Week 8: *Python in R*

EMSE 4571: Intro to Programming for Analytics

John Paul Helveston

March 03, 2022

Quiz 5

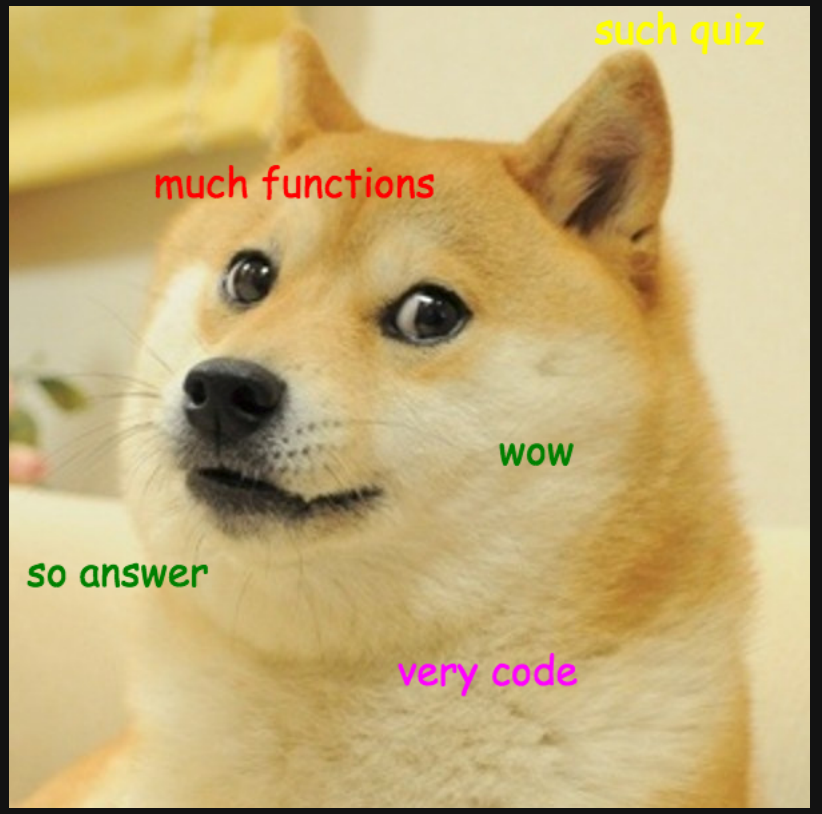
05:00

Go to `#class` channel in Slack for quiz link

Open RStudio first!

Rules:

- You may use your notes and RStudio
- You may **not** use any other resources (e.g. the internet, your classmates, etc.)



R tip of the week:

`styler`

Install `styler` package

```
install.packages("styler")
```

Go to Addins menu, search for "style", select "Style active file"

