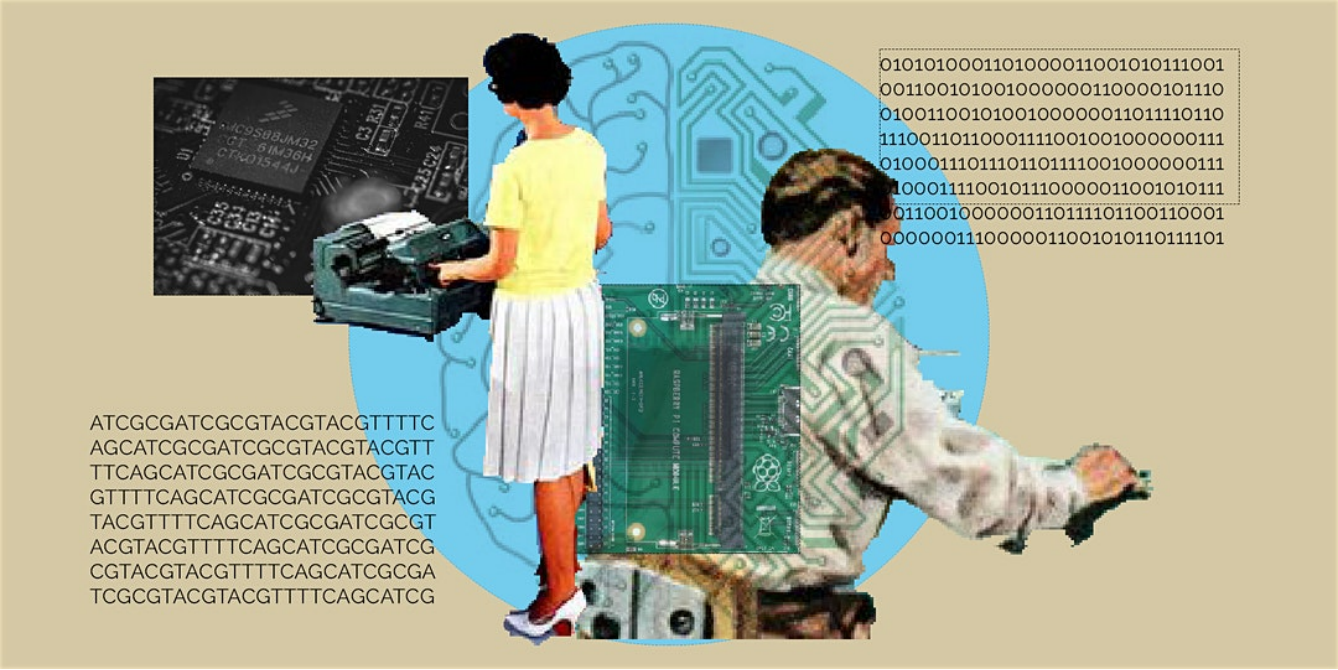
Genspace

Mining big data and demystifying the genome: learn R

April 18-19, 2020



My background

I got a bachelor of arts in Biology

I got a Ph.D. in Cell and Molecular Biology

I don't have a formal computer science background

I had one bioinformatics course in grad school (we learned Ruby)

I taught myself R

Now, I'm a postdoc at NYU School of Medicine

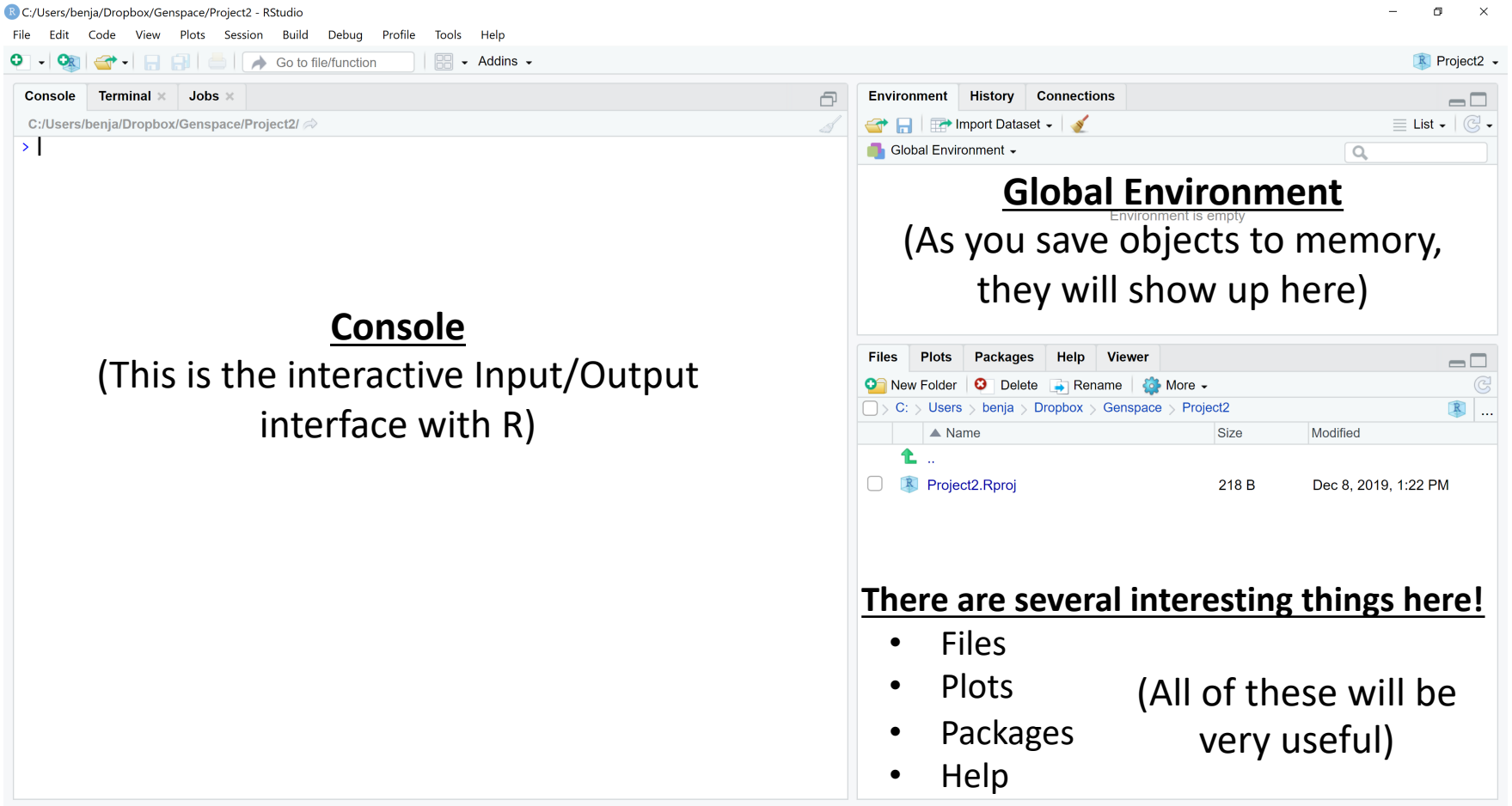
Now, I use R everyday to analyze RNA sequencing data

We're trying to understand why some ovarian cancer cells can become resistant to chemotherapeutic drugs

Computational Biology/Bioinformatics is an interesting field because a lot of us are largely self taught (this is my impression at least)

RStudio should look something like this:

What are these different panels?



Console

(This is the interactive Input/Output interface with R)

Global Environment

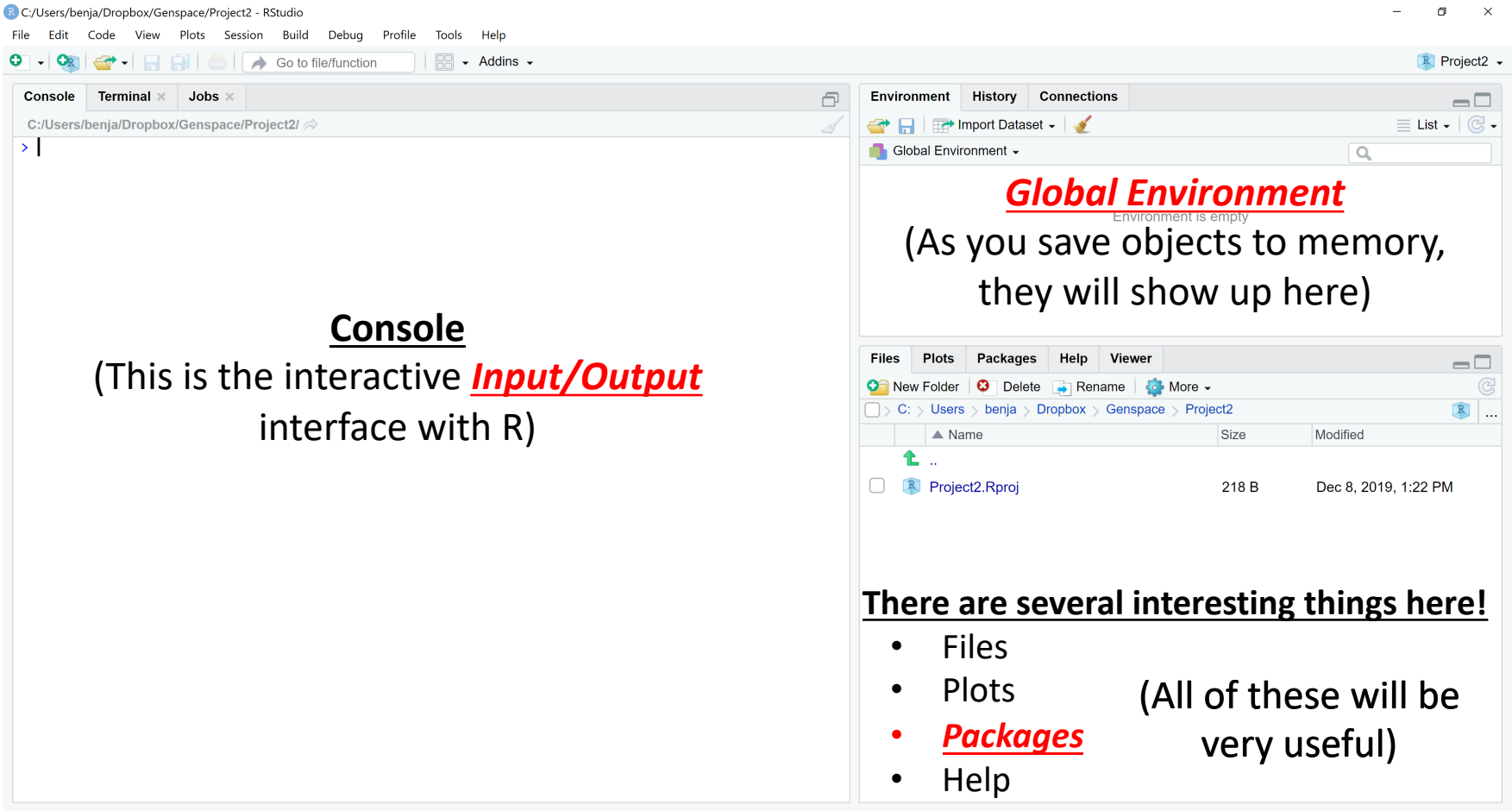
(As you save objects to memory, they will show up here)

There are several interesting things here!

