Sept 30, 2020

Programming basics
with Python and R

Suzy Strickler
srs57@cornell.edu
Boyce Thompson Institute

Practice Exercises

a) Go to your Desktop directory

b) Create a file called:
Do not Use “special characters” in file names!.txt

c) Delete the file you created in exercise b)

d) Create a folder called unix_data in your desktop

e) Find the file called unix_class_file_samples.zip

f) Copy unix_class_fild_samples.zip to the folder unix_data, in your desktop

g) Uncompress the file unix_class_file_samples.zip in /home/bioinfo/Desktop/unix_data

h) Remove the _MACOSX folder

Solutions from last time

a) cd Desktop **(from your home: /home/bioinfo/) or** cd /home/bioinfo/Desktop **or** cd ~/Desktop
b) touch Do\ not\ Use\ \“special\ characters\”\ in\ file\ names\!.txt
c) rm Do\ not\ Use\ \“special\ characters\”\ in\ file\ names\!.txt **(use the tab key)**
d) mkdir unix_data **(from Desktop: /home/bioinfo/Desktop) or** mkdir /home/bioinfo/Desktop/unix_data **or** mkdir ~/Desktop/unix_data
e) locate unix_class_file_samples.zip
f) cp /home/bioinfo/Data/unix_class_file_samples.zip /home/bioinfo/Desktop/unix_data
g) cd /home/bioinfo/Desktop/unix_data && unzip unix_class_file_samples.zip
h) rm -r _MACOSX

Text Editors for Programming

- Don’t write code in Word!
- Vi, Vim, Emacs
- **Atom**
- Sublime
- Visual Studio

What are scripts?

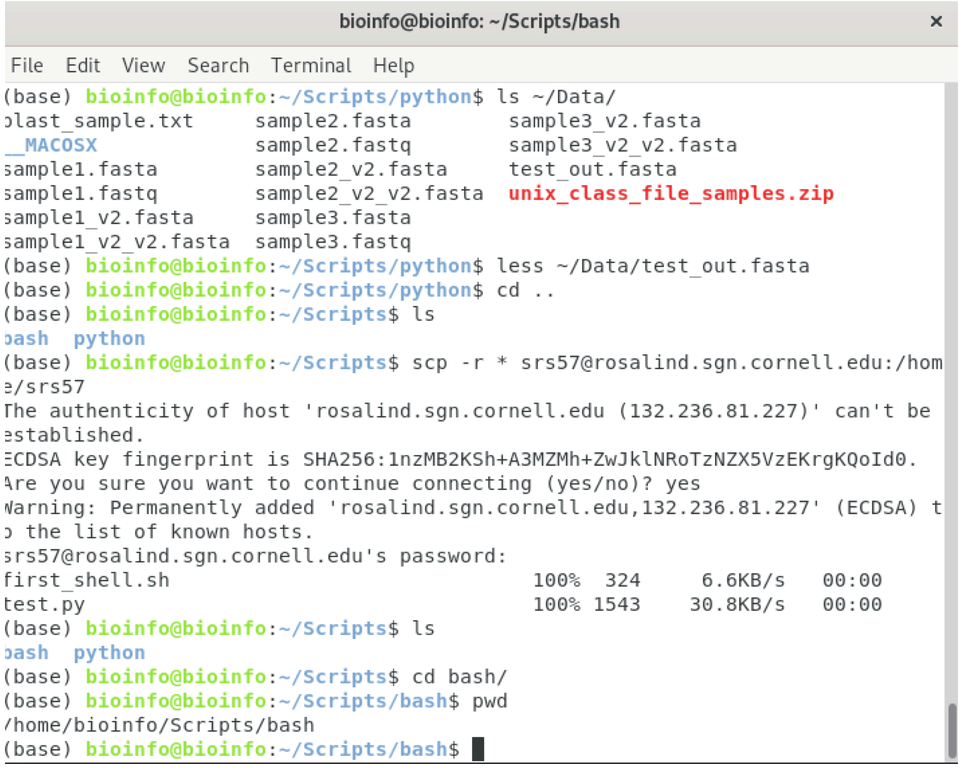
- A specific type of coding
- A text document that contains instructions to be executed by a program
- Usually combine existing components
- Usually use an interpreter instead of compiler to translate to machine code
    - **Bash**, **Python**, Perl, JavaScript, PHP, Ruby

Why use scripts?

- Automation of step-by-step processes
- Reproducibility
- Share with others

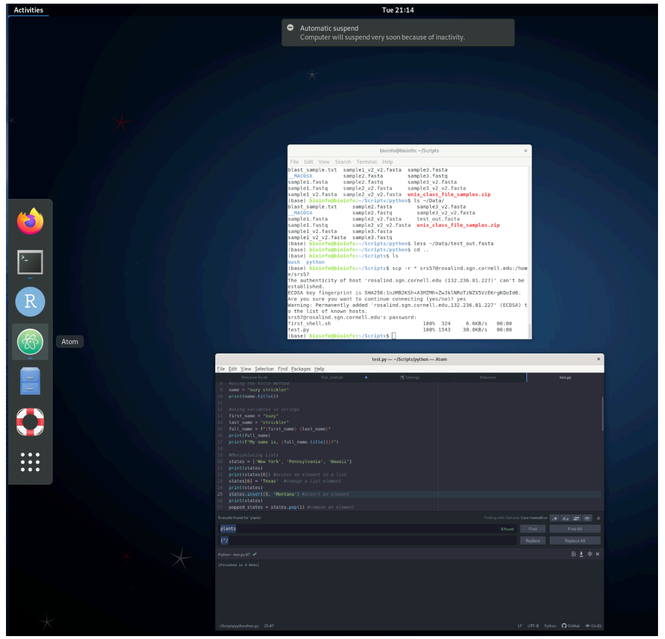
Simple Scripting with Bash

- Bash is a Unix shell, both a command language and scripting language
- Configuration file, bash.rc, is written in bash
- A bash shell script contains a list of commands
  - Since we are using the bash shell in our vm, these are the same commands we went over last time