Week 8: *Python in R*

1. Getting started
2. Python basics
3. Functions & methods
4. Loops & lists

BREAK

5. Strings

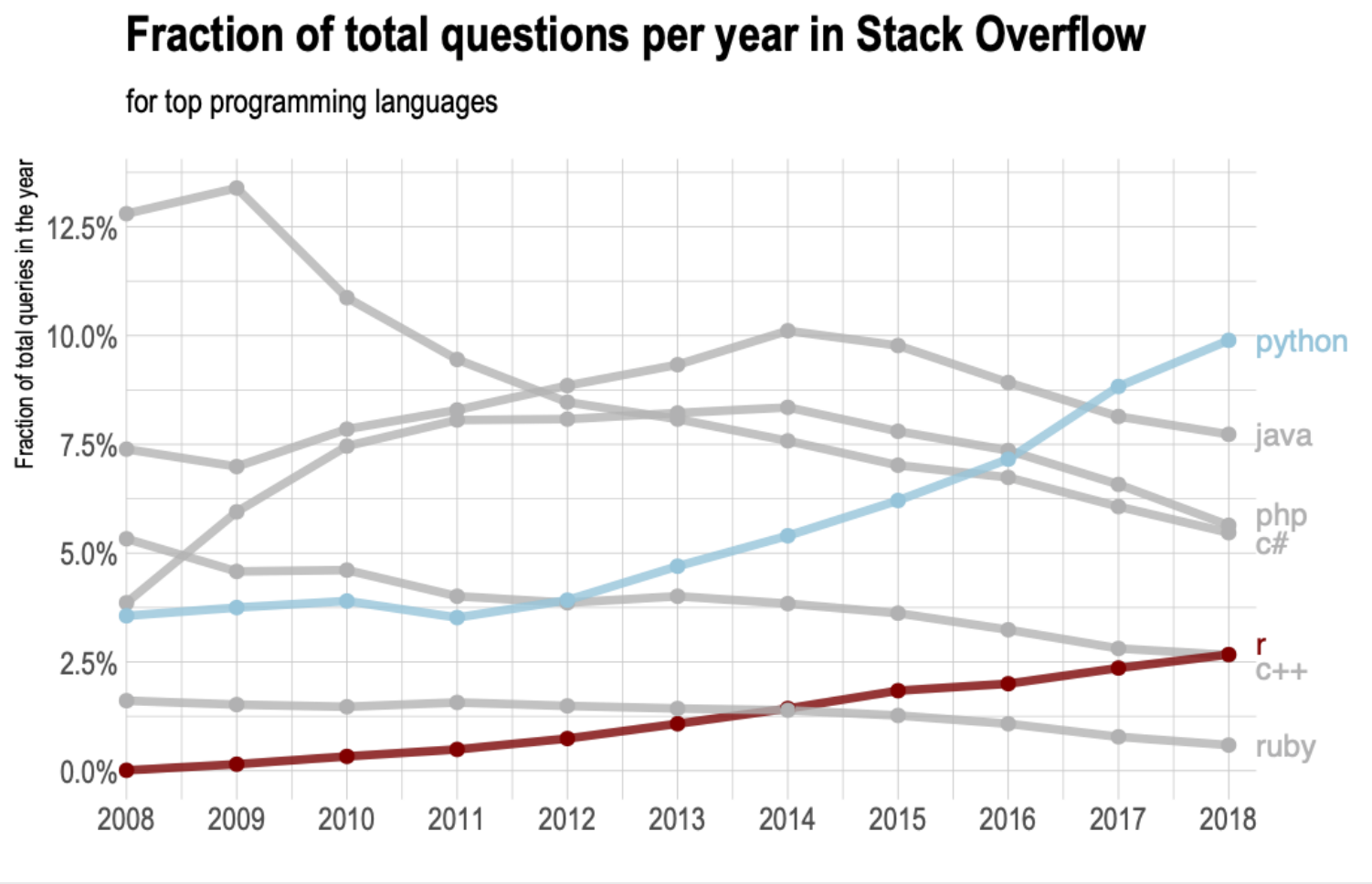
Week 8: *Python in R*

1. Getting started
2. Python basics
3. Functions & methods
4. Loops & lists

BREAK

5. Strings

Why Python?


Fraction of total questions per year in Stack Overflow
for top programming languages
Fraction of total queries in the year
12.5%
10.0%
7.5%
5.0%
2.5%
0.0%
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
python
java
php
c#
r
c++
ruby


A
B
DATA ANALYSIS

Install the `reticulate` library

```
install.packages("reticulate")
```

(Only do this once)

Load the `reticulate` library

```
library(reticulate)
```

(Do this every time you use the package)

Do you have Python on your computer?

If note, you may see the following message pop up:

```
Would you like to install Miniconda? [Y/n]:
```

My recommendation: type `y` and press `enter`

Starting Python

Open a Python REPL ("**R**ead–**E**val–**P**rint-**L**oop"):

```
repl_python()
```

You should see the `>>>` symbol in the console. This means you're now using Python!

(Remember, the R console has only one `>` symbol).

You want to use Python 3, not Python 2

Above the `>>>` symbols, it should say `"Python 3..."`

Exiting Python (but we just got started?)

If you want to get back to good 'ol R, just type the command `exit` into the Python console:

```
exit
```

(Note that you type `exit` and not `exit()` with parentheses).

Open a Python script

File --> New File --> Python Script

When you run code from a Python script, R automatically opens a Python REPL

Week 8: *Python in R*

1. Getting started
2. Python basics
3. Functions & methods
4. Loops & lists

BREAK

5. Strings

Operators

Arithmetic operators

Operator	R	Python
Integer division	`%/%`	`//`
Modulus	`%%`	`%`
Powers	`^`	`**`

Logical operators

Operator	R	Python
And	`&`	`and`; `&`
Or	`|`	`or`; `|`
Not	`!`	`not`; `!`

You can do this in Python:

```
(3 == 3) and (4 == 4)
```

```
#> True
```

Variable assignment

Python only uses the = symbol to make assignments (no <–):

```
value = 3
value
```

```
#> 3
```