- Files
- Plots
- Packages
- Help

(All of these will be very useful)

Jargon! Oh lord

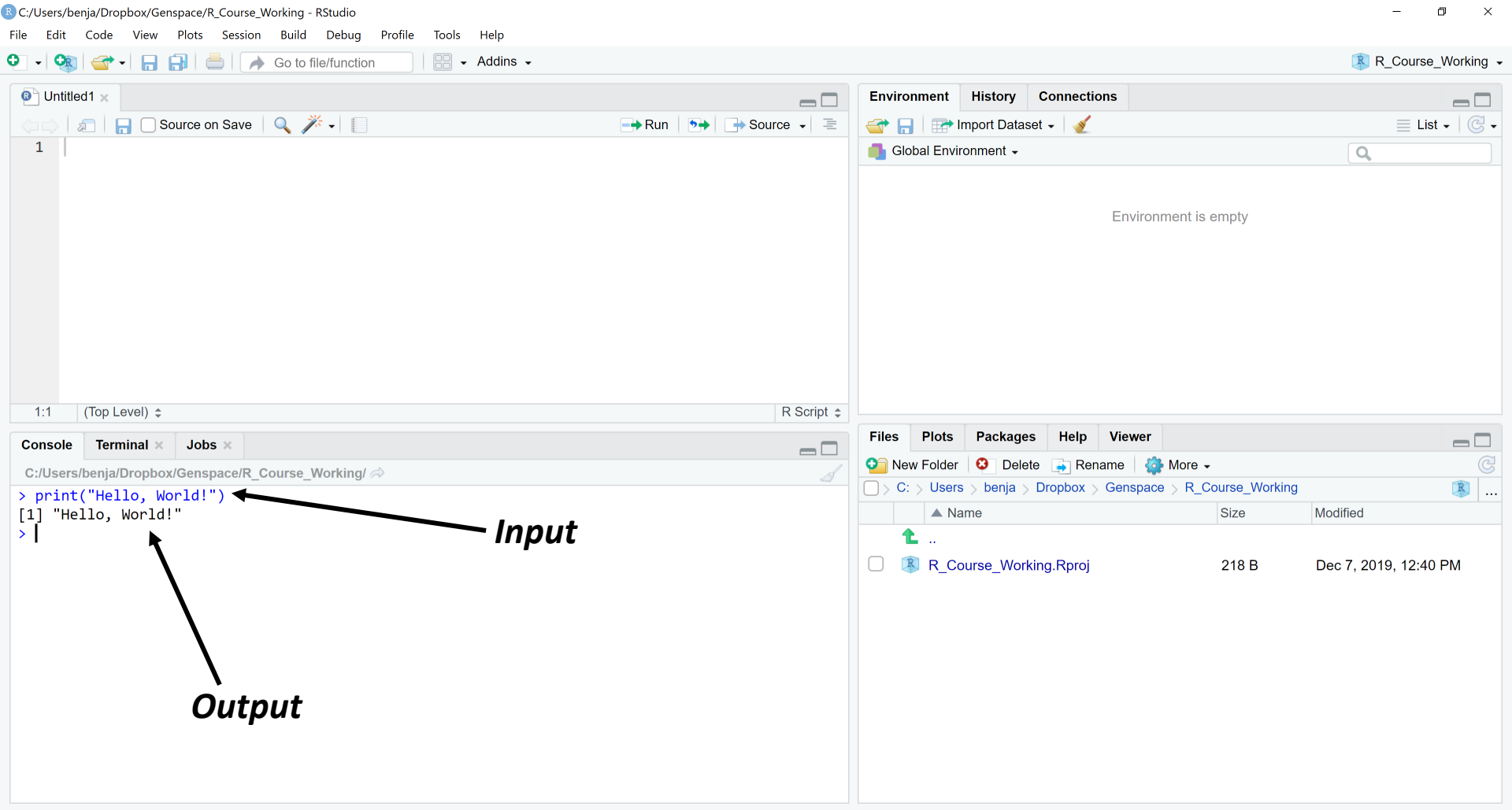


- There is a specific vocabulary used by computer scientists
- Stop me when I use jargon! Let me explain!
- It is helpful to learn some of this jargon so we can communicate with others

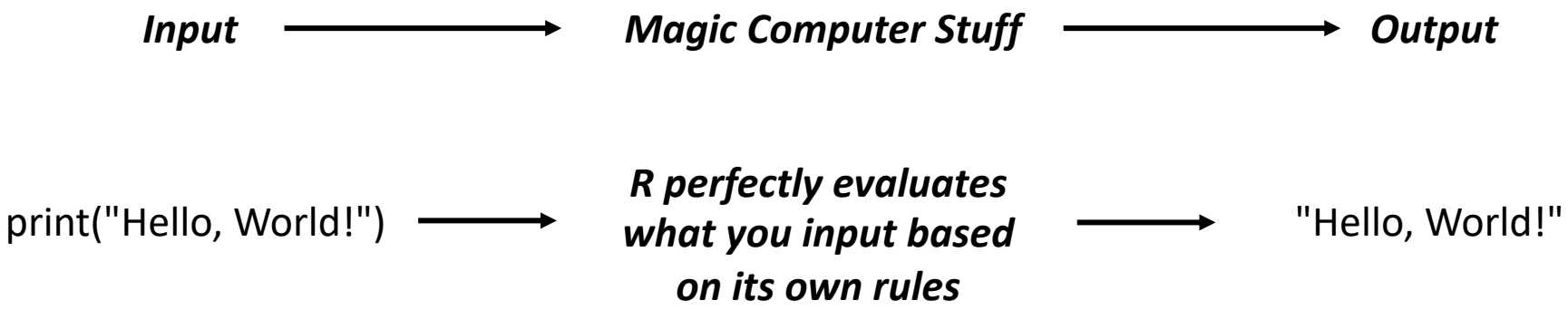
Input/Output and the Console Window

- Back to RStudio

Input/Output and the Console Window

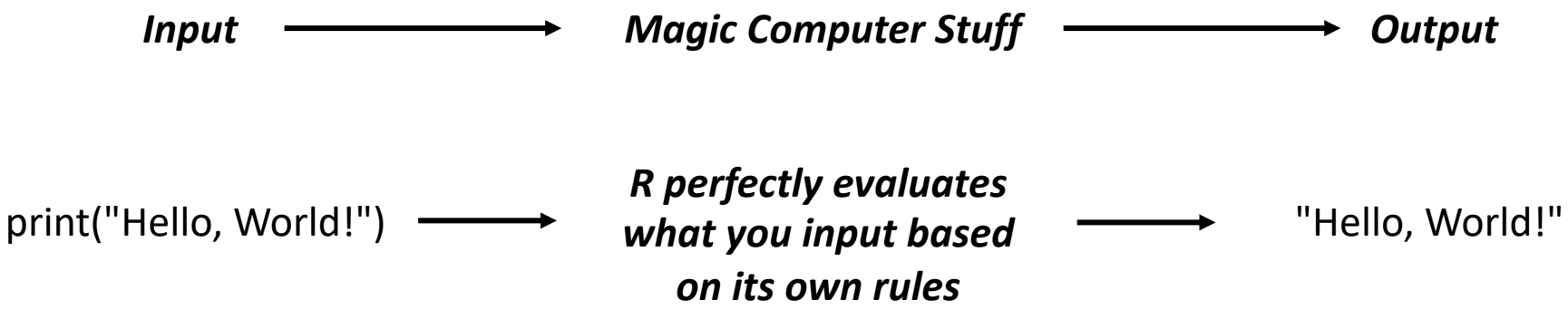


What is happening here?



The rules that R (or any coding language, for that matter) uses to "read" the code you have given it, is called syntax

What is happening here?



The rules that R (or any coding language, for that matter) uses to "read" the **code** you have given it, is called **syntax**

code is like a recipe that the computer reads.

Talk like a computer scientist! Jargon interlude:

The rules that R (or any coding language, for that matter) uses to "read" the **code** you have given it, is called **syntax**

code is like a recipe that you (the user) have written. The computer reads this recipe. It follows each step in the code until it reaches the end

code is synonomous with **script**

You could tell your boss/professor/teacher, "Hey, look at the R code that I wrote! This script lets me look for mutations in the genomes of cancer cells"

If you want R to do what you want, you need to know R's syntax!

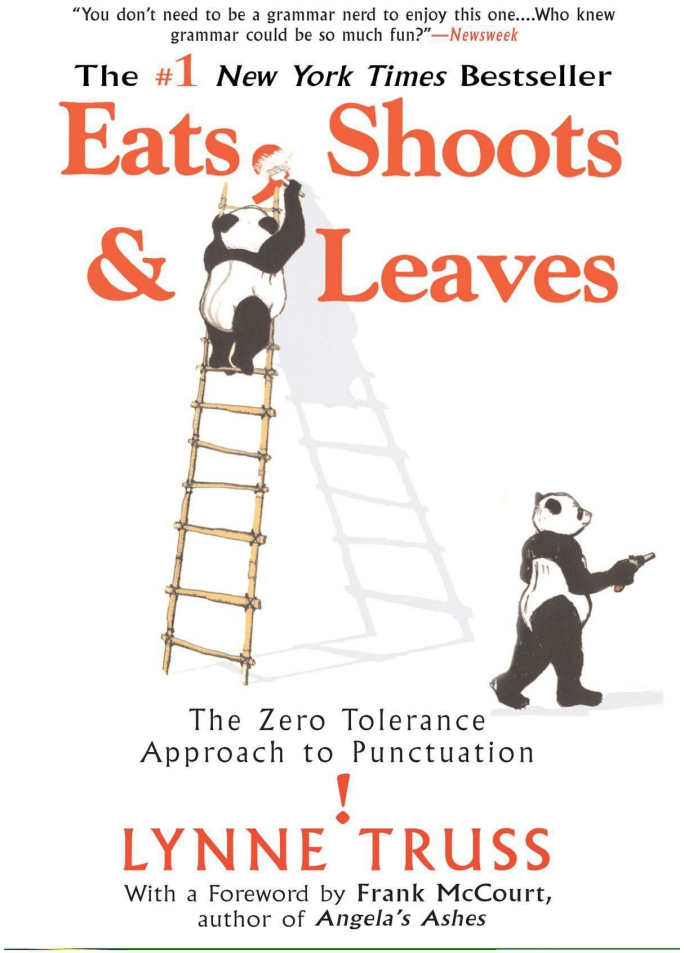
- Syntax in computer science has the same meaning as syntax in our everyday life:
- What is syntax?

- How do you "decode" the phrases:

  "Eats, shoots & leaves"

  versus

  "Eats shoots & leaves"



Title: Syntax is important
Subtitle: Computers are not humans

- Back to RStudio!
- In the **console** (bottom left), try typing the following lines of code

First,

```
print("Hello, World!")                {press enter}
```

Then,

```
 print "Hello, World!"                {press enter}
```

Then,

```
"print Hello, World!"                 {press enter}
```

Then,

```
print(‘Hello, World!’)                {press enter}
```

Then,

```
print(Hello, World!)                  {press enter}
```

Last,

```
HelloWorld = "Hello, World!"          {press enter}
print(HelloWorld)                     {press enter}
```

What happened? [Report back to the class!]

print("Hello, World!")	`[1] "Hello, World!"`
print "Hello, World!"	`Error: unexpected string constant in "print "Hello, World!""`
"print Hello, World!"	`[1] "print Hello, World!"`
print(‘Hello, World!’)	`[1] "Hello, World!"`
print(Hello, World!)	`Error: unexpected '!' in "print(Hello, World!"`
HelloWorld = "Hello, World!" print(HelloWorld)	`[1] "Hello, World!"`

Talk to your partner(s): Explain this behavior (2 minutes)

```
print("Hello, World!")
```
```
[1] "Hello, World!"
```

```
print "Hello, World!"
```
```
Error: unexpected string constant in "print "Hello, World!""
```

```
"print Hello, World!"
```
```
[1] "print Hello, World!"
```

```
print(‘Hello, World!’)
```
```
[1] "Hello, World!"
```

```
print(Hello, World!)
```
```
Error: unexpected '!' in "print(Hello, World!"
```

```
HelloWorld = "Hello, World!"
print(HelloWorld)
```
```
[1] "Hello, World!"
```

There are different types of ***objects*** in R

- When you write code, everything you write is an ***object***
- Every type of ***object*** is special
- The syntax of your code tells R what type of ***object*** it is
- R treats different ***objects*** in different ways

Based on your brainstorming with your partner(s),

To make this code work correctly, what are the important things R is looking for? (Type into Group Chat)

In this example, R knows what to do because of

- Quotation marks " " or ‘ ’
- Parentheses ( )
- Equals signs =

How does R interpret each of these?

You got it!

"Hello, World!"