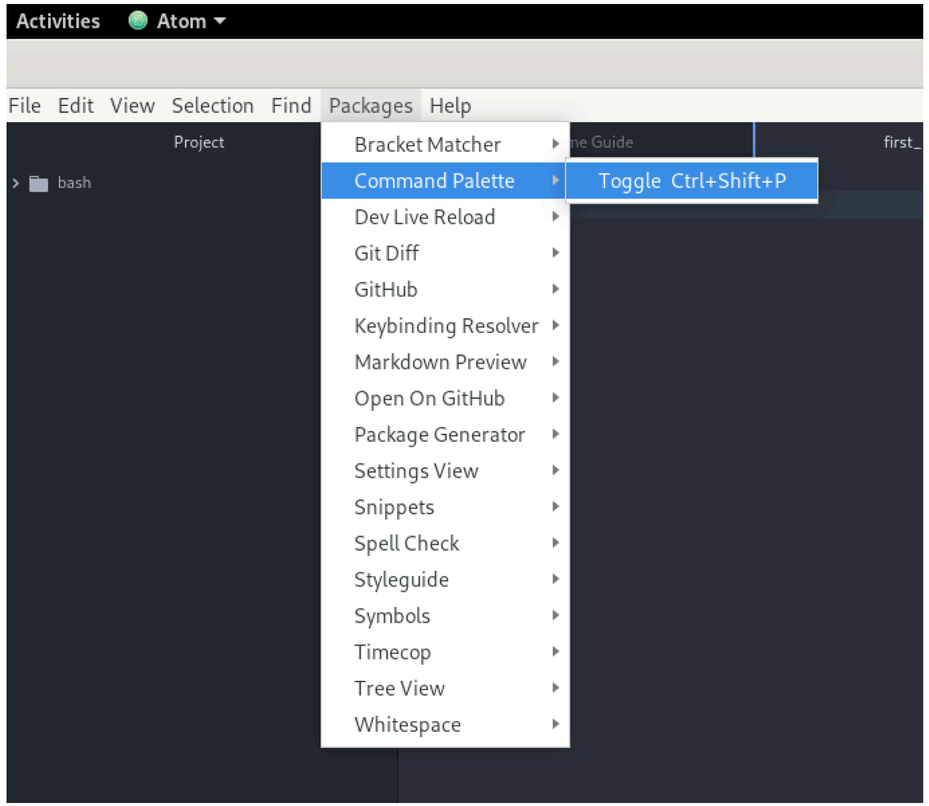


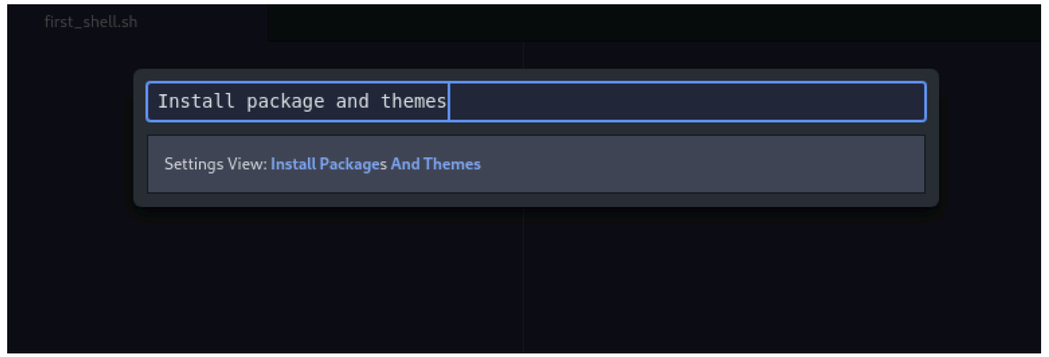
Let’s create a bash shell script

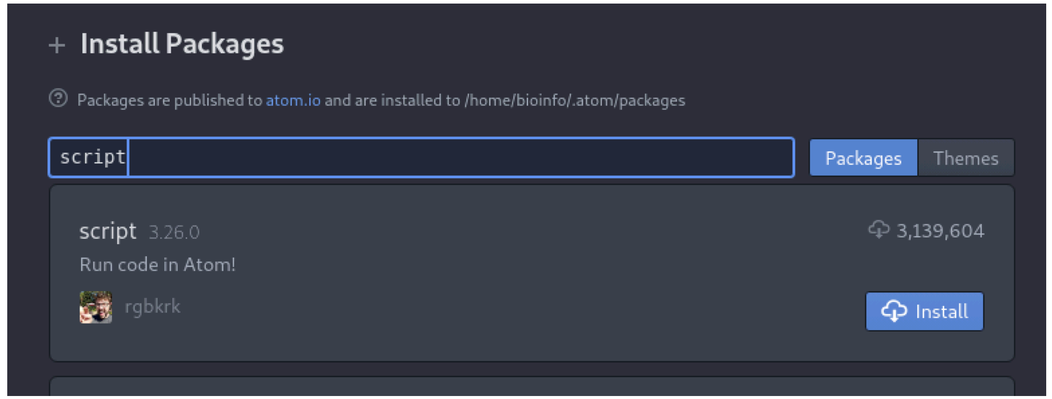
- Open Atom on your VM
- File -> Add Project Folder
  - Select /home/bioinfo/Scripts
  - New folder: “bash”
- File -> New file



Install script Package in Atom



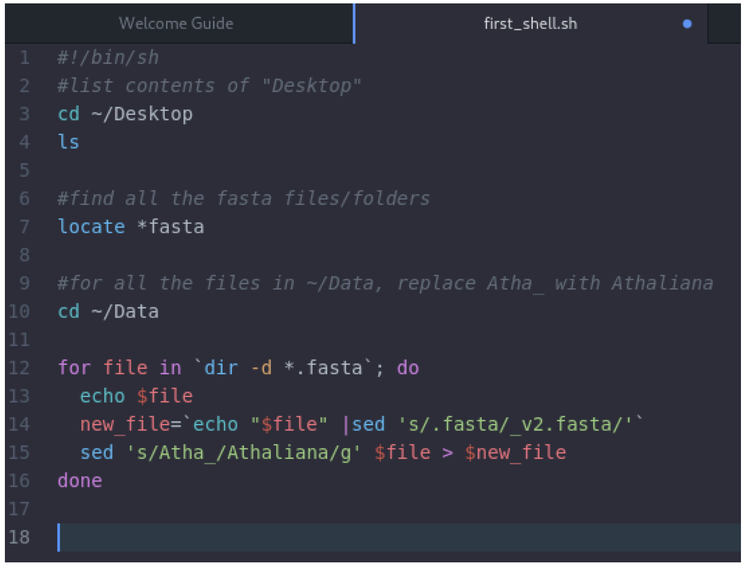




- Ctl + shift + b

Let’s create a bash shell script

We’ll call it first_shell.sh



We can run the script from within Atom
Shift + Ctl + b

We can also run it from the command line
bash ~/scripts/bash/test.sh

Why use bash?

- It is useful for chaining other programs together.
- This allows one to automate tasks and build pipelines.

The Python Language

- First released in 1991
- Efficient and elegant - fewer lines of code
- Object-oriented
- Runs on many operating systems
- Monty Python
- Python 2 (discontinued 2020) vs Python 3 - not completely backward compatible

Setting up Python

- Linux - usually already installed but may need updating
- OSX
- Windows
- What version is on our VM?

Code Libraries

- Modularizing code
  - Breaking apart problem into individual units
  - Easier to maintain
  - Reusable
- Modules, Packages - grouping of modules, Library - a collection of packages
- Libraries we will use:
  - BioPython, NumPy, Pandas in later classes
- Package Repos
  - Python Package Index (PyPi) and Anaconda

Comments

- Allows you to add human readable text to your code
- In Python comment lines start with: #
- Allows you and others to understand your code

Let’s look at some example Python code

- Can run programs in Atom (ctl+shift + b)
- Can run Python in a terminal session

```
>>> quit()
(base) bioinfo@bioinfo:~/Scripts/python$ python
Python 3.8.3 (default, May 19 2020, 18:47:26)
[GCC 7.3.0] :: Anaconda, Inc. on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

- Can run as a Python script from the command line

```
base) bioinfo@bioinfo:~/Scripts/python$ python test.py
```

- **print(“Python is fun!”)**

```
1  #testing out python
2  print("Python is fun!")
```

Variables

- Labels that are assigned to values
- Names contain only letters, numbers, and underscores. They cannot start with a number and cannot contain spaces
- Do not use function names, for example print
- Keep names short but descriptive
- Example:

message = “Python is fun!”

print(message)

```
4  #using variables
5  message = "Python is fun!"
6  print(message)
```

Strings

- A series of characters
- Are surrounded by single or double quotes
- Example:

  “This is a string”

  ‘This is also a string’

Methods

- An action Python can perform on a piece of data
- Example:

name = “suzy strickler”

print(name.title())

```
 8  #using the title method
 9  name = "suzy strickler"
10  print(name.title())
```

Variables in Strings

- Example:

first_name = “suzy”

last_name = “strickler”

full_name = f”{first_name} {last_name}”

print(full_name)

print(f”My name is, {full_name.title()}!)