Data types

Same data types as R, but with more "Computer Science-y" names:

Description	R	Python
numeric (w/decimal)	`double`	`float`
integer	`integer`	`int`
character	`character`	`str`
logical	`logical`	`bool`

Data types

Three important distinctions:

Data type	R	Python
Logical	`TRUE` or `FALSE`	`True` or `False`
Numbers	`double` by default	`int` by default (unless has decimal)
Nothing	`NULL`	`None`

Get type

R: `typeof()`

```
typeof(3.14)
```

```
#> [1] "double"
```

```
typeof(3L)
```

```
#> [1] "integer"
```

```
typeof("3")
```

```
#> [1] "character"
```

```
typeof(TRUE)
```

```
#> [1] "logical"
```

Python: `type()`

```
type(3.14)
```

```
#> <class 'float'>
```

```
type(3)
```

```
#> <class 'int'>
```

```
type("3")
```

```
#> <class 'str'>
```

```
type(True)
```

```
#> <class 'bool'>
```

Check type

R: `is.__()`

```r
is.double(3.14)
```

```
#> [1] TRUE
```

```r
is.integer(3L)
```

```
#> [1] TRUE
```

```r
is.character("3")
```

```
#> [1] TRUE
```

```r
is.logical(TRUE)
```

```
#> [1] TRUE
```

Python: `type() == type`

```python
type(3.14) == float
```

```
#> True
```

```python
type(3) == int
```

```
#> True
```

```python
type("3") == str
```

```
#> True
```

```python
type(True) == bool
```

```
#> True
```

Convert type

R: `as.__()`

```r
as.double("3")
```

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

```r
as.integer(3.14)
```

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

```r
as.character(3.14)
```

```
#> [1] "3.14"
```

```r
as.logical(3.14)
```

```
#> [1] TRUE
```

Python: `__()`

```python
float("3")
```

```
#> 3.0
```

```python
int(3.14)
```

```
#> 3
```

```python
str(3.14)
```

```
#> '3.14'
```

```python
bool(3.14)
```

```
#> True
```

Quick practice

02:00

Write Python code to do the following:

1. Create an object `x` that stores the value `"123"`
2. Create an object `y` that is `x` converted to an integer
3. Write code to confirm that `y` is indeed an integer
4. Write a logical statement to determine if `y` is odd or even

Week 8: *Python in R*

1. Getting started
2. Python basics
3. Functions & methods
4. Loops & lists

BREAK

5. Strings

Python and R have many similar functions

R

```
abs(-1)
```

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

```
round(3.14)
```

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

```
round(3.14, 1)
```

```
#> [1] 3.1
```

Python

```
abs(-1)
```

```
#> 1
```

```
round(3.14)
```

```
#> 3
```

```
round(3.14, 1)
```

```
#> 3.1
```

Writing functions

R

```r
isEven <- function(n) {
    if (n %% 2 == 0) {
        return(TRUE)
    }
    return(FALSE)
}
```

Python

```python
def isEven(n):
    if (n % 2 == 0):
        return(True)
    return(False)
```

Note:

- Functions start with `def`
- Use `:` and indentation instead of `{}`
- **Indentation is precisely 4 spaces!**

Writing test functions

R

```r
test_isEven <- function() {
    cat("Testing isEven(n)...")
    stopifnot(isEven(2) == TRUE)
    stopifnot(isEven(1) == FALSE)
    cat("Passed!")
}
```

Python

```python
def test_isEven():
    print("Testing isEven(n)...")
    assert(isEven(2) == True)
    assert(isEven(1) == False)
    print("Passed!")
```

Note:

- Use `print()` instead of `cat()`
- Use `assert()` instead of `stopifnot()`

Python Methods

Python objects have "methods" - special functions that *belong* to certain object classes.