Here, the quotation marks are telling R that the stuff inside is a ***String***

print(___)

Here, the parentheses are telling R that print is a ***Function***

HelloWorld

The lack of quotation marks tells R that this is a ***Variable***

HelloWorld = "Hello, World!"

The equals sign is telling R to set this ***variable*** to equal this ***string***. Cool thing here, this ***variable*** is stored in the computer's memory. You can access this ***string*** again by telling R to retrieve that ***variable***

*** this ***variable*** is now stored in R's ***Global Environment***. You can now see this in the "Global Environment" panel in the upper right of Rstudio

R is flexible.
There are many ways to do something right.

```
print("Hello, World!")
```
→
```
[1] "Hello, World!"
```

```
HelloWorld = "Hello, World!"
print(HelloWorld)
```
→
```
[1] "Hello, World!"
```

But syntax is important, and there are many ways something can go wrong

```
print "Hello, World!"
```
→
```
Error: unexpected string constant in "print "Hello, World!""
```

```
"print Hello, World!"
```
→
```
[1] "print Hello, World!"
```

```
print(Hello, World!)
```
→
```
Error: unexpected '!' in "print(Hello, World!"
```

When you code, most of your time will be spent ***debugging***

OMG, my code has a ***Bug***:

```
print "Hello, World!" → Error: unexpected string constant in "print "Hello, World!""
```

You run some code. And it gives an error!

```
"print Hello, World!" → [1] "print Hello, World!"
```

You run some code. And the output isn't what you want!

When you code, most of your time will be spent ***debugging***

```
print "Hello, World!" → Error: unexpected string constant in "print "Hello, World!""
```

OR

```
"print Hello, World!" → [1] "print Hello, World!"
```

This is ***debugging***

- The process of figuring out why your code isn't doing what you think it should
- And changing it so that it does

OMG, my code has a ***bug***:

- Am I stupid?

  Lol, No. The vast majority of everyone's time is spent debugging

```
print("Hello, World!") ⟶ [1] "Hello, World!"
```

Some important types of objects

Strings

Examples:

"cat"
"female"
"Homo sapiens"

Numbers

Examples:

0
1
0.25
0.33333333333
1000
10^{32}
10^{-18}
-3

Character and Numeric Classes

- You can ask R what kind of object something is

  (and you will often want to do this as your code gets more complicated)
- Back to RStudio
- Enter some (or all) of the following into the console of Rstudio:

```
class("cat")
class("print")
class(print)
class(1)
class(0.3)
class(-50)
class("1")
```

class() is a ***function***:

- A ***function*** always has parentheses
- The stuff inside its parentheses are called ***arguments***
- A ***function*** is this little black box. It grabs the ***arguments*** that you give it, it does something behind the scenes, and gives you something else
- We'll talk more about this, so don't worry about it now
- class() is looking at the object you give it, determining what kind of an object it is, and then it tells you what kind object it is

Character and Numeric Classes

- You can ask R what kind of object something is

  (and you will often want to do this as your code gets more complicated)
- Back to RStudio
- Enter some (or all) of the following into the console of Rstudio:

```
class("cat")
class("print")
class(print)
class(1)
class(0.3)
class(-50)
class("1")
```

What outputs did you get?

```
[1] "numeric"

[1] "function"

[1] "character"
```

Anything unexpected?

```
> class("1")
[1] "character"
```

What can you do with Numeric and Character Classes?

- Back to RStudio
- Try typing some of the following into the console of Rstudio
  (the numbers and words are arbitrary, use your own examples too)

```
1 + 1
1 - 1
2 / 4
2 / 3
2 * 0.5
2^2
4^(1/2)
```

```
"cat" + "dog"
"cat" * 3
"cat" / "dog"
"1" + "2"
"300" / "10"
"5" + 3
```

- What behavior did you observe?
- How do you explain this?

Character and Numeric Classes

- You can do math on objects that are Numeric
- You can't really do math on objects that are Character
- If you tell R to work with `"1"`, it will make no assumptions. It will not know it is a number.
- 1 ≠ "1"

Booleans: Another cool class of object!

- Does anyone know what a ***Boolean*** is?

TRUE or FALSE

(These are the only two options!)

- RStudio – try this code (and/or swap in your own objects):

```
1 == 1
1 == 1.0
1 == 1.1
"cat" == "cat"
"cat" == "CAT"
```

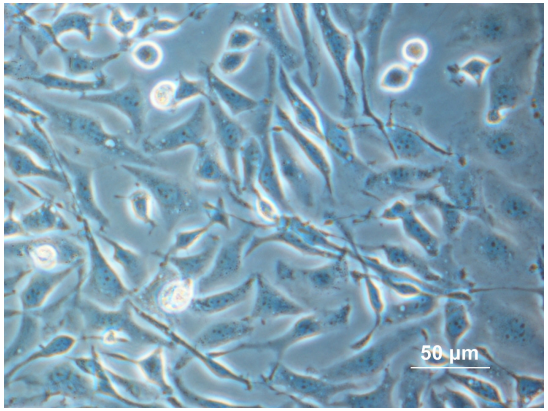
What do you think `==` does?

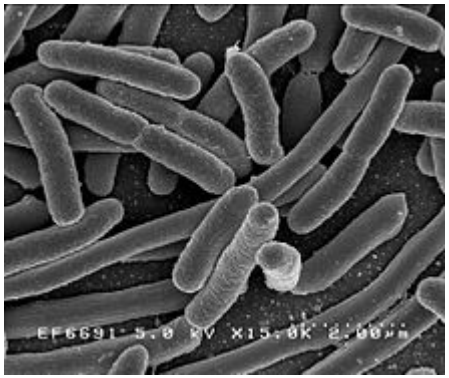
`==` is what is called a ***logical operator***
(There are other logical operators too)

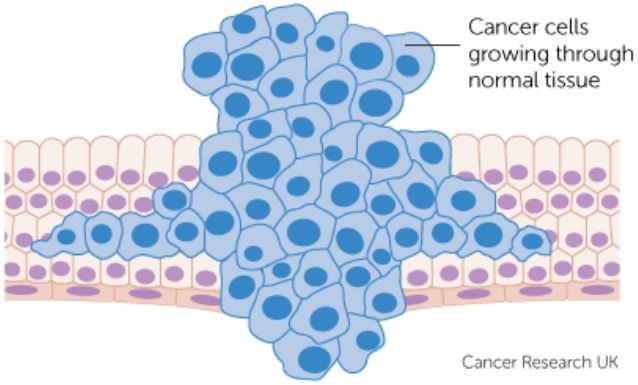
`==` is a way to ask if the first thing is the same as the second thing

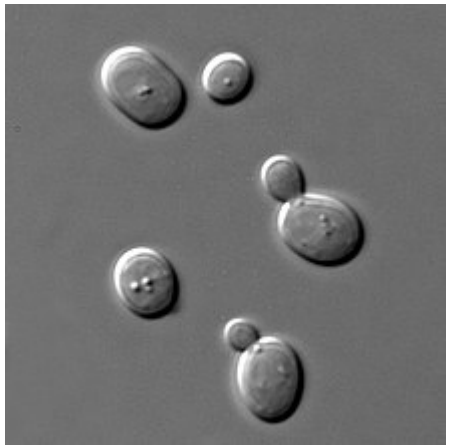
DNA sequencing!

- What organism are you interested in?
- How might you get DNA?



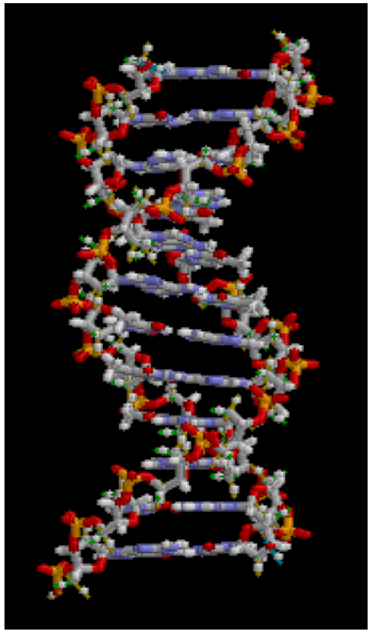




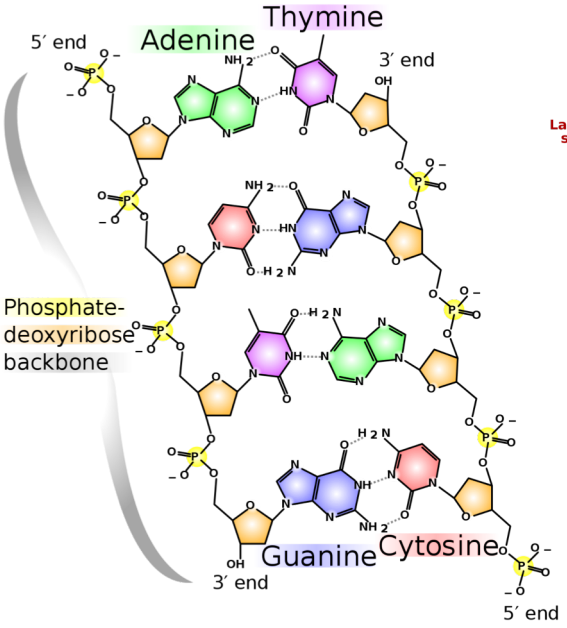


DNA sequencing!

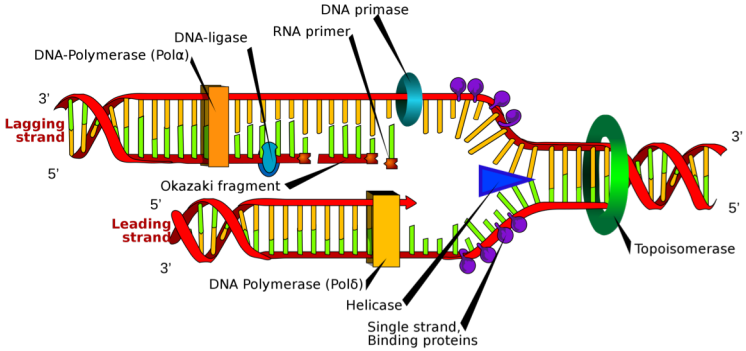
- You have to get that DNA ready for sequencing
- We call this sample prep
- The final processed DNA that is ready for sequencing is called your ***library***



By brian0918™ - Own work, Public Domain, https://commons.wikimedia.org/w/index.php?curid=404735



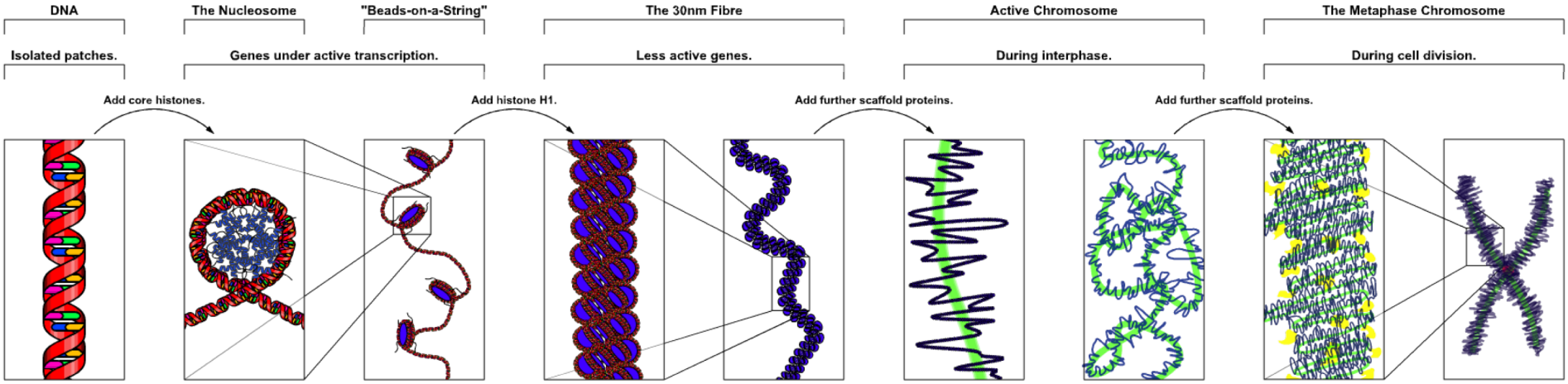
By Madprime (talk · contribs) - Own workiThe source code of this SVG is valid.This vector image was created with Inkscape., CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=1848174



By LadyofHats Mariana Ruiz - Own work (Original text: Own work. Image renamed from File:DNA replication.svg), Public Domain, https://commons.wikimedia.org/w/index.php?curid=6916971

DNA sequencing!