```
12  #using variables in strings
13  first_name = "suzy"
14  last_name = "strickler"
15  full_name = f"{first_name} {last_name}"
16  print(full_name)
17  print(f"My name is, {full_name.title()}!")
18
```

Lists

- A collection of items in a particular order
- Example:

  states = [‘New York’, ‘Pennsylvania’, ‘Hawaii’]

  print(states)

- Accessing elements of a list:

  print(states[0])

- Changing an element

  states[0] = ‘Texas’

  print(states)

```
19  #Manipulating Lists
20  states = ['New York', 'Pennsylvania', 'Hawaii']
21  print(states)
22  print(states[0]) #access an element in a list
23  states[0] = 'Texas'  #change a list element
24  print(states)
25  states.insert(0, 'Montana') #insert an element
26  print(states)
27  popped_states = states.pop(1) #remove an element
28  print(popped_states)
29  print(states)
```

Lists cont’d

- Adding elements

  states.insert(0, ‘Montana’)

  print(states)

- Removing elements

  popped_states = states.pop(1)

  print(popped_states)

```
19  #Manipulating Lists
20  states = ['New York', 'Pennsylvania', 'Hawaii']
21  print(states)
22  print(states[0]) #access an element in a list
23  states[0] = 'Texas'  #change a list element
24  print(states)
25  states.insert(0, 'Montana') #insert an element
26  print(states)
27  popped_states = states.pop(1) #remove an element
28  print(popped_states)
29  print(states)
```

Loops

- Repeat an action with every item in a list
- Indentation
- Example

for state in states:

print(state)

print(“All states in list printed”)

```
31  #Loops
32  states = ['New York', 'Pennsylvania', 'Hawaii']
33  for state in states:
34      print(state)
35      print("All states in list printed")
36
```

If statements

```
for state in states:

    if state == ‘Hawaii’:

        print(state.upper())

    elif state == “New York”:

        print(state.lower()

    else:

        print(state.title())
```

```
37  #Loops - if statements
38  states = ['New York', 'Pennsylvania', 'Hawaii']
39  for state in states:
40      if state == 'Hawaii':
41          print(state.upper())
42      elif state == 'Montana':
43          print(state.lower())
44      else: print(state.title())
```

Dictionaries

- A collection of key-value pairs
- Example:

plants = {‘color’: ‘blue’, ‘height’: 10}

print(plants[‘color’])

#add a new key-value pair

plants[‘sepal’] = 3

print(plants)

#remove a key-value pair

del plants[‘color’]

```
46  #Dictionaries
47  plants = {'color': 'blue', 'height': 10}
48  print(plants['color'])
49  plants['sepal'] = 3  #add a new key-value pair
50  print(plants)
51  del plants['color']  #remove a key-value pair
52  print(plants)
53
```

Functions

- Blocks of code that do a specific job
- docstring = describes what the function does
- Example

def weather():

“””Find out the weather”””

answer = input(“How is the weather?”)

print(answer)

```
54  #example of a function
55  def weather():
56      """Find out the weather"""
57      answer = input("How is the weather?")
58      print(answer)
59  weather()
```

- Module - storing function in a separate file and then import it (**import weather**)

Reading from a file

- In many cases, you will want your script to operate on some input file
- Example:

```
fasta_file = open("/home/bioinfo/Data/sample1.fasta", "r")

if fasta_file.mode == "r":

    contents = fasta_file.read()

    print(contents)

fasta_file.close()
```

Writing output to a file

- Example:

```
fasta_file = open("/home/bioinfo/Data/sample1.fasta", "r")

if fasta_file.mode == "r":

    contents = fasta_file.read()

   outF = open(“/home/bioinfo/Date/test_out.fasta”, “w”)

    outF.write(contents)

    print(contents)

fasta_file.close()
```

```
61  #reading in a file
62  fasta_file = open("/home/bioinfo/Data/sample1.fasta", "r")
63  if fasta_file.mode == "r":
64      contents = fasta_file.read()
65      outF = open("/home/bioinfo/Data/test_out.fasta", "w")
66      outF.write(contents)
67  fasta_file.close()
68  outF.close()
```

Python Package managers

- pip

