# Lecture 10 - Basic Programming

## DSE 511

Drew Schmidt
2022-09-27

# Announcements

- Nothing unresolved from last time
- Homework is live!
- Questions?

# Content

- Introduction to R and Python
- Basic Math
- Control Flow
- Functions
- Classes and Objects

# Introduction to R and Python

# R and Python

## R

- *Lingua franca* for statistical computing
- Part programming language, part data analysis package
- Dialect of S (May 5, 1976, Bell Labs)
- Notable names: Ross Ihaka, Robert Gentleman, **John Chambers**
- Free software (GPL >= 2)

## Python

- General purpose programming language
- Created late 80's / early 90's
- Notable names: Guido Van Rossum
- Reference to Monty Python's Flying Circus
- Python Software Foundation License

# Similarities

- Interactive languages
- Can be used in batch (not considered the default)
- Extensible via add-on packages
- Mostly similar syntax

# Important Differences

## R

- Indexing is 1-based
- Whitespace not semantic; use braces for multi-line blocks
- Assignment often done via `<-`
- Copyleft licensed
- Data science package with programming language tacked on

## Python

- Indexing is 0-based
- Whitespace is semantic; use indentation for multi-line blocks
- Assignment done via `=`
- Non-copyleft licensed
- Programming language with data science package tacked on

# Popular Editors/IDE's

## R

- RStudio

## Universal

- Jupyter
- VSCode, Atom
- vim
- ...

## Python

- Pycharm

# Core Data Science Packages

## R

- Base R
- data.table
- dplyr
- ggplot2
- caret

## Python

- Pandas
- Numpy
- Scipy
- Matplotlib
- scikit-learn

# "Killer Apps"

## R

- Tidyverse
- Rmarkdown
- Shiny

## Python

- Numba
- Tensorflow
- JAX

# Installing Packages

## R

- CRAN
- BioConductor
- ...

```
install.packages("data.table")
```

## Python

- PyPI
- pip
- conda

```
pip install pandas
```

# Help

## R

```
help(print)
?print
??print
```

## Python

```
help(print)
```

# Assignment

## R

```r
x <- 1
y <<- 2
z = 3
4 -> w
```

## Python

```python
x = 1
```

# Variable Naming

## R

- Technically you can do (almost) anything - no really
- You should probably stick to reasonable conventions
- Certain keywords restricted (e.g. `function`)

```
rm(list=ls())
` 1 2 3 a b c` = 1
ls()
```

```
## [1] " 1 2 3 a b c"
```

## Python

- Start with a letter or underscore
- Letters, numbers, understores
- Certain keywords restricted (e.g. `class`)

# Basic Math

# Basic Math

## R

```
3 + 2
```

```
## [1] 5
```

```
3 - 2
```

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

```
3 * 2
```

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

```
3 / 2
```

```
## [1] 1.5
```

## Python

```
3 + 2
```

```
## 5
```

```
3 - 2
```

```
## 1
```

```
3 * 2
```

```
## 6
```

```
3 / 2
```

```
## 1.5
```

# Basic Math

## R

```
3 ^ 2
```

```
## [1] 9
```

```
3 %% 2
```

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

## Python

```
3 ** 2
```

```
## 9
```

```
3 % 2
```

```
## 1
```

# Basic Math

## R

```
abs(-1)
```

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

```
round(1.2345, 3)
```

```
## [1] 1.234
```

```
max(c(1, 2, 3))
```

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

## Python

```
abs(-1)
```

```
## 1
```

```
round(1.2345, 3)
```

```
## 1.234
```

```
max([1, 2, 3])
```

```
## 3
```

# Basic Math

## R

```
sqrt(2)
```

```
## [1] 1.414214
```

```
exp(1)
```

```
## [1] 2.718282
```

```
sin(pi)
```

```
## [1] 1.224647e-16
```

## Python

```
import math
math.sqrt(2)
```

```
## 1.4142135623730951
```

```
math.exp(1)
```

```
## 2.718281828459045
```

```
math.sin(math.pi)
```

```
## 1.2246467991473532e-16
```

# Basic Math

## R

```
dnorm(.75)
```

```
## [1] 0.3011374
```

```
pnorm(.75)
```

```
## [1] 0.7733726
```

```
qnorm(.75)
```

```
## [1] 0.6744898
```

## Python

```
import scipy.stats
scipy.stats.norm.pdf(.75)
```

```
## 0.30113743215480443
```

```
scipy.stats.norm.cdf(.75)
```

```
## 0.7733726476231317
```

```
scipy.stats.norm.ppf(.75)
```

```
## 0.6744897501960817
```