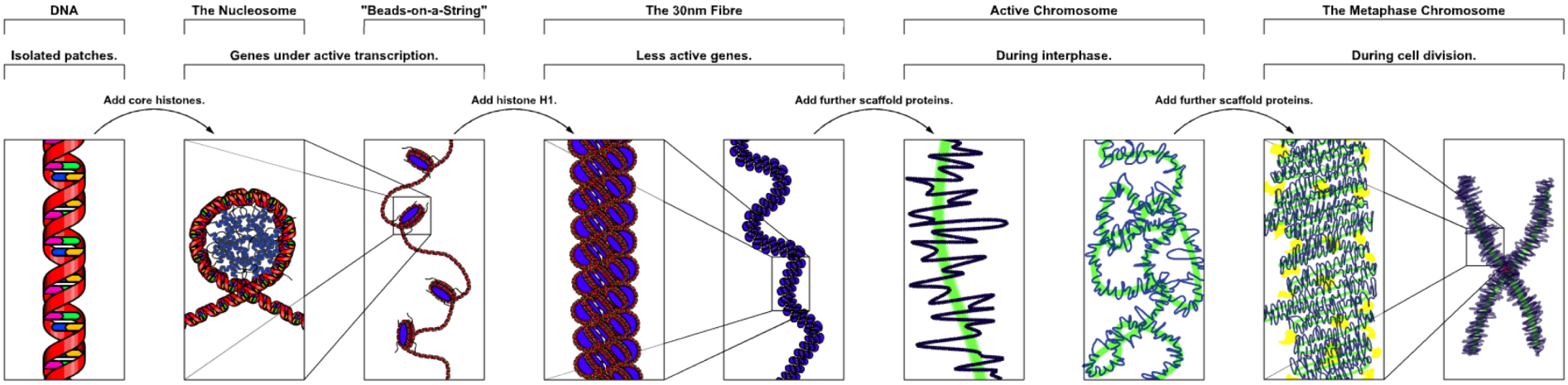
- You have to get that DNA ready for sequencing
- We call this sample prep
- The final processed DNA that is ready for sequencing is called your ***library***



Illumina is a company that makes the most popular sequencing platform (at this moment in time)

Basic principles underlying Illumina sequencing are no different than other molecular biology techniques you are probably familiar with

- Polymerase Chain Reaction (PCR)



DNA sequencing!

- How do you sequence

- https://www.youtube.com/watch?v=fCd6B5HRaZ8&feature=emb_logo

The Human Genome is pretty big

- One diploid human genome is 6 billion base pairs in length
- Illumina sequencing can only sequence pieces of DNA that are 100-150 bp in length
- You would need 40 million 150 bp fragments if you want to capture the whole human genome

- Illumina NovaSeq 6000 stats here:

S2 Flow Cell (one of the possible options)

8 human genomes per flow cell (at 30x coverage)

$16,000 Cost per flow cell

$2,000 Cost per genome

Chromosome #	size (base pairs)
1	248,956,422
2	242,193,529
3	198,295,559
4	190,214,555
5	181,538,259
6	170,805,979
7	159,345,973
8	145,138,636
9	138,394,717
10	133,797,422
11	135,086,622
12	133,275,309
13	114,364,328
14	107,043,718
15	101,991,189
16	90,338,345
17	83,257,441
18	80,373,285
19	58,617,616
20	64,444,167
21	46,709,983
22	50,818,468
X	156,040,895
Y	57,227,415
Haploid Genome	3,088,269,832
Diploid Genome	6,176,539,664

DNA sequencing!

- What do you do it next?
- Brainstorm questions that you could answer with Whole Genome DNA sequencing data

You can load DNA sequences into R!

- Back to RStudio
- Try typing some of the following into the console of Rstudio
  (Use some of your own examples too)

```
dna_1 = "ATGCAT"
dna_2 = "ATGCCT"
dna_3 = "ATGCAT"
print(dna_1)
```

What kind of an object is dna_1?

Character!

Maybe you want to check to see if there is a mutation in your DNA sequence?

- How might you do this in R?

- In RStudio:

```
dna_1 = "ATGCAT"
dna_2 = "ATGCCT"
dna_3 = "ATGCAT"

dna_ref = "ATGCAT"

dna_1 == dna_ref
dna_2 == dna_ref
dna_3 == dna_ref
```

Which object(s) has a mutation?

How do you know?

What type of object is the output?

This is overkill! Why do I need to use R?
I could see that dna_2 was different just by looking at it

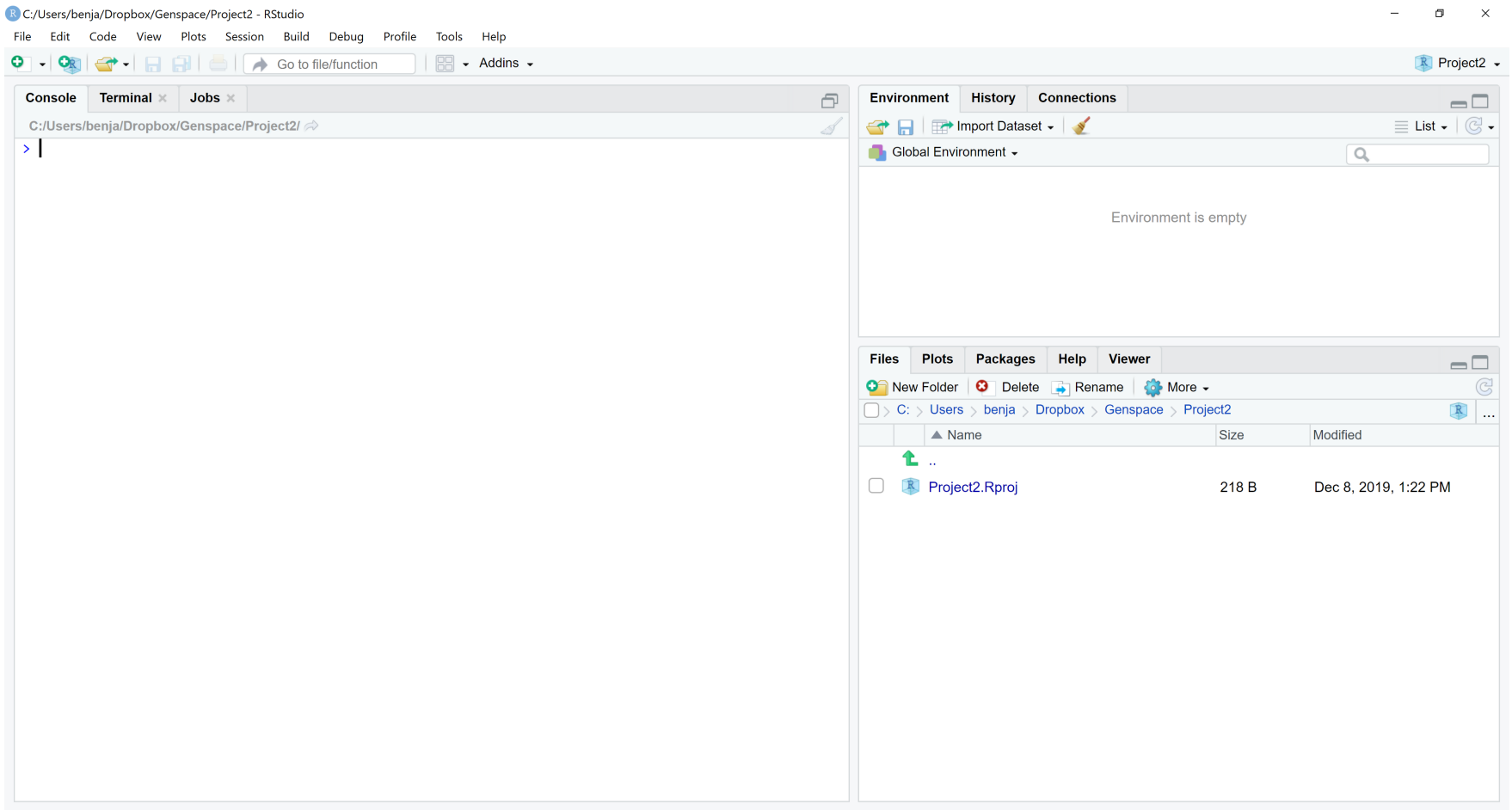
- In the Google Drive folder there are 4 Beta Hemoglobin coding sequences
- One sequence is labeled HBB_ref.fasta
- This is the reference sequence depositied at NCBI
- (NCBI – National Center for Biotechnology Information)
- NCBI is an invaluable resource where you can find lots of sequences that have been genereated by researchers all around the world
- Here: "reference sequence" is what the research community has decided is the standard human sequence of the HBB gene

This is overkill! Why do I need to use R?
I could see that dna_2 was different just by looking at it

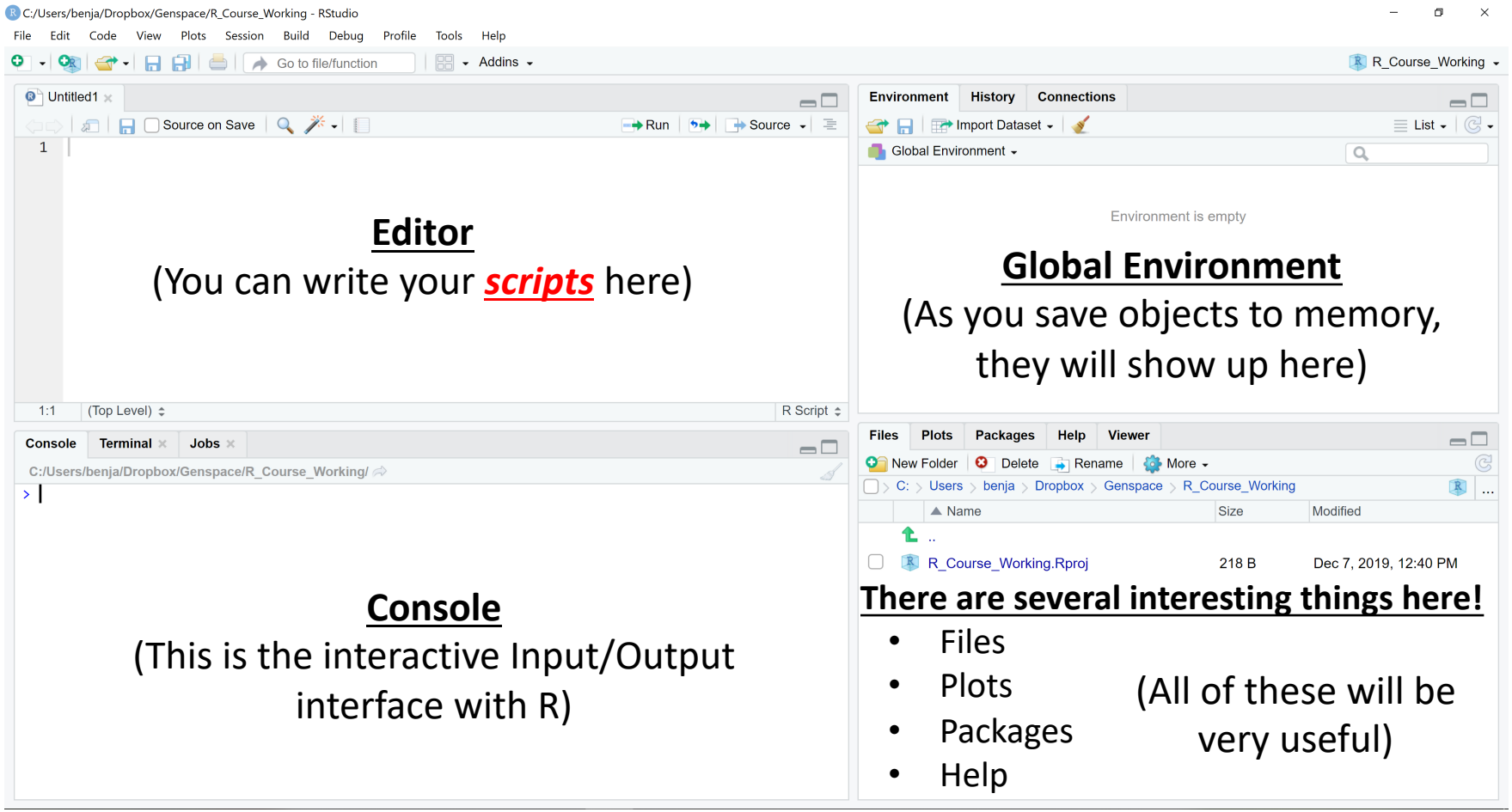
- In the Google Drive folder there are 4 Beta Hemoglobin coding sequences
- One sequence is labeled HBB_ref.fasta
- There are 3 additional HBB variant files saved in this folder
- Open these in your text editor
- Is there a mutation?
- Let's use R to figure that out

It is kind of tedious to always write one line of code at a time. Right?