```
pip install "SomeProject==1.4"
```

- Conda

```
conda install package-name=2.3.4
```

  - Virtual environments

General Introduction - R and RStudio

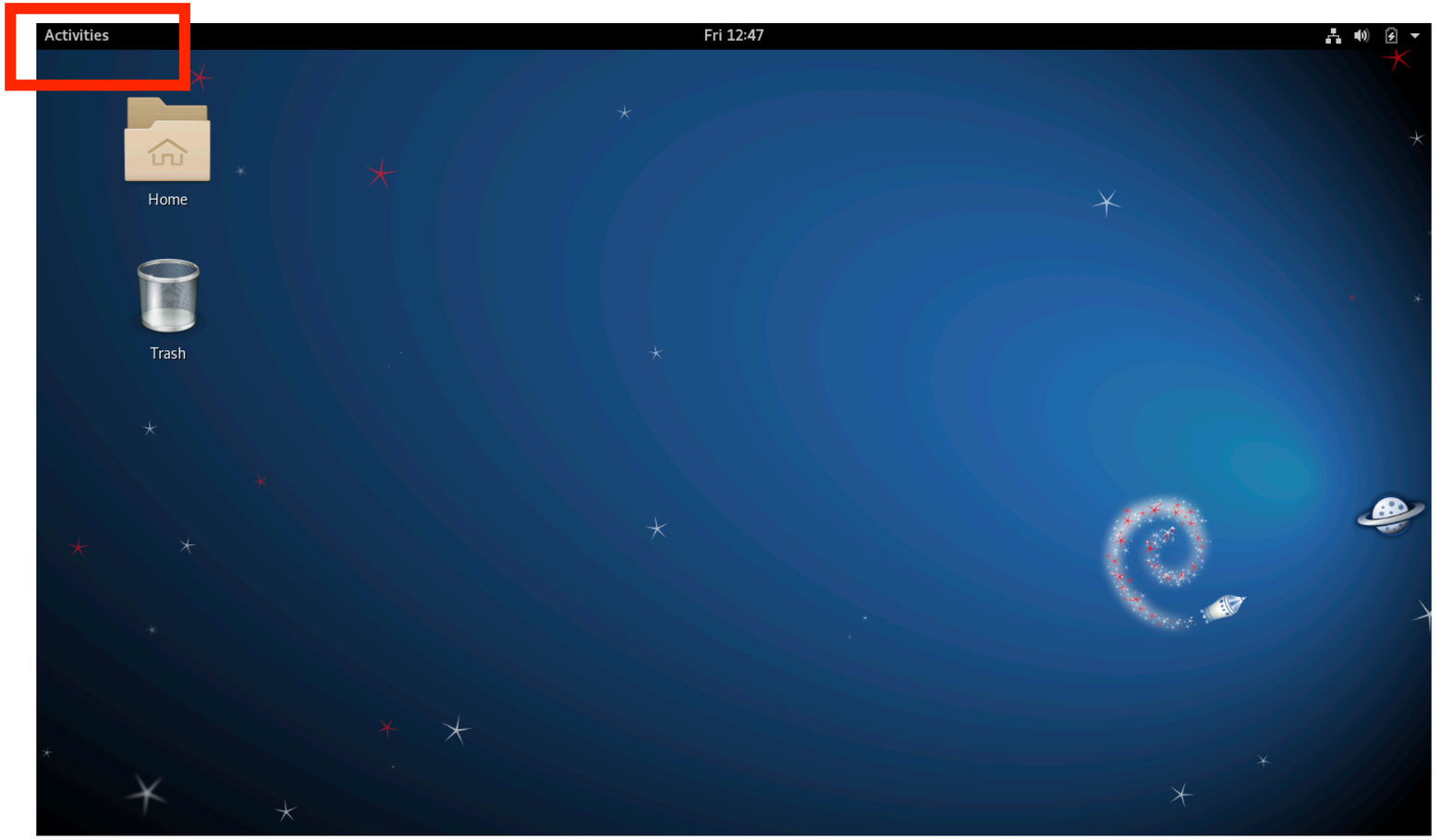
- What is R?
  - A programming, ‘scripting’ language

- Why use R?
  - Community
  - Free, open-source
  - Useful for statistics and data visualization

- How to use R?
  - We will use RStudio, a user-friendly interface

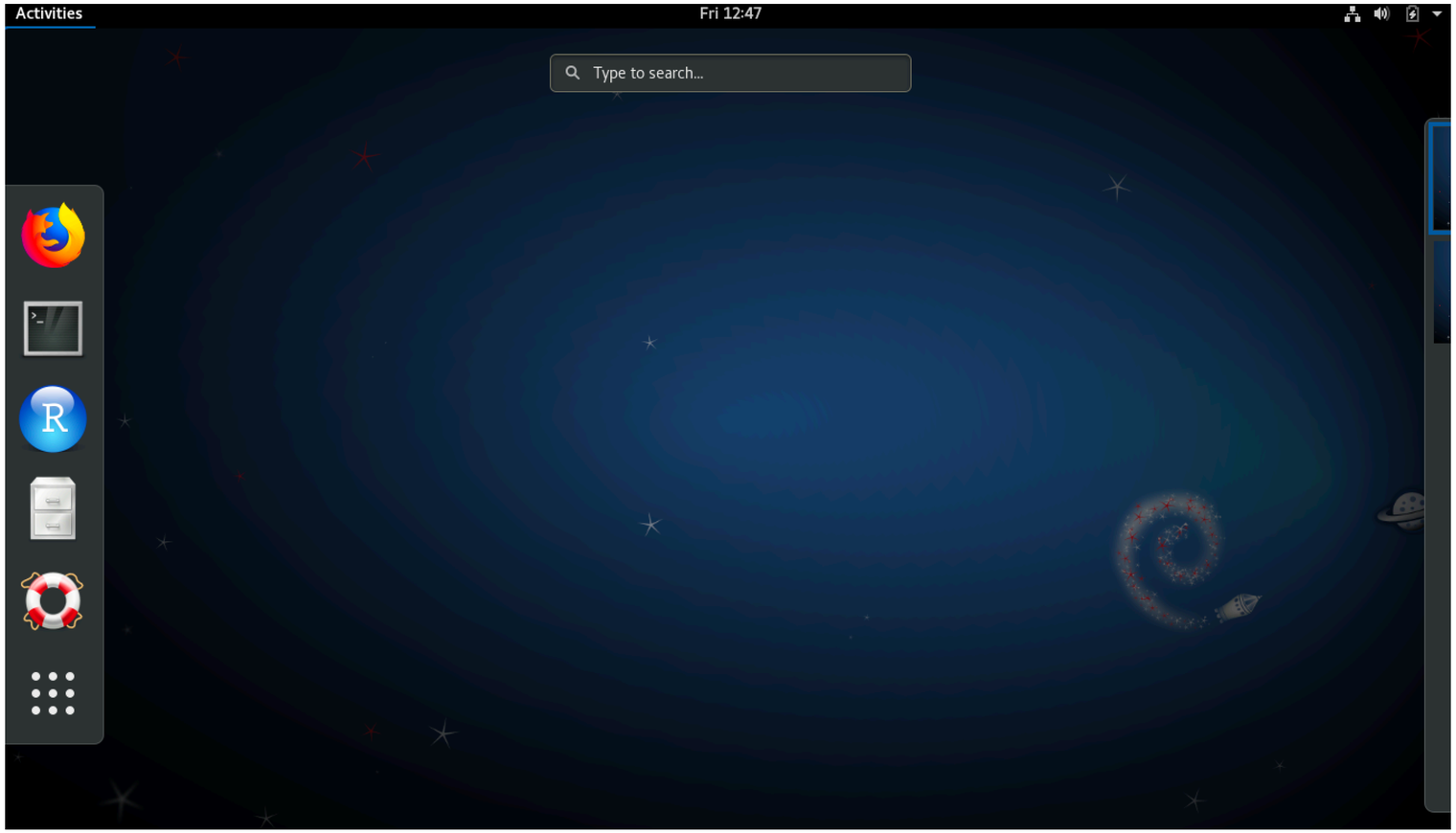
