

Analysis of Environmental Data Lab

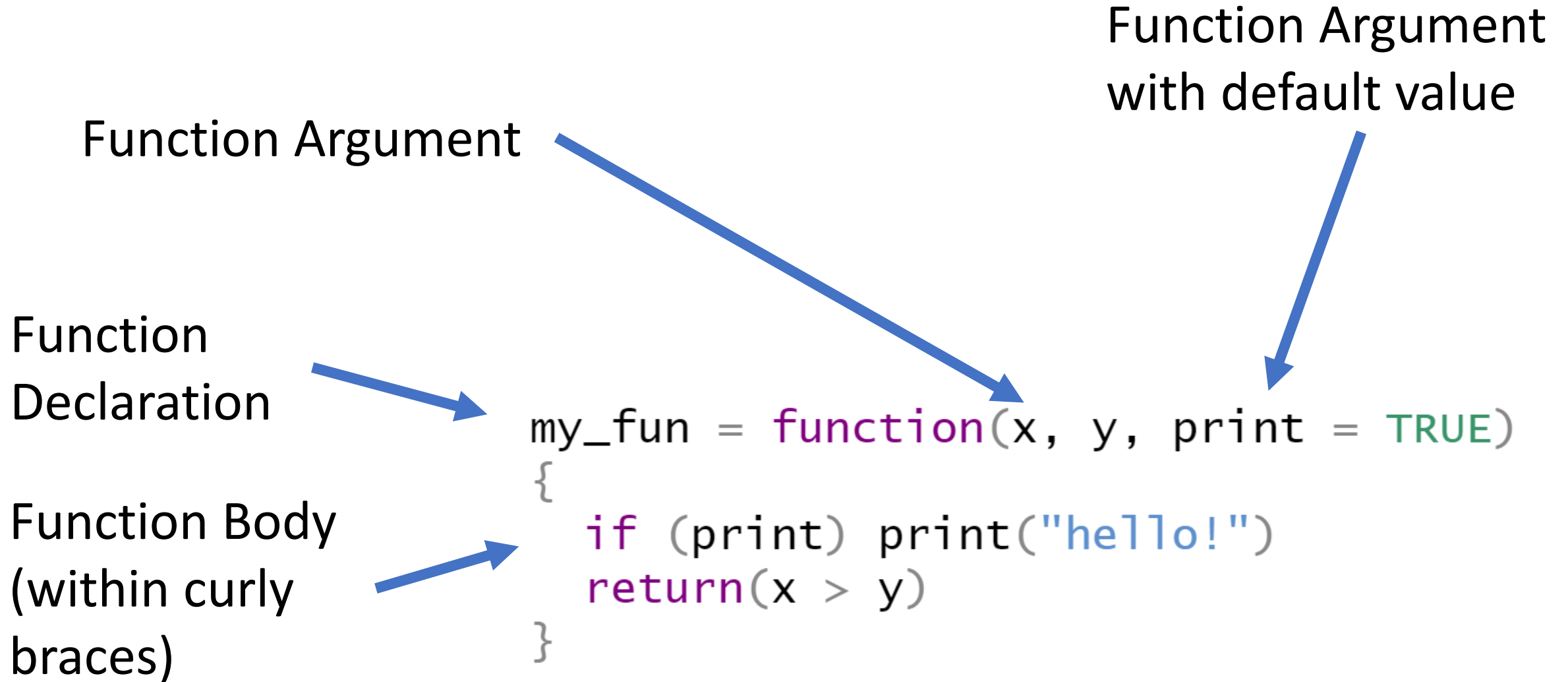
Functions

Eco 602 – University of Massachusetts, Amherst – Fall 2021
Michael France Nelson

Important Concepts

- Function declaration
- Function arguments (also known as parameters)
- Function body
- Function scope
- Return statements
- Function-building strategies

Anatomy of a Function



What does a function do?

1. Optionally accepts some input
2. Optionally performs some task
3. Optionally returns a value, perhaps silently, and exits

```
hello = function()  
{  
    print("hello!")  
}
```

What does this function do?

Declaration

```
hello = function()  
{  
  print("hello!")  
}
```

Invocation and Output

```
> hello()  
[1] "hello!"
```

What does this function do?

Declaration

```
hello = function()  
{  
    print("hello!")  
}
```

Steps

- Accepts no input
- Prints a message
- Does not return a value

What does this function do?

Declaration

```
my_fun = function(x, y, print = TRUE)
{
  if (print) print("hello!")
  return(x > y)
}
```

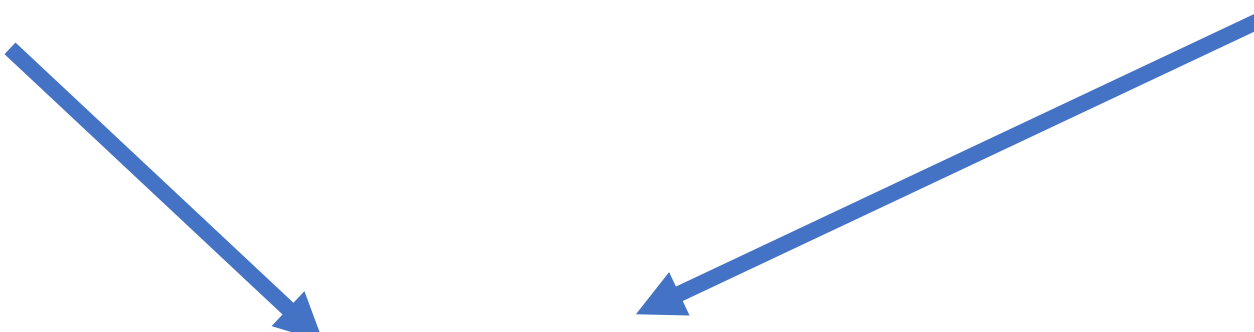
Steps

- Requires you to specify values for x and y.
- You can optionally specify a value for 'print'
- If print is TRUE, it prints a message.
- Returns TRUE if x is greater than y, otherwise returns false