- Let's write a ***script***!
- In RStudio, "File" → "New File" → "R Script"



Now you have a new panel in RStudio



Editor

(You can write your ***scripts*** here)

Global Environment

(As you save objects to memory, they will show up here)

Console

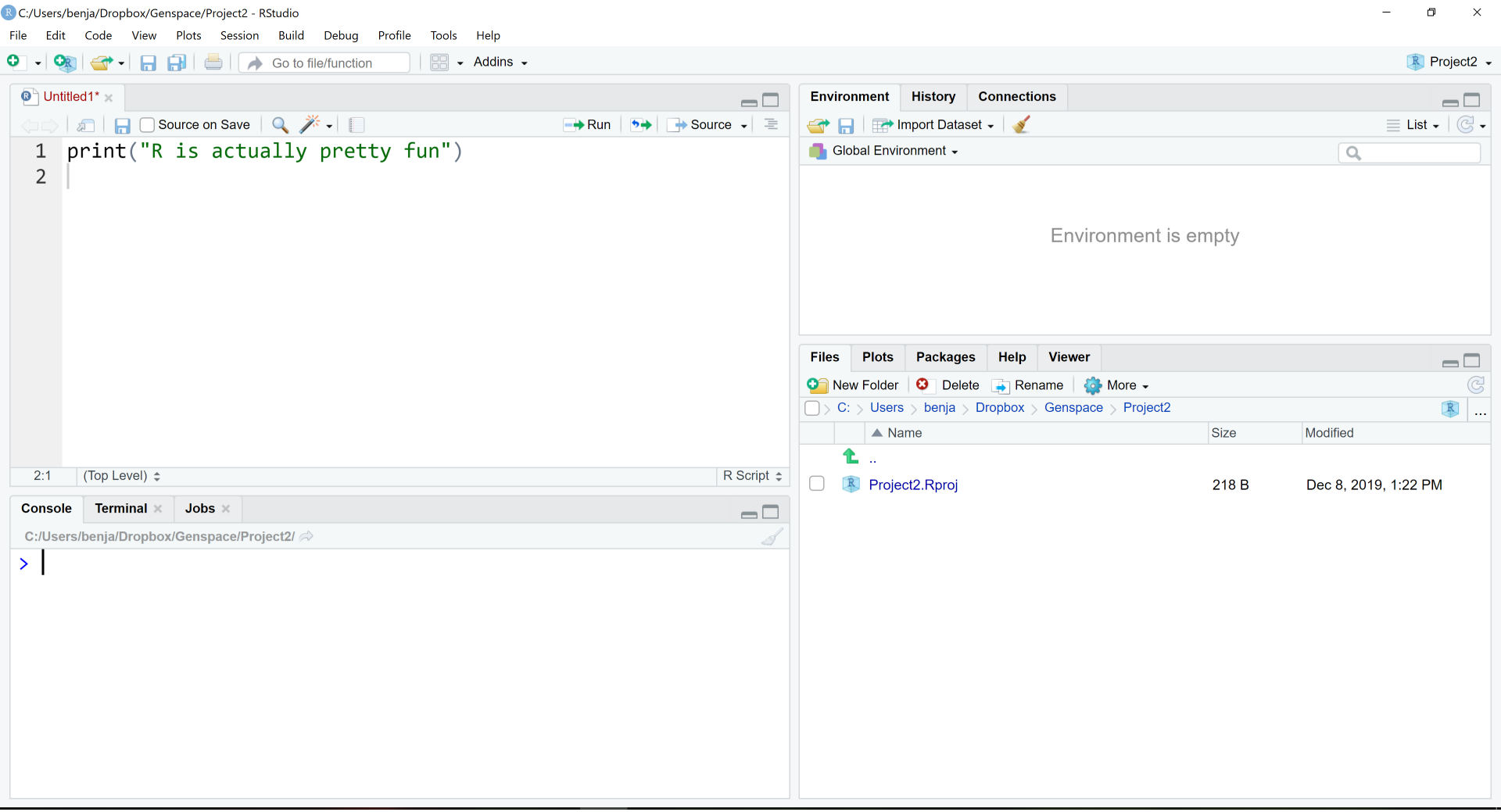
(This is the interactive Input/Output interface with R)

There are several interesting things here!

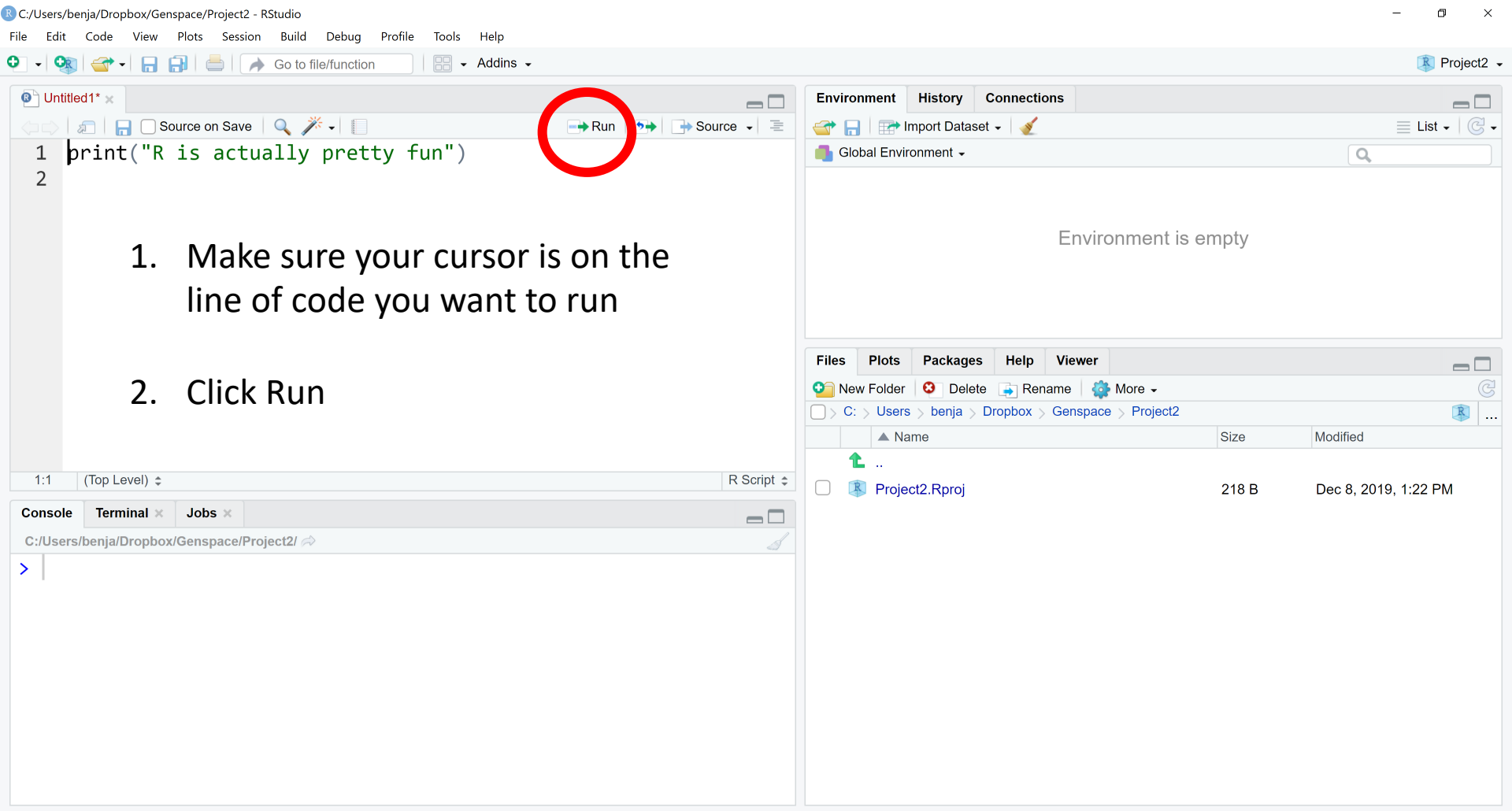
- Files
- Plots
- Packages
- Help

(All of these will be very useful)

Write some code in the editor panel in RStudio (Write whatever you want)

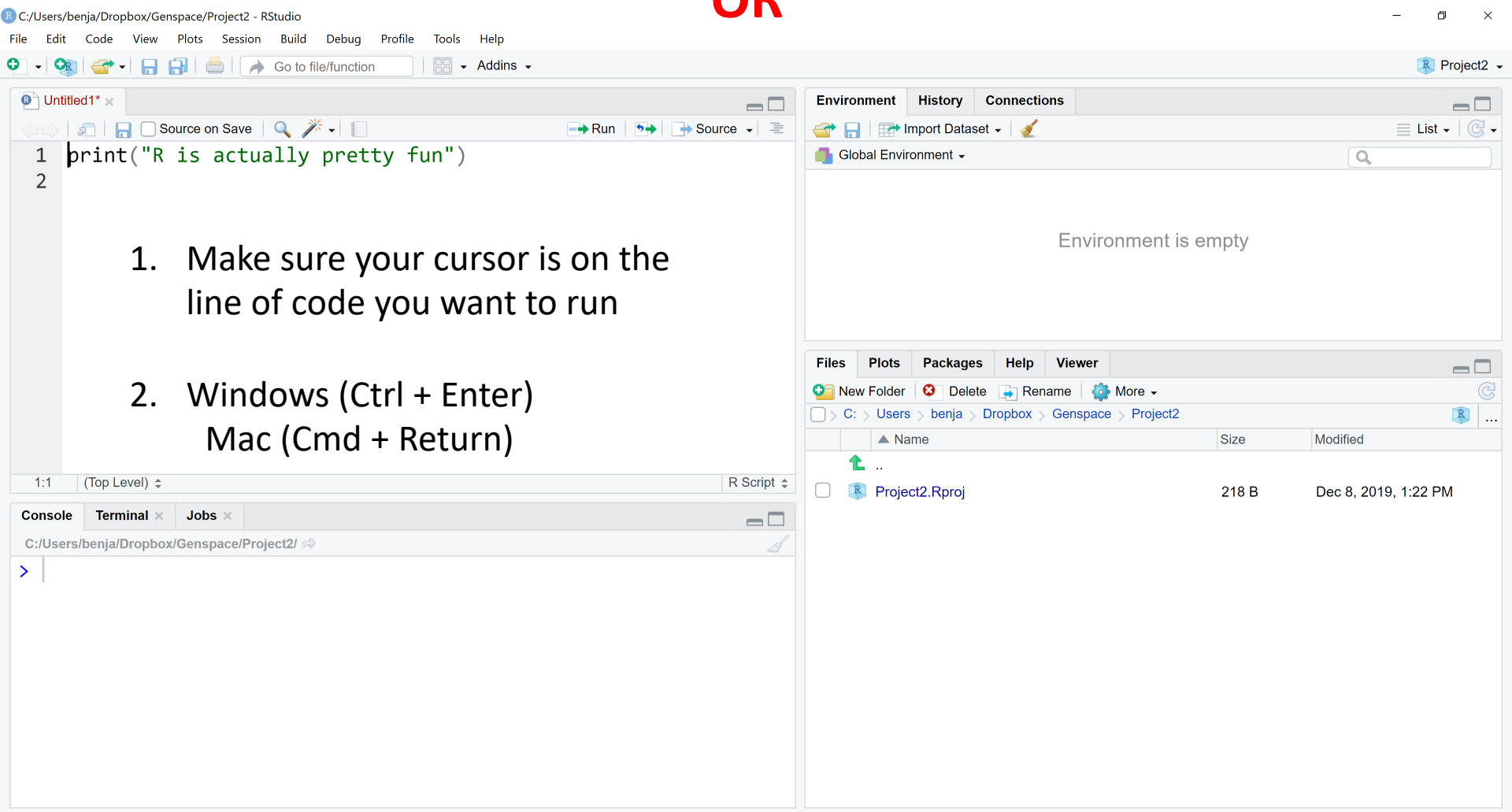


Now to run that code (You have several options)