General Introduction - R and RStudio

- Let’s open RStudio



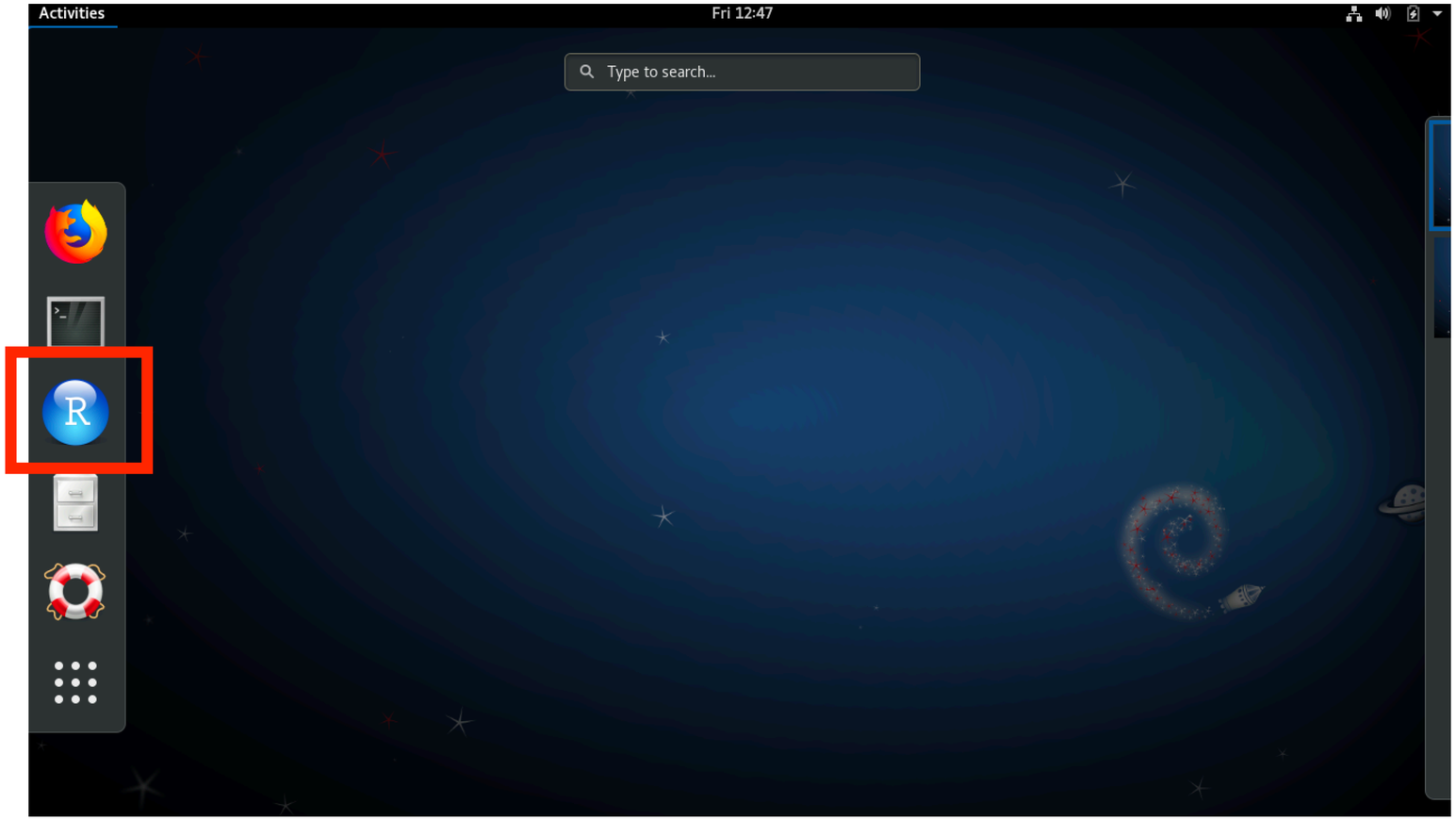
General Introduction - R and RStudio

- Let’s open RStudio



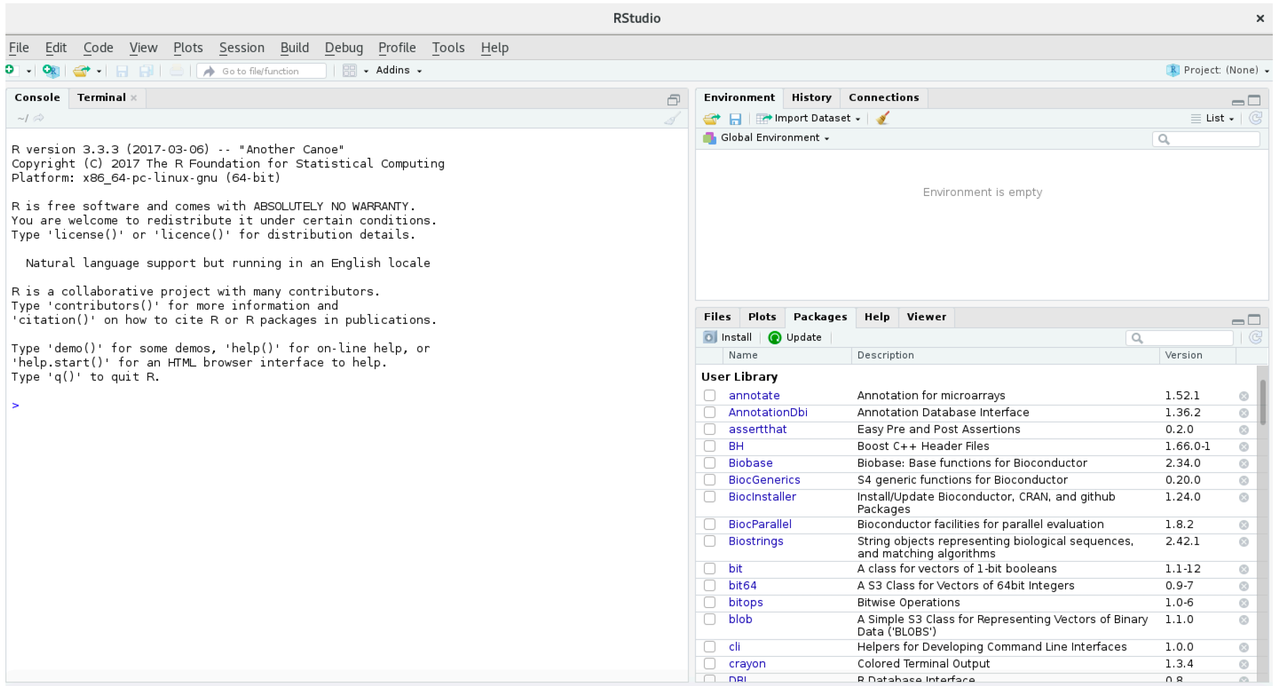
General Introduction - R and RStudio

- Let’s open RStudio



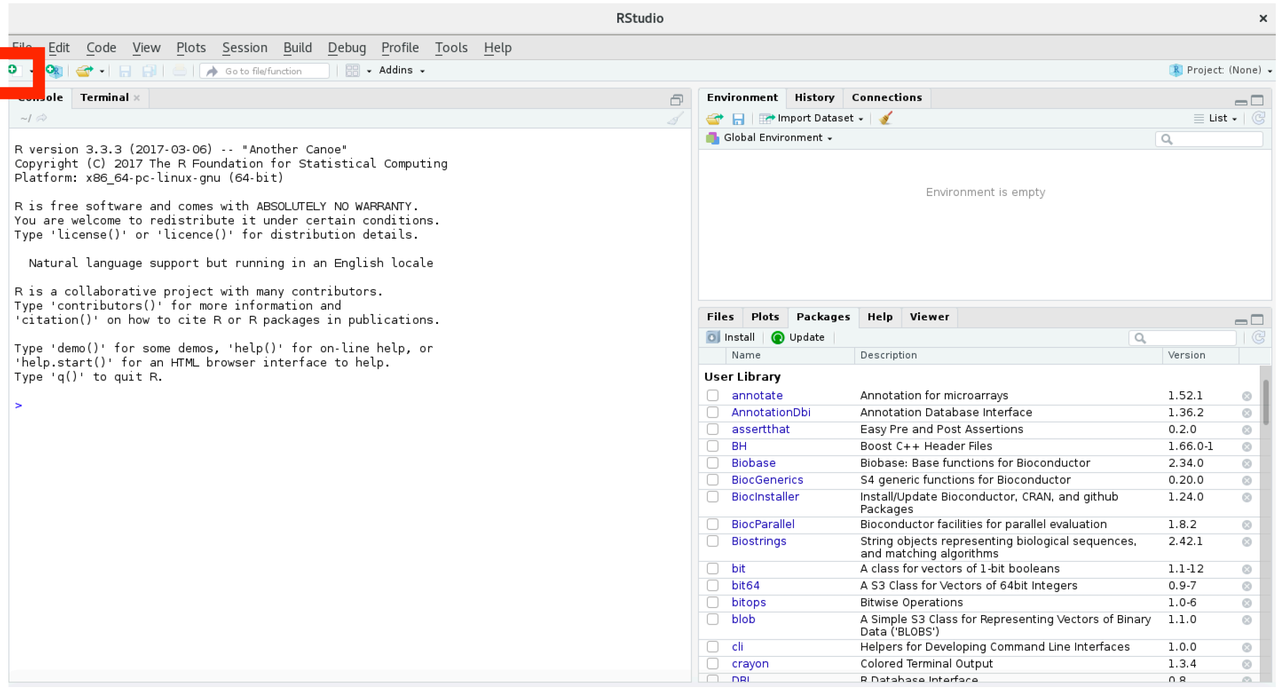
General Introduction - R and RStudio

- Getting oriented in RStudio



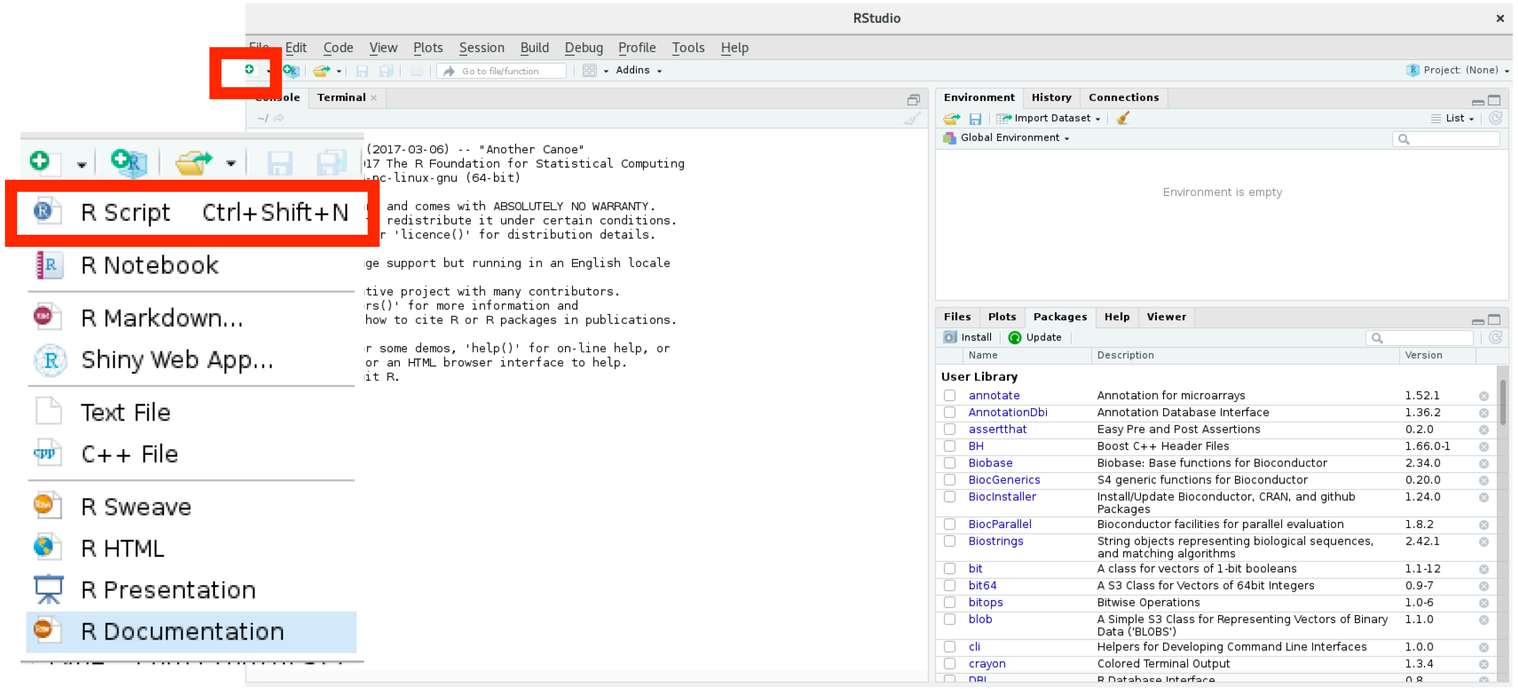
