	Image
rotate(35, rectangle(30, 80, "solid", "purple"))				
scale	::	Number, Image	->	Image
scale(0.8, triangle(30, "solid", "red"))				
string-repeat	::	String, Number	->	String
string-repeat("cheetah ", 5)				
string-contains	::	String, String	->	Boolean
string-contains("rockstar", "star")				
num-min	::	Number, Number	->	Number
num-min(80, 20)				
num-max	::	Number, Number	->	Number
num-max(80, 20)				
count	::	Table, String	->	Table
count(animals-table, "species")				
mean	::	Table, String	->	Number
mean(animals-table, "age")				

Contracts

Contracts tell us how to use a function. For example: `num-min :: (a :: Number, b :: Number) -> Number` tells us that the name of the function is `num-min`, it takes two inputs (both Numbers), and it evaluates to a `Number`. From the contract, we know `num-min(4, 6)` will evaluate to a `Number`. **Use the blank line under each contract for notes or sample code for that function!**

Name		Domain		Range
median	::	Table, String	->	Number
median(animals-table, "age")				
modes	::	Table, String	->	List<Number>
modes(animals-table, "age")				
bar-chart	::	Table, String	->	Image
bar-chart(animals-table, "legs")				
pie-chart	::	Table, String	->	Image
pie-chart(animals-table, "species")				
histogram	::	(t :: Table, column :: String, bin-width :: Number)	->	Image
histogram(animals-table, "age", 2)				
box-plot	::	Table, String	->	Image
box-plot(animals-table, "age")				
modified-box-plot	::	Table, String)	->	Image
modified-box-plot(animals-table, "age")				
scatter-plot	::	(t :: Table, labels :: String, xs :: String, ys :: String)	->	Image
scatter-plot(animals-table, "species", "pounds", "weeks")				
image-scatter-plot	::	(t :: Table, xs :: String, ys :: String, f :: (Row -> Image))	->	Image
image-scatter-plot(animals-table, "pounds", "weeks", animal-img)				

Contracts

Contracts tell us how to use a function. For example: `num-min :: (a :: Number, b :: Number) -> Number` tells us that the name of the function is `num-min`, it takes two inputs (both Numbers), and it evaluates to a `Number`. From the contract, we know `num-min(4, 6)` will evaluate to a `Number`. **Use the blank line under each contract for notes or sample code for that function!**

Name		Domain		Range
r-value	::	(t :: Table, xs :: String, ys :: String)	->	Number
r-value(animals-table,"pounds", "weeks")				
lr-plot	::	(t :: Table, labels :: String, xs :: String, ys :: String)	->	Image
lr-plot(animals-table, "species", "pounds", "weeks")				
random-rows	::	(t :: Table, num-rows :: Number)	->	Table
random-rows(animals-table, 5)				
<Table>.row-n	::	Number	->	Row
animals-table.row-n(5)				
<Table>.order-by	::	(col :: String, increasing :: Boolean)	->	Table
animals-table.order-by("species", true)				
<Table>.filter	::	(test :: (Row -> Boolean))	->	Table
animal-table.filter(is-cat)				
<Table>.build-column	::	(col :: String, builder :: (Row -> Any))	->	Table
animals-table.build-column("sticker", label)				
bar-chart-summarized	::	(t :: Table, labels :: String, values :: String)	->	Image
bar-chart-summarized(animals-table, "species", "pounds")				
pie-chart-summarized	::	(t :: Table, labels :: String, values :: String)	->	Image
pie-chart-summarized(animals-table, "age", "pounds")				

Contracts

Contracts tell us how to use a function. For example: `num-min :: (a :: Number, b :: Number) -> Number` tells us that the name of the function is `num-min`, it takes two inputs (both Numbers), and it evaluates to a `Number`. From the contract, we know `num-min(4, 6)` will evaluate to a `Number`. **Use the blank line under each contract for notes or sample code for that function!**

Name	Domain	Range
::		->
::		->
::		->
::		->
::		->
::		->
::		->
::		->
::		->