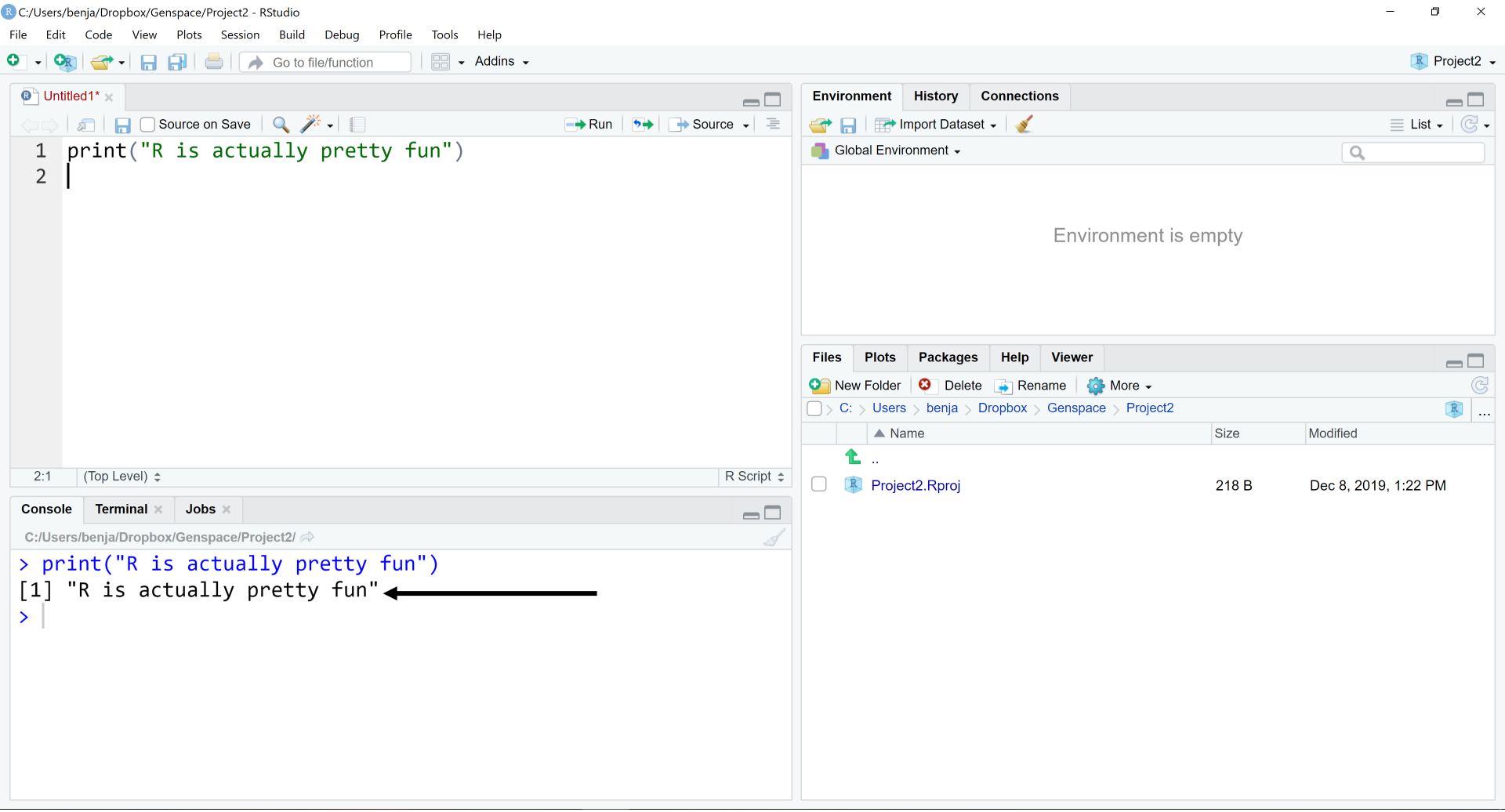


Now to run that code (You have several options)

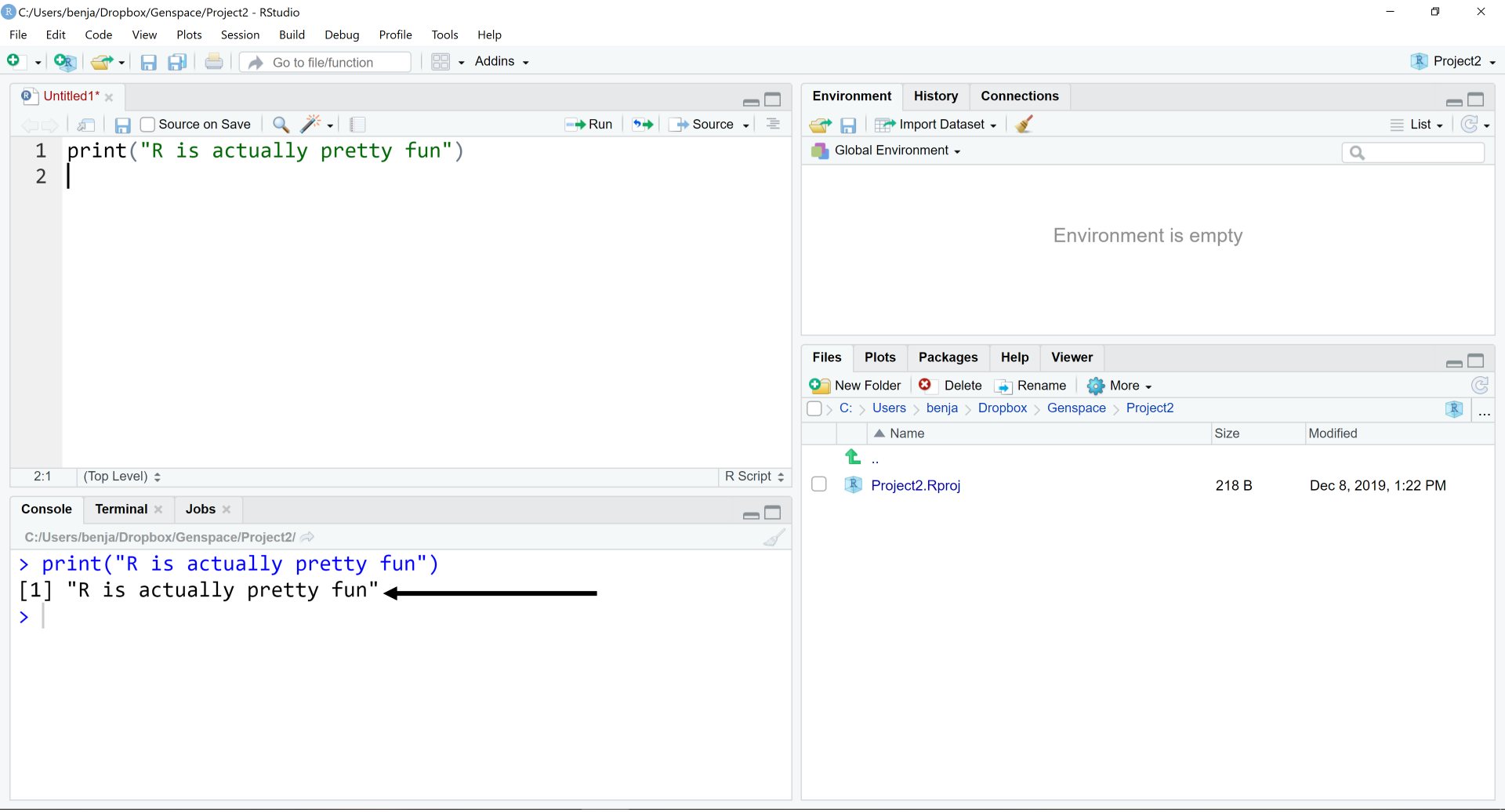
OR



And the output shows up in the console panel



Compare the HBB gene sequences again. But this time use editor panel to write a little script (Take a few minutes)



Let's make things more complicated!

- So far the objects we have created contain a single thing
- For example,

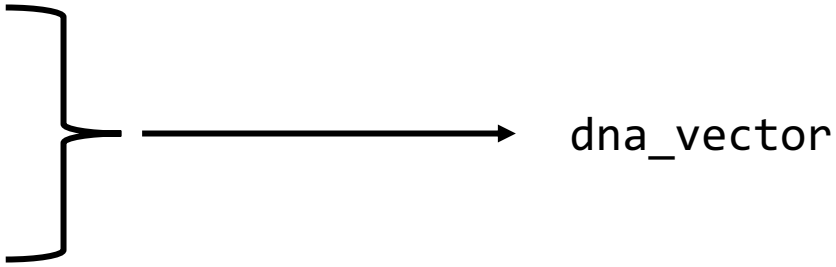
```
dna_ref = "ATGCAT"
dna_1 = "ATGCAT"
dna_2 = "ATGCCT"
dna_3 = "ATGCAT"
```

- But sometimes/often, data will be complex
- If you had the sequence of the hemoglobin beta gene from 20,000 people, would you want to create 20,000 variables to hold each sequence on its own?
- Probably not:
  - it would take a long time
  - it could be confusing
  - it would be easy to make a mistake

You can save multiple items within a single ***Vector***

- So you can take these four DNA sequences,

```
dna_ref = "ATGCAT"
dna_1 = "ATGCAT"
dna_2 = "ATGCCT"
dna_3 = "ATGCAT"
```



- And put them all into a single object
- And this new object will look like this:

```
[1] "ATGCAT" "ATGCAT" "ATGCCT" "ATGCAT"
```

- How do you think you might do that?
- (Hint, in R, what do you call an object that performs a task for you?)
- A ***function***!

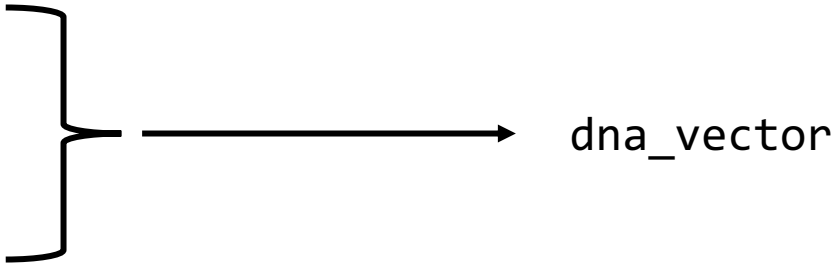
The art of Googling for answers to your R problems

- As you work more in R, you will get better and better

  (practice is helpful)
- But no one ever remembers everything
- there is no reason you should have to
- There are a lot of online resources to help you effectively use R
- There is a VERY large community of R users in the world
- I guarantee, when you experience a problem with R:
  - someone else has had the exact same problem,
  - they have posted that question to an internet forum (often on stackoverflow.com)
  - And many people have posted their helpful responses to help solve that problem
    (Sometimes their answers are hard to follow. This will get easier over time)
- **So back to our problem:**

How do you do this?