General Introduction - R and RStudio

- Getting oriented in RStudio



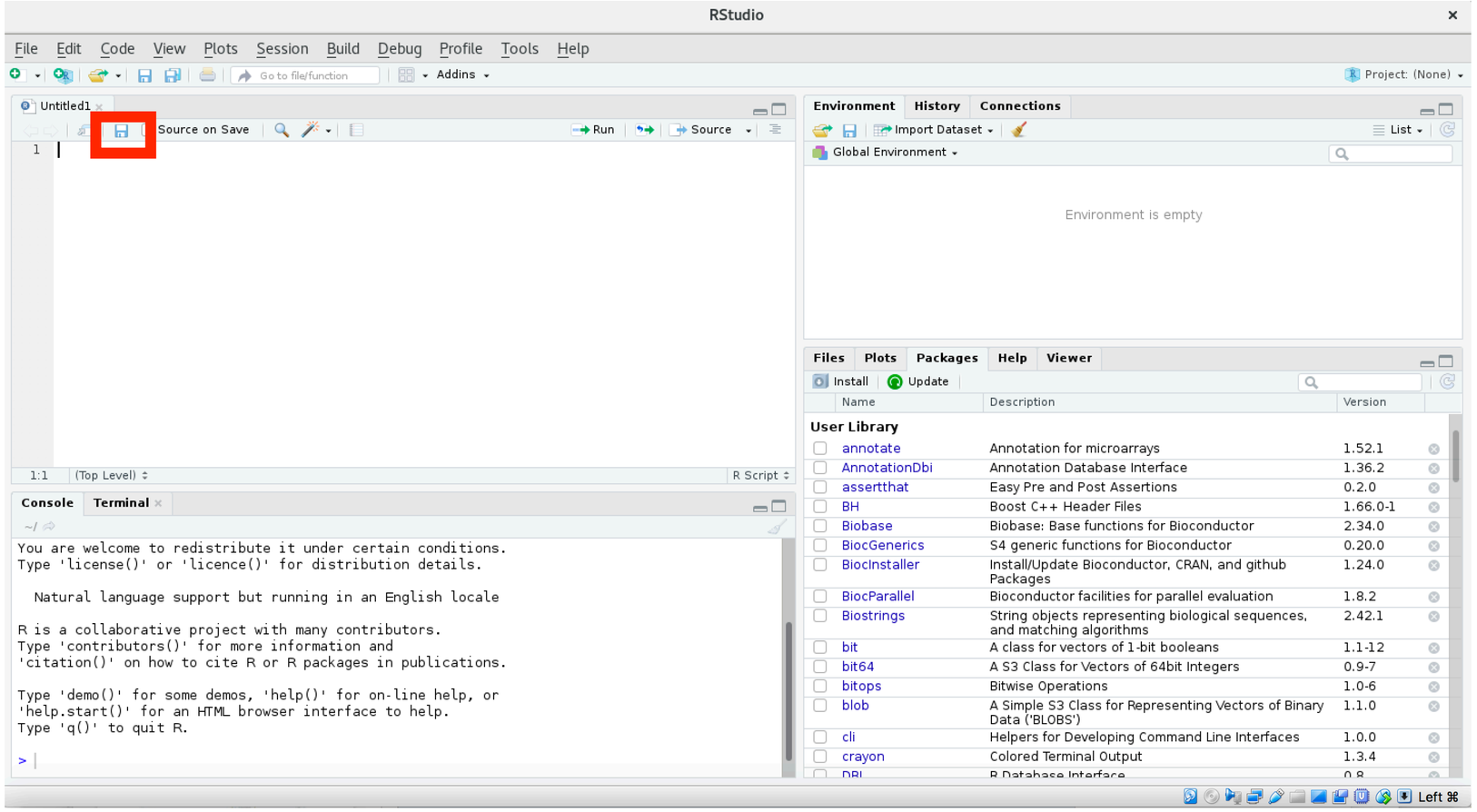
General Introduction - R and RStudio

- Getting oriented in RStudio



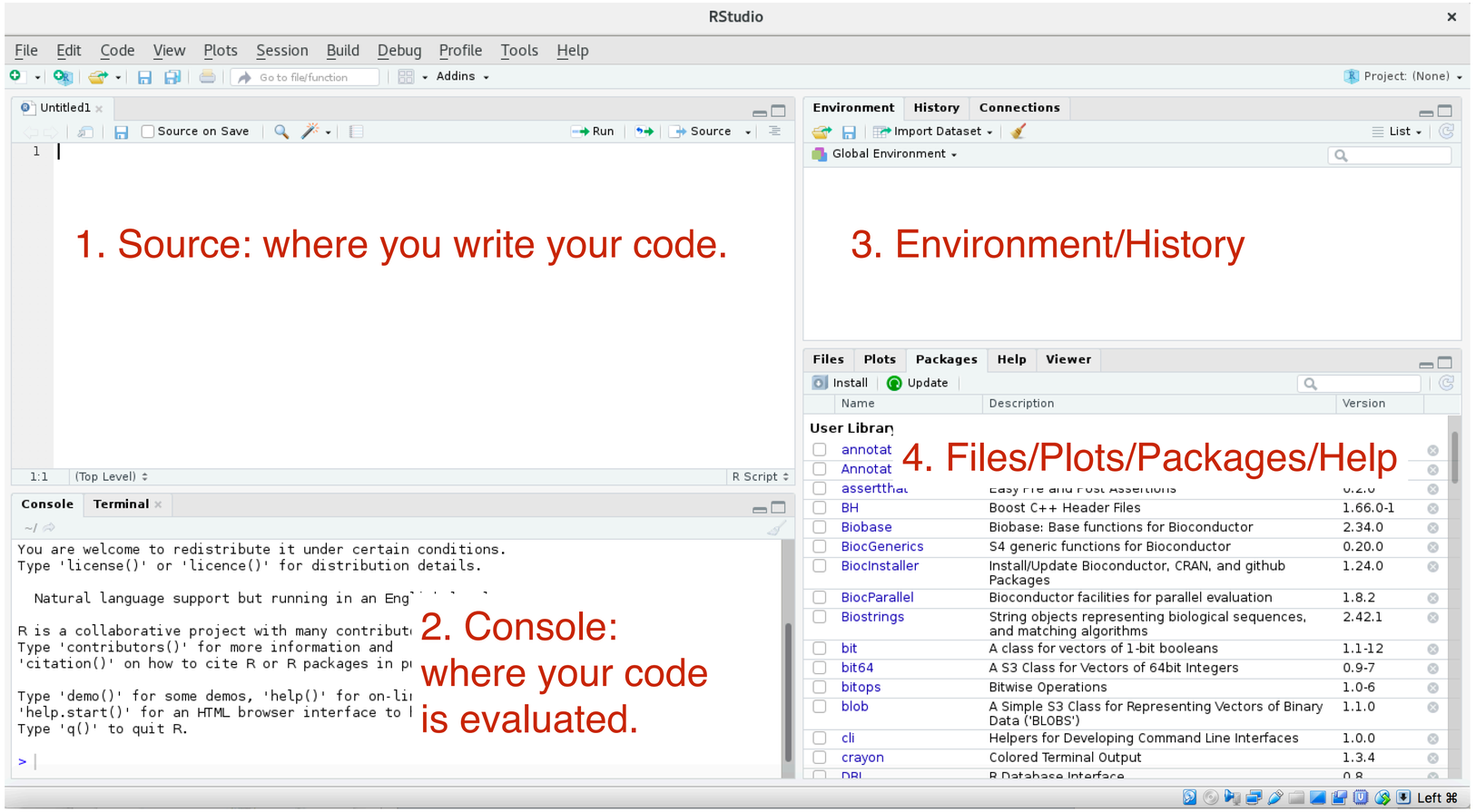
General Introduction - R and RStudio

- Getting oriented in RStudio



General Introduction - R and RStudio

- Getting oriented in RStudio



Location & Working Directory

To see your working directory (similar to pwd in Terminal):