Arguments

Function Argument
without default
value

Function Argument
with default value

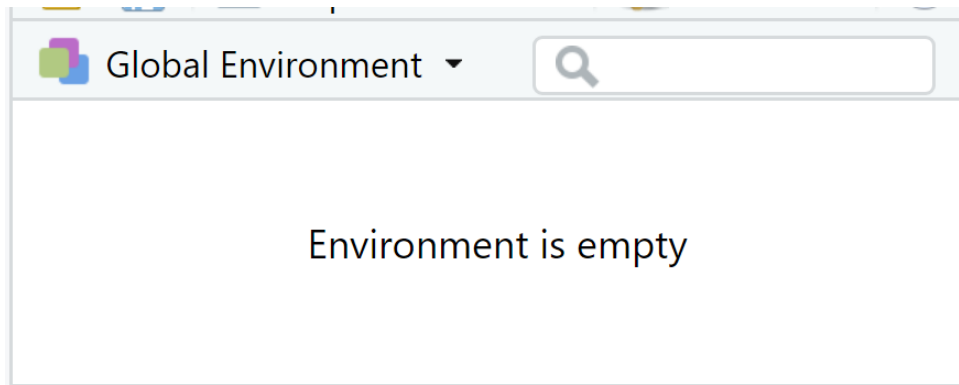


```
my_fun = function(x, y, print = TRUE)
{
  if (print) print("hello!")
  return(x > y)
}
```


Arguments and Scope

- Inside the function body, your function can see:
 - Everything within the global environment.
 - The values of any arguments you have supplied.
 - Any temporary variables you create within the function body.
- After the function exits, temporary variables and argument values are not visible in the global environment.

Scope: What can this function see?



```
prnt_msg = function(msg)
{
    print(msg)
}
```

Scope: What can this function see?

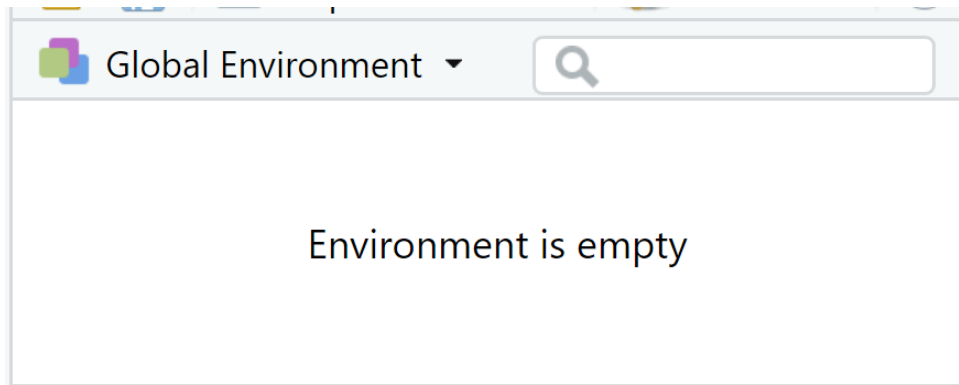


A screenshot of a Python environment's Global Environment window. The window title is "Global Environment" with a search bar. Below the title is a section labeled "values". A table shows a variable named "x" with a value of "2".

values	
x	2


```
print_msg = function(msg)
{
    print(msg)
}
```

Scope: Will this function run?



```
print_msg = function(msg)
{
  print(x)
}
```

Scope: Will this function run?



Global Environment	
values	
x	2

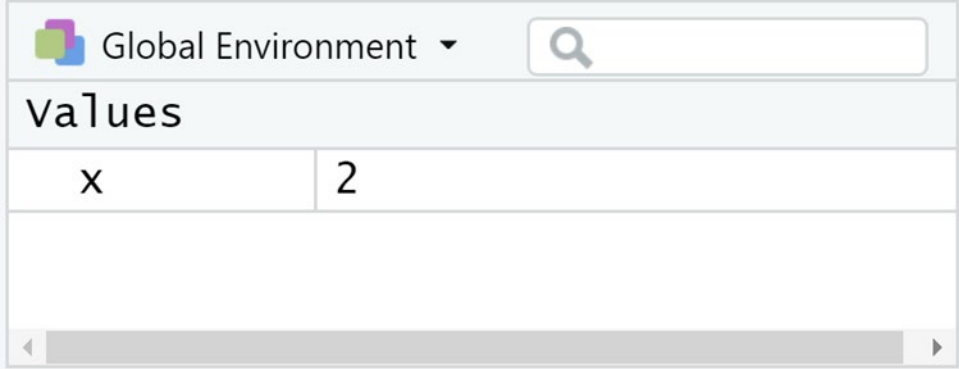
```
print_msg = function(msg)
{
  print(x)
}
```

Scope: Function and Global Environments

```
> x = 2
> prnt_msg = function(msg)
+ {
+   print(msg)
+   print(x)
+ }
> prnt_msg("abc")
[1] "abc"
[1] 2
> |
```

- The function can see both 'msg' and 'x'.
- The global environment only sees 'x'

Before



Global Environment

values	
x	2

After



Global Environment

values	
x	2

Strategies

- Work on your function code outside of the function.
 - When your code accomplishes what you want, you can ‘package’ it into a function.
- Use distinctive argument names.
 - It’s easy to have name collisions if you use argument names like ‘min’, ‘mean’, etc.
- Your function shouldn’t reference global variables.
 - You can test this by running your function in a clean R environment.
- Let’s practice!