# Control Flow

# Control Flow

- Loops
  - `for`
  - `while`
- Conditionals
  - `if`
  - `else if` / `elif`
  - `else`
- Transfer
  - `break`
  - `next` / `continue`

# General Rules

## R

```
keyword (conditions) {
  body
}
```

## Python

```
keyword conditions :
  body
```

# For Loops

## R

```r
X = 1:5
for (x in X) {
  print(x)
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
```

## Python

```python
X = range(1, 5+1)
for x in X:
  print(x)
```

```
## 1
## 2
## 3
## 4
## 5
```

# While Loops

## R

```r
x = 1
while (x <= 5) {
  print(x)
  x = x + 1
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
```

## Python

```python
x = 1
while x <= 5:
  print(x)
  x += 1
```

```
## 1
## 2
## 3
## 4
## 5
```

# Conditionals

## R

```
x = 1
if (x < 5) {
  print("x is less than 5")
} else {
  print("x is not less than 5")
}
```

```
## [1] "x is less than 5"
```

## Python

```
x = 1
if x < 5:
  print("x is less than 5")
else:
  print("x is not less than 5")
```

```
## x is less than 5
```

# Transfer

## R

```r
set.seed(1234)
for (i in 1:10) {
  if (runif(1) < 0.5) {
    break
  }
}
print(i)
```

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

## Python

```python
import random
random.seed(1234)
for i in range(1, 10+1):
  if random.uniform(0, 1) < 0.5:
    break

print(i)
```

```
## 2
```

# Function Programming

Both languages have support for FP

## R

```
X = 1:3
sapply(X, sqrt)
```

```
## [1] 1.000000 1.414214 1.732051
```

## Python

```
import math
X = range(1, 3+1)
list(map(math.sqrt, X))
```

```
## [1.0, 1.4142135623730951, 1.7320508075688772]
```

# Functions

# Functions

- An encapsulated block of code
- Mandatory for proper code organization
- Well-named functions can often mitigate need for comments

```
x = read_data("myfile.csv")
x_processed = process(x)
mdl = fit_model(x_processed)
```

# Function Syntax

## R

```r
f = function(x) {
  return(x+1)
}
f(1)
```

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

## Python - Not Optional

```python
def f(x):
  return x+1

print(f(1))
```

```
## 2
```

# Named Returns

## R - Optional

```r
f = function(x) {
  x+1
}
f(1)
```

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

## Python - Not Optional

```python
def f(x):
  x+1

print(f(1))
```

```
## None
```

# Scoping

## R

```r
x = 1
f = function() {
  return(x+1)
}
f()
```

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

## Python

```python
x = 1
def f():
  return x+1

print(f())
```

```
## 2
```

# Copy Semantics

## R

```r
f = function(x) {
  x = x + 1
  return(x)
}
x = 1
f(x)
```

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

```r
print(x)
```

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

## Python

```python
def f(x):
  x += 1
  return x

x = 1
f(x)
```

```
## 2
```

```python
print(x)
```

```
## 1
```

# Classes and Objects

# Classes and Objects

- Object Oriented Programming (OOP)
  - Useful organizational strategy
  - Not the only paradigm
- A *class* is a custom data type
  - Describes the data
  - Defines the methods
- It describes an *object*, or instantiation of the class
- The methods operate on the objects
- Methods may use side-effects

## Pseudocode

```
x = create_class_object()
x.compute()
y = x.get_result()
```

# Why OOP?

## What is "x"?

```
x+1
write_to_disk(x)
compute_svd(x)
```

# OOP Systems

## R

- S3
- S4
- R6 (CRAN package)

## Python

- The `class` system

# Classes -- R6

```
Timer = R6::R6Class(
  public = list(
    start = function() {
      private$t0 = proc.time()
      invisible(self)
    },
    stop = function() {
      runtime = (proc.time() - private$t0)
      print(runtime)
    }
  ),
  private = list(
    t0 = NULL
  )
)
```

```
t = Timer$new()

t$start()
Sys.sleep(.63)
t$stop()
```

```
## [1] 0.633
```

# Classes -- Python

```
import time

class Timer(object):
  def start(self):
    self.t0 = time.perf_counter()

  def stop(self):
    t1 = time.perf_counter()
    print(t1-self.t0)
```

```
t = Timer()

t.start()
time.sleep(.63)
t.stop()
```

```
## 0.6461440641433001
```

# Wrapup

# Wrapup

- In most ways, these languages are very similar.
  - Basic math
  - Loops
  - Functions
  - Even classes!
- Differences materialize at the extremes
  - HPC
  - Neural networks
  - (Very) modern statistics

# Questions?