```
getwd()
```

To set your working directory (similar to cd in Terminal):

```
setwd("./")
```

To list what is in the directory (similar to ls in Terminal):

```
dir()
```

BCBC

BTI

Simple Operations and Calculations Using R

We can do basic operations in R:

Addition or subtraction:

```
56+29
```

Multiplication:

```
3*12
```

Division:

```
40/8
```

Exponents:

```
2^12
```

Try to calculate the following:

```
( (8 / 4) - 4.8 + (12 / (2+1) ) )^3
```

Simple Operations and Calculations Using R

We can do basic operations in R:

Addition or subtraction:

```
56+29
```

Multiplication:

```
3*12
```

Division:

```
40/8
```

Exponents:

```
2^12
```

Try to calculate the following:

```
( (8 / 4) - 4.8 + (12 / (2+1) ) )^3
```

1.728

BCBC

BTI

Scalar Variables & Data Types

Variables can be used to "store" values:

```
number.of.total.progeny.peas <- 16
ratio.wrinkled <- 0.75

number.of.wrinkled.progeny.peas <- number.of.total.progeny.peas * ratio.wrinkled
number.of.wrinkled.progeny.peas
```

Scalar Variables & Data Types

Variables can be used to "store" values:

```
number.of.total.progeny.peas <- 16
ratio.wrinkled <- 0.75

number.of.wrinkled.progeny.peas <- number.of.total.progeny.peas * ratio.wrinkled
number.of.wrinkled.progeny.peas
```

Scalar Variables & Data Types

Variables can be used to "store" values:

```
number.of.total.progeny.peas <- 16
ratio.wrinkled <- 0.75

number.of.wrinkled.progeny.peas <- number.of.total.progeny.peas * ratio.wrinkled
number.of.wrinkled.progeny.peas
```

Now, how can we calculate the number of round progeny peas using our variables, assuming there are only either round or wrinkled peas?

Scalar Variables & Data Types

Variables can be used to "store" values:

```
number.of.total.progeny.peas <- 16
ratio.wrinkled <- 0.75

number.of.wrinkled.progeny.peas <- number.of.total.progeny.peas * ratio.wrinkled
number.of.wrinkled.progeny.peas
```

Now, how can we calculate the number of round progeny peas using our variables, assuming there are only either round or wrinkled peas?

