Programming: Intro to Functions

Stat 133 with Gaston Sanchez

R Expressions

Introduction

Before describing some of the common programming structures in R, we need to talk about a basic concept called **Expressions**.

You’ve been using simple expressions so far, but we need to introduce the notion of a compound expression.

Simple Expressions

```
a <- "hi"

print(2 + 2)

mean(1:10)
```

Simple Expressions

```
a <- "hi"; print(2 + 2); mean(1:10)
```

Simple Expressions

```
a <- "hi"; print(2 + 2); mean(1:10)
```

Simple expressions, separated by semicolons, written in a single line of text

Although this is a perfectly valid piece of code, we don’t recommend this format because it’s hard to inspect visually.

Simple Expressions

```
a <- "hello"

print(2 + 2)

mean(1:10)
```

Compound Expressions

```
{
    a <- "hello"
    print(2 + 2)
    mean(1:10)
}
```

R will treat this as one "unit" or "block" of code

Compound Expressions

```
{

    a <- "hello"

    print(2 + 2)

    mean(1:10)

}
```

Although this is a perfectly valid piece of code, we never write an R expression like this (in and of itself)

So, when do we use

`{...}`

compound expressions?

Use of compund expressions

We use compound expressions (i.e. single expressions wrapped within braces) in programming structures like:

- Functions
- Conditionals (if-else)
- Loops (for, while)

Parenthesis, Brackets, and Braces

( )	functions	mean(1:10)
[ ]	objects	vec[3] mat[2,4]

{ }	compound expressions	{
		  a <- 3
		  b <- a^2
		}

Every expression has a value!

What happens when R executes this code?

```
{
   a <- "hi"
   print(2 + 2)
   mean(1:10)
}
```

What is the value of `x`?

```r
x <-  {
    a <- "hi"
    print(2 + 2)
    mean(1:10)
}
```

Repeat this mantra

Every expression in R has a value: the value of the last statement that is evaluated

Every expression in R has a value: the value of the last statement that is evaluated

Every expression in R has a value: the value of the last statement that is evaluated

Functions

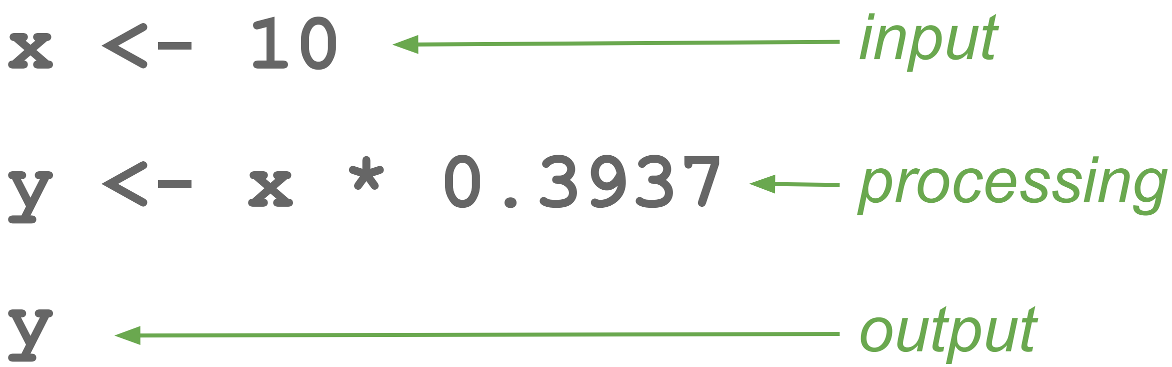
Centimeters to inches

```
# 1 cm = 0.3937 in

 x <- 10
 y <- x * 0.3937
 y
```

Centimeters to inches

```
# 1 cm = 0.3937 in
```



```
x <- 10

y <- x * 0.3937

y
```

input

processing

output

Centimeters to inches

```
# 1 cm = 0.3937 in
 x <- 10
{
 y <- x * 0.3937
 y
}
```

*Wrap the **body** of the function within an R expression*
(i.e. within braces)

Centimeters to inches

```
# 1 cm = 0.3937 in
function(x)
{
  y <- x * 0.3937
  y
}
```

Declare it as a function, and specify the argument(s)

Centimeters to inches

```
# 1 cm = 0.3937 in
cm2in <- function(x)
{
  y <- x * 0.3937
  y
}
```

Assign it to an object (give it a name)

Centimeters to inches

```
# 1 cm = 0.3937 in
cm2in <- function(x) {
  y <- x * 0.3937
  return(y)
}
```

Reformat for readibility purposes

Centimeters to inches

```
# 1 cm = 0.3937 in
cm2in <- function(x) {
  y <- x * 0.3937
  return(y)
}
cm2in(5)
```

test it