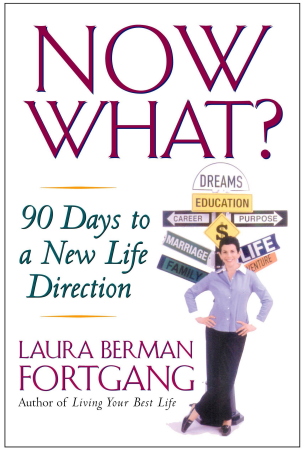
```
dna_ref = "ATGCAT"
dna_1 = "ATGCAT"
dna_2 = "ATGCCT"
dna_3 = "ATGCAT"
```



- How would you phrase a Google search to give you your answer?
- (Take a minute, talk with your partner, Google it)

- What is the function? **c( )**
- What were some of your Google search phrases?
- What were the important words to include?
- Anything weird or unexpected? Any trouble finding the specific info you needed?

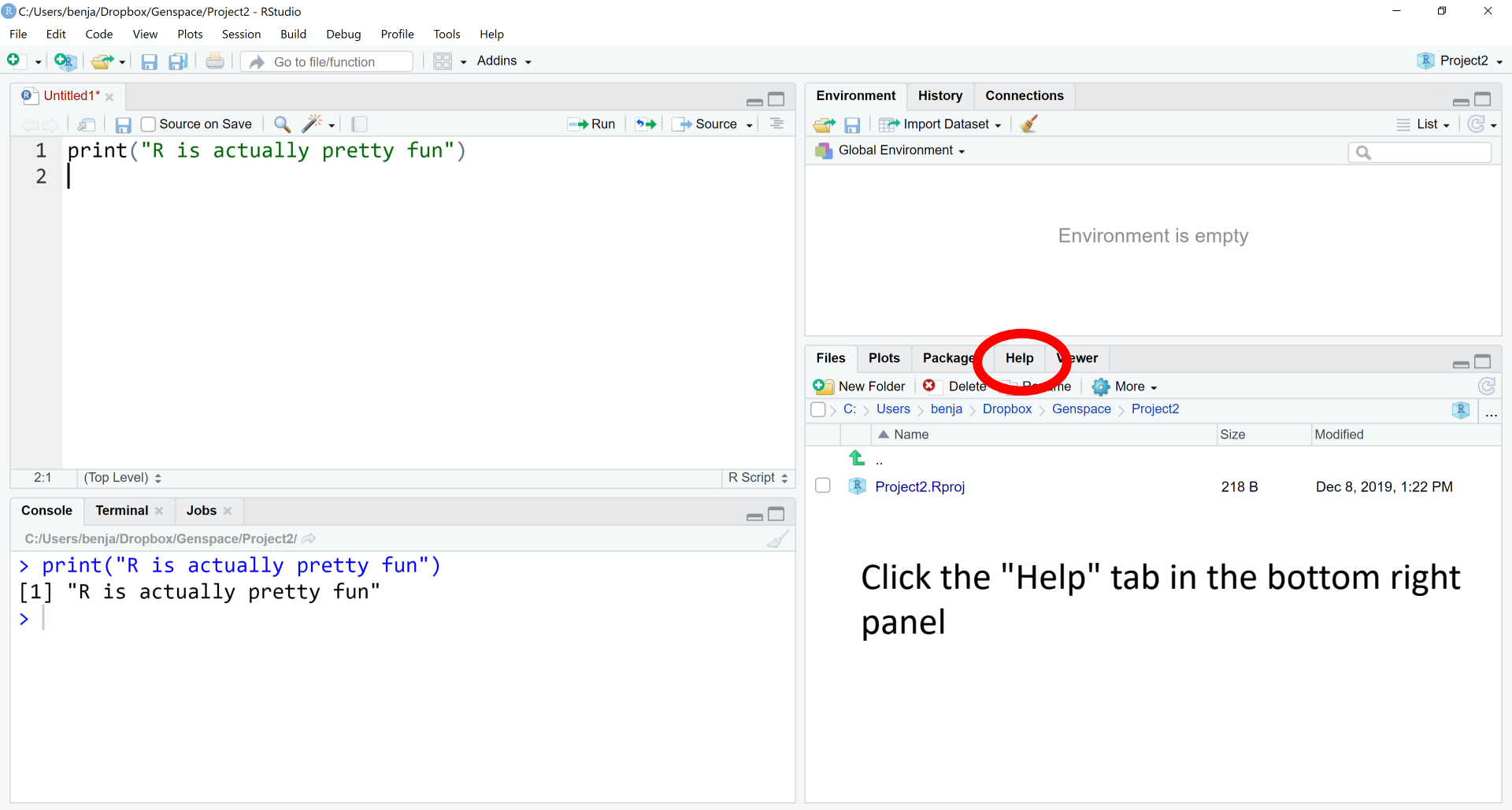
So you have found the right function to use, Now What?



- You need to know how the function works!
- This info is called ***documentation***
- All functions should have documentation
- You can find this info in a couple of different places:

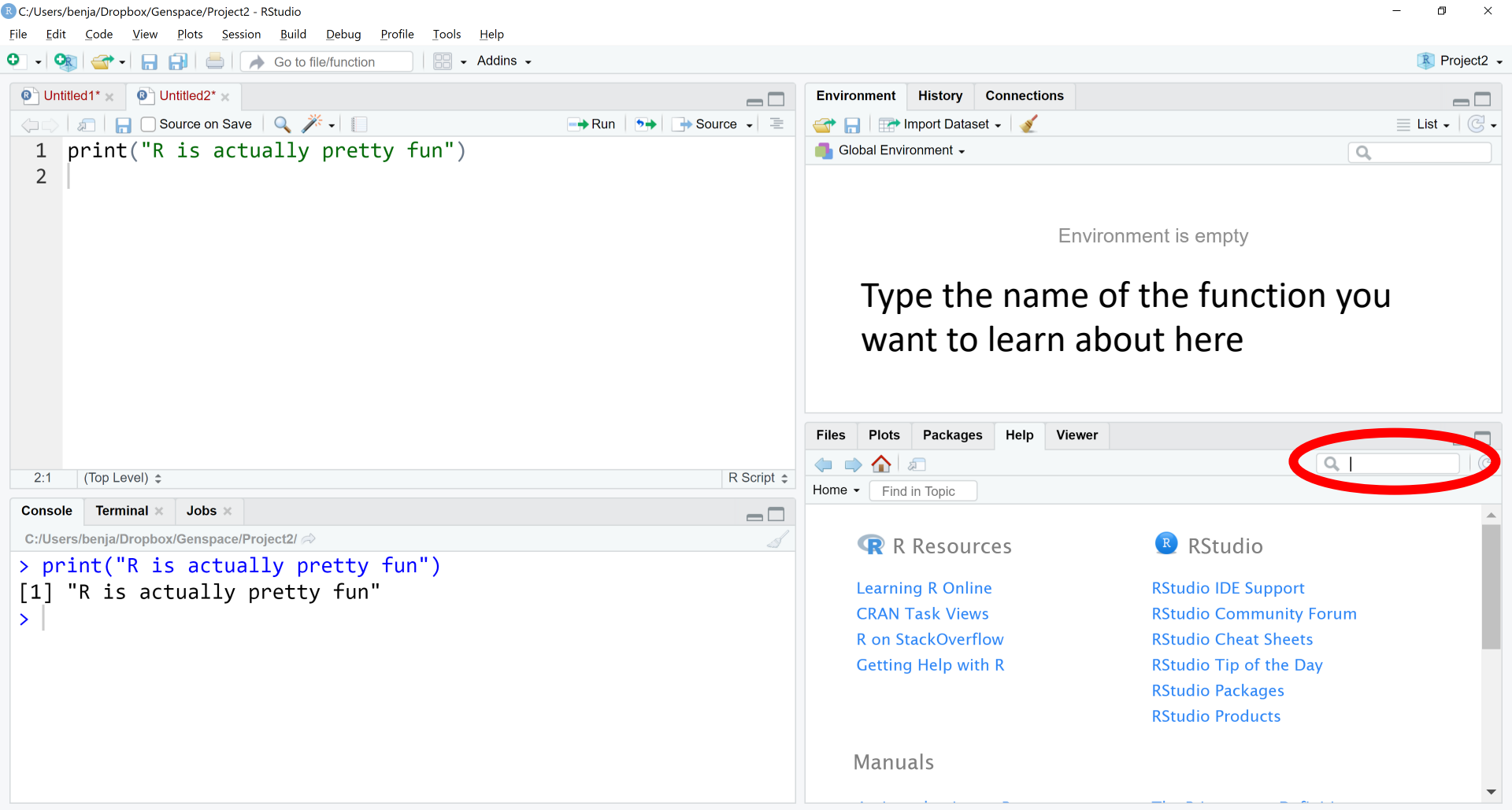
- Probably in one of the Google results you just opened
- Help within RStudio