```
number.of.round.progeny.peas <- number.of.total.progeny.peas * (1 - ratio.wrinkled)

number.of.round.progeny.peas
```

BCBC

BTI

Scalar Variables & Data Types

- A few tips
  - Choose meaningful variable names
  - Separate words using dots or underscores (no spaces allowed)
  - Variable names are case-sensitive

Scalar Variables & Data Types

We can use different data types in our variables. To check the type of a variable, use class():

Numeric

```
my_lucky_number <- 7

class(my_lucky_number)

class(number.of.round.progeny.peas)
```

Character

```
cool_fruit <- "tomato"

class(cool_fruit)
```

BCBC

BTI

Scalar Variables & Data Types

Logical

```
my_name_is_Adrian <- TRUE

class(my_name_is_Adrian)
```

Scalar Variables & Data Types

Logical

```
my_name_is_Adrian <- TRUE

class(my_name_is_Adrian)
```

What would class() tell you for the following?

```
tomato_is_the_best_fruit <- "TRUE"

class(tomato_is_the_best_fruit)
```

BCBC

BTI

Functions

- A function is basically just a set of instructions
  - Takes some input
  - Does some things
  - Gives an output

- We can access functions in a few ways…

Functions

- We can define our own custom functions

Let's define and use a simple function:

```
my_awesome_function <- function(input1){
  output <- input1^12 * 5 + 3
  return(output)
}

my_awesome_function(3)

my_awesome_function(5)

my_awesome_new_variable <- my_awesome_function(7)

my_awesome_new_variable
```

BCBC

BTI

Functions

- R also has ‘pre-made’ functions… can you think of any examples that we’ve used?

Functions

- R also has ‘pre-made’ functions… can you think of any examples that we’ve used?

To access R documentation for a function, etc.:

```
?class

?plot
```

Functions

- We can also install packages and load libraries to access sets of additional functions…

Packages

- Let’s try installing/loading a package called ggplot2:

```
install.packages("ggplot2")

library(ggplot2)
```

Data Structures

- Vectors
- Matrices
- Data frames
- Lists

Data Structures: Vectors

```
vector1 <- c(1,2,5.3,6,-2,4)
vector1
```

```
## [1]  1.0  2.0  5.3  6.0 -2.0  4.0
```

```
vector2 <- c("one","two","three")
vector2
```

```
## [1] "one"   "two"   "three"
```

```
vector3 <- c(TRUE,TRUE,TRUE,FALSE,TRUE,FALSE)
vector3
```

```
## [1]  TRUE  TRUE  TRUE FALSE  TRUE FALSE
```

```
vector4 <- rep(1:3, each=2)
vector4
```

```
## [1] 1 1 2 2 3 3
```

Data Structures: Vectors

```
vector5 <- rep(c(1, 0, 7), times = 5) #repeats c(1,0,7) vector 5 times
vector5
```

```
##  [1] 1 0 7 1 0 7 1 0 7 1 0 7 1 0 7
```

```
vector6 <- rep(c(0, 4), times = c(5,3)) # you could tell R how many times to repeat each value.
vector6
```

```
## [1] 0 0 0 0 0 4 4 4
```

```
vector7 <- rep(1:4,length.out=6) # you could also repeat the vector to the specifield length even if th
vector7
```

```
## [1] 1 2 3 4 1 2
```

```
vector8 <- seq(from = 7.5, to = 2.5, by = -0.5) # vector with a numbers 7.5 to 2.5 in steps of 0.5.
vector8
```

```
##  [1] 7.5 7.0 6.5 6.0 5.5 5.0 4.5 4.0 3.5 3.0 2.5
```

BCBC

BTI

Data Structures: Vectors

We can transpose a vector:

```
t(vector1)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    1    2  5.3    6   -2    4
```

We can access elements in a vector using the element's vector position:

```
vector1[c(2,4)]
```

```
## [1] 2 6
```

We can add, subtract, multiply, divide, square root, and exponentiate vectors and scalars of numeric class. These can be done using operations:

```
vector1
```

```
## [1]  1.0  2.0  5.3  6.0 -2.0  4.0
```

```
vector7
```

```
## [1] 1 2 3 4 1 2
```

```
vector1 + vector7
```

```
## [1]  2.0  4.0  8.3 10.0 -1.0  6.0
```

Data Structures: Vectors

We can apply functions on vectors:

```
mean(vector1)
```

```
## [1] 2.716667
```

```
median(vector1)
```

```
## [1] 3
```

Data Structures: Matrices

We can create matrix using matrix(c(), byrow=, ncol=). Matrices can mix element class types.

```
matrix1 <- matrix(c(1,3,2,2,8,9), ncol=2)
matrix1
```

```
##      [,1] [,2]
## [1,]    1    2
## [2,]    3    8
## [3,]    2    9
```

```
matrix2 <- matrix(c(1,3,2,2,8,9), ncol=2, byrow = T)
matrix2
```

```
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    2
## [3,]    8    9
```

Data Structures: Matrices

We can access elements in matrix using element position:

```
c(matrix1[1,2], matrix1[2,2],matrix1[3,1])
```

```
## [1] 2 8 2
```

Matrix operations (matrix multiplication, etc.) can also be performed.

Transpose of matrices (It is basically to interchange of rows and columns:

```
t(matrix1)
```

```
##      [,1] [,2] [,3]
## [1,]    1    3    2
## [2,]    2    8    9
```

We can also create vectors from a matrix:

```
matrix1[2, ]
```

```
## [1] 3 8
```

```
matrix1[ ,2]
```

```
## [1] 2 8 9
```

BTI

Data Structures: Data Frames

Data frames are used for storing data tables. It is usually a list of equal vectors of the same length.

```
c <- c(2, 3, 5)
d <- c("aa", "bb", "cc")
e <- c(TRUE, FALSE, TRUE)
df <- data.frame(c, d, e)
colnames(df) <- c("numeric", "character", "logical") # Add column names
df
```

```
##   numeric character logical
## 1       2        aa    TRUE
## 2       3        bb   FALSE
## 3       5        cc    TRUE
```

BCBC

BTI

Reading Data into R

- Use functions such as:
  - read.table()
  - read.csv()

R Package Repos

- CRAN

```
> install.packages("fortunes")
```

- Bioconductor

```
source("https://bioconductor.org/biocLite.R")
biocLite("S4Vectors")
```

- GitHub

```
library(githubinstall)
githubinstall("AnomalyDetection")
```

When would you use bash vs R vs Python?

- Designing a data analysis pipeline that runs several commands and programs
- Writing a more sophisticated program to deploy a machine learning on a large data set
- Performing statistical analysis on a large data set

When would you use bash vs R vs Python?

- Bash
  - Simple, great performance wise, good for piping output from one command into another command
  - Gets complicated with larger projects, not very elegant
- Python
  - Is more general purpose and can be used beyond data analysis, elegant
  - Not as many libraries as R
- R
  - Great for visualization (plotting) and for statistical analysis
  - Can be slow, code not as elegant, can be more difficult to learn
- Also depends on what you are comfortable with, some people prefer R, some prefer Python.

- All scripts from today will be added to GitHub here (in addition to on Canvas):

https://github.com/bcbc-group/PLSCi7202

Assignment for next time

- Write an R or Python script that performs a task that will be useful for your research. If relevant, you can automate it with a bash shell script. Feel free to discuss with others!

- Please complete your script(s) before next class. We will use these in our next class to demonstrate best practices.