RStudio's Help Panel

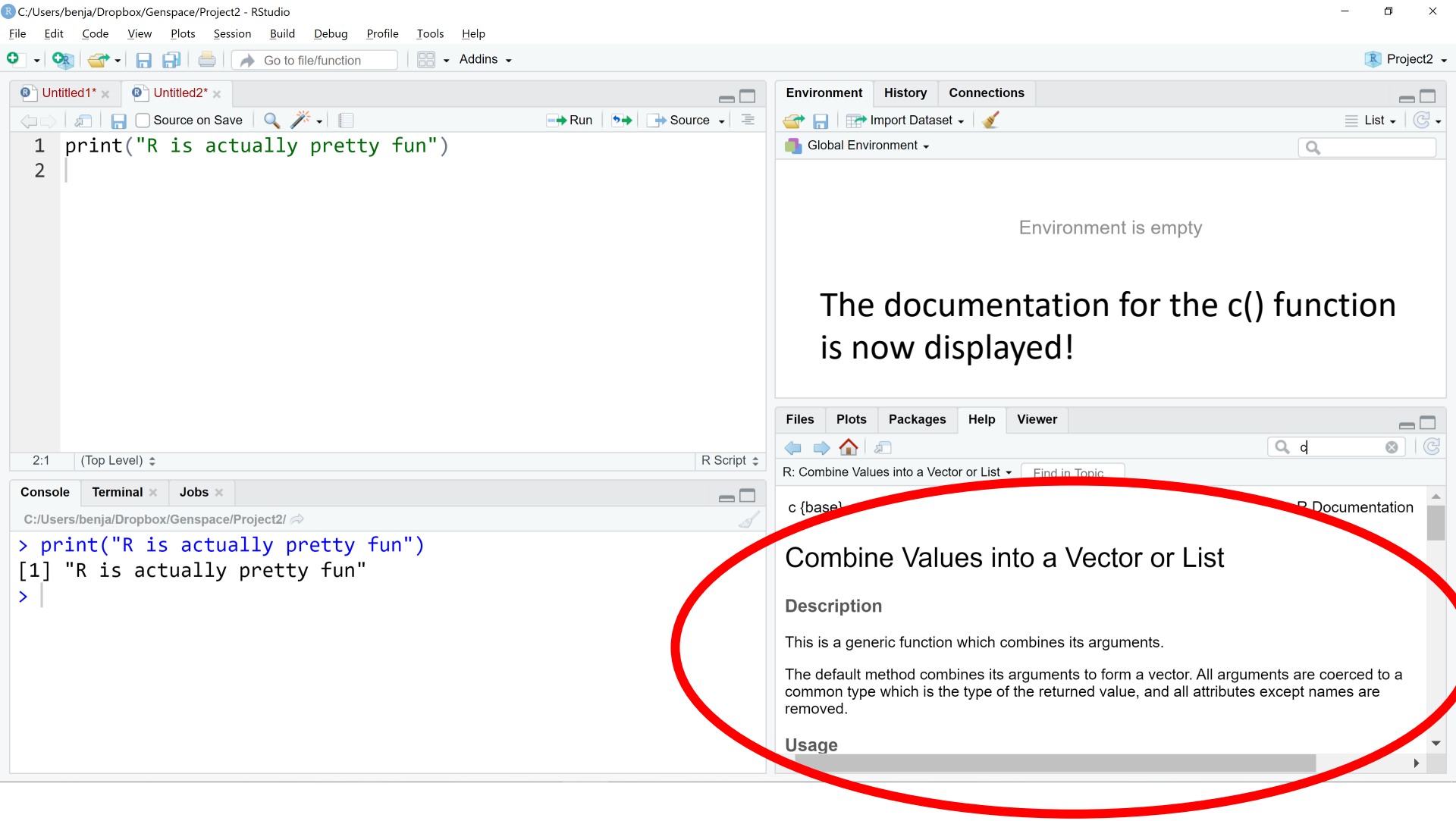


Click the "Help" tab in the bottom right panel

RStudio's Help Panel

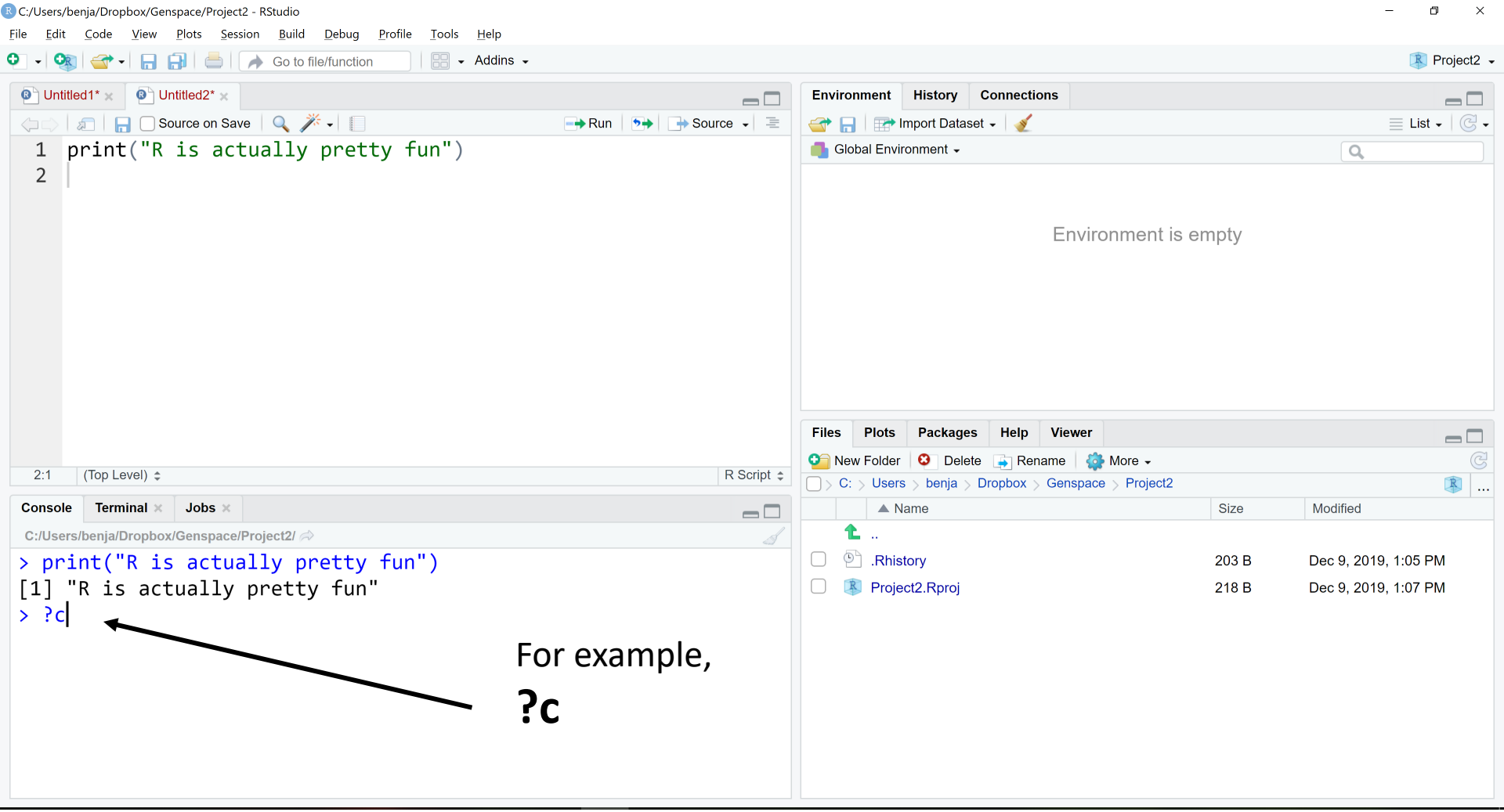


RStudio's Help Panel



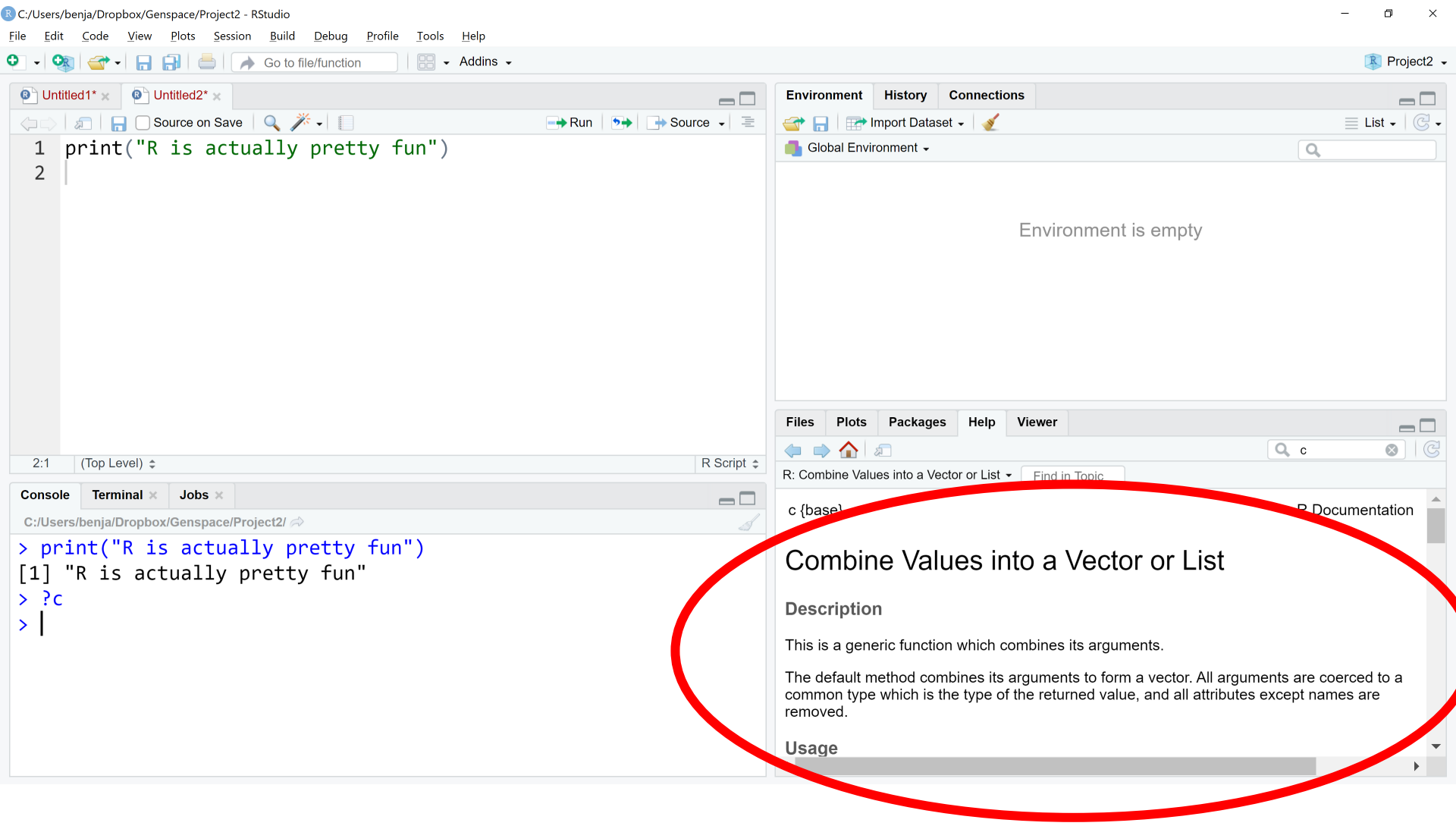
Alternative way to get help with a function:

- In the console, type **?{function_name}**

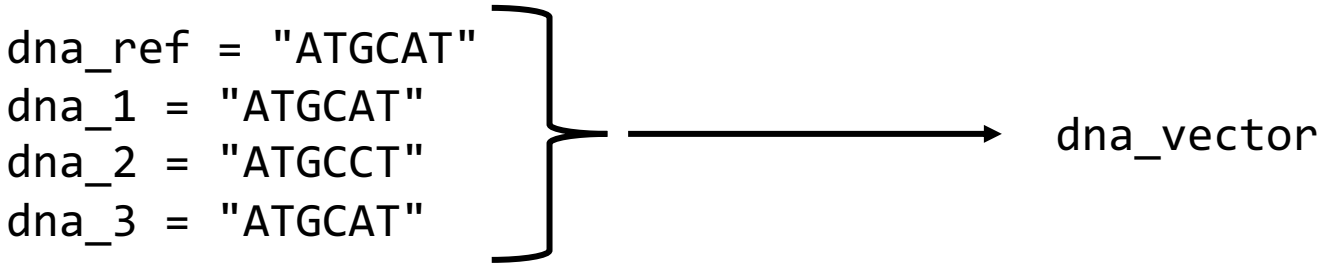


Alternative way to get help with a function:

- In the console, type **?{function_name}**



Do it (take a few minutes)



- Read the documentation for the c() function
- (There is always more info than you need, look for what you need, ignore the rest)
- Make a vector combining these 4 DNA sequences and set them equal to a new variable you create called **dna_vector**

Hint 1:

- A ***function*** always has parentheses
- The stuff inside its parentheses are called ***arguments***
- A ***function*** is a little machine. It grabs the ***arguments*** that you give it, it does something behind the scenes, and gives you (***returns***) something else

Hint 2:

- So far, we have used the = sign to set a value to a variable
- There are actually two ways to do this.
- The two ways are (more or less) identical
- The other way is to use **<-** (the less than sign followed by a dash). Think of it like a little arrow funneling the thing on the right into the variable on the left

So these two expressions are identical:

```
dna_ref = "ATGCAT"
dna_ref <- "ATGCAT"
```

Did you have something like this?

```
dna_ref = "ATGCAT"
dna_1 = "ATGCAT"
dna_2 = "ATGCCT"
dna_3 = "ATGCAT"

dna_vector = c(dna_ref,dna_1,dna_2,dna_3)
```

Any other solutions?

Other things to know about vectors in R

- Every item stored in a vector is called an ***element***
- The position of an ***element*** within a vector is called its ***index***
- In RStudio make this new vector

```
animal_vector = c("Human", "Mouse", "Fly", "Worm")
```

- In the new object you just made, **animal_vector**, what is the ***index*** of "Human"?
- 1
- It seems obvious, but in many other programming languages, the index of the first item in a vector (or vector like object) is arbitrarily called 0

- You can add elements to an existing vector
- Some functions work on vectors

Activity (Break out groups):

Figure out how to solve the following problems using Google: (search "working with vectors in r" the first hit is very helpful)

- Write a line of R code that will return the 4th element in animal_vector
- Write a line of R code that will return the first 3 elements of animal_vector
- Write a line of R codes that will return elements 2 and 4 of animal_vector

- Uh oh, You realized you forgot that "Panda", and "Maple" should be in your vector animal_vector
- Add them
- Calculate (using a function) how many species are in your vector
- Uh oh, you realize that the last element is a plant. Please remove it
- How many elements are in your vector now?
- Please make two new vectors one called vertebrate_vector containing the vertebrate species and one called invertebrate_vector containing the invertebrate species (** Accomplish this using what you have learned about indices)