R

Use `str_to_upper()` function

```r
s <- "foo"
stringr::str_to_upper(s)
```

```
#> [1] "FOO"
```

Python

Use `upper()` *method*

```python
s = "foo"
s.upper()
```

```
#> 'FOO'
```

Python Methods

See all the available methods with `dir` function:

```
s = "foo"
dir(s)
```

```
#> ['__add__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__',
'__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__', '__getnewargs__',
'__gt__', '__hash__', '__init__', '__init_subclass__', '__iter__', '__le__', '__len__',
'__lt__', '__mod__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__',
'__repr__', '__rmod__', '__rmul__', '__setattr__', '__sizeof__', '__str__',
'__subclasshook__', 'capitalize', 'casefold', 'center', 'count', 'encode', 'endswith',
'expandtabs', 'find', 'format', 'format_map', 'index', 'isalnum', 'isalpha',
'isdecimal', 'isdigit', 'isidentifier', 'islower', 'isnumeric', 'isprintable',
'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip', 'maketrans',
'partition', 'replace', 'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip',
'split', 'splitlines', 'startswith', 'strip', 'swapcase', 'title', 'translate', 'upper',
'zfill']
```

R-Python magic

R-Python magic

You can source a Python script from R, then use the Python function in R!

Inside your `notes-blank.py` file, you have the following function defined:

```
def isEven(n):
    if (n % 2 == 0):
        return(True)
    return(False)
```

Open your `notes.R` file and *source* the `notes-blank.py` file:

```
reticulate::source_python('notes-blank.py')
```

Magically, the function `isEven(n)` now works inside R!

Your turn

15:00

Write the following two functions in Python code:

1. `hypotenuse(a, b)`: Returns the hypotenuse of the two lines of length `a` and `b`.
2. `isRightTriangle(a, b, c)`: Returns `True` if the triangle formed by the lines of length `a`, `b`, and `c` is a right triangle and `False` otherwise. **Hint**: you may not know which value (`a`, `b`, or `c`) is the hypotenuse.

Week 8: *Python in R*

1. Getting started
2. Python basics
3. Functions & methods
4. Loops & lists

BREAK

5. Strings

`for` loops

R

```r
for (i in seq(1, 5, 2)) {
    cat(i, '\n')
}
```

```
#> 1
#> 3
#> 5
```

Python

```python
for i in range(1, 5, 2):
    print(i)
```

```
#> 1
#> 3
```

Notes:

- `range()` leaves out stopping number
- No `()` in `for` loop line

`while` loops

R

```r
i <- 1
while (i <= 5) {
    print(i)
    i <- i + 2
}
```

```
#> [1] 1
#> [1] 3
#> [1] 5
```

Python

```python
i = 1
while i <= 5:
    print(i)
    i += 2
```

```
#> 1
#> 3
#> 5
```

Notes:

- Could also use `i = i + 2` to increment
- No `()` in `while` loop line

Python lists

These are **not** the same as R vectors! (They're equivalent to R lists)

Universal list creator: `[]`

```
[1, 2, 3]
```

```
#> [1, 2, 3]
```

Lists can store different types

```
[1, "foo", True]
```

```
#> [1, 'foo', True]
```

Adding and removing items

Add items with `list.append()`

```
x = [1, 2, 3]
x.append(7)
x
```

```
#> [1, 2, 3, 7]
```

Remove items with `list.remove()`

```
x = [1, 2, 3]
x.remove(3)
x
```

```
#> [1, 2]
```

Note: You don't have to overright `a`,
i.e. Don't do this: `x = x.append(7)`

Sorting lists

```
x = [1, 5, 3]
```

Sorting that returns a new object

```
sorted(x)
```

```
#> [1, 3, 5]
```

```
sorted(x, reverse = True)
```

```
#> [5, 3, 1]
```

```
x
```

```
#> [1, 5, 3]
```

Sort the object `x` *without* creating a new object

```
x.sort()
x
```

```
#> [1, 3, 5]
```

Slicing lists with `[ ]`

```
x = ['A', 'list', 'of', 'words']
```

Indices start at 0:

```
x[0] # Returns the first element
```

```
#> 'A'
```

```
x[3] # Returns the third element
```

```
#> 'words'
```

```
x[len(x)-1] # Returns the last element
```

```
#> 'words'
```

Slicing with a vector of indices:

```
x[0:3]  # Returns the first 3 elements
```

```
#> ['A', 'list', 'of']
```

Negative indices slice from the end

```
x = ['A', 'list', 'of', 'words']
```

Indices start at 0:

```
x[-1] # Returns the last element
```

```
#> 'words'
```

```
x[-2] # Returns 2nd-to-last element
```

```
#> 'of'
```

```
x[-len(x)] # Returns first element
```

```
#> 'A'
```

Slicing with a vector of indices:

```
x[-3:-1]  # Returns middle 2 elements
```

```
#> ['list', 'of']
```

Note on 0 indexing

```
x = ["A", "B", "C", "D", "E"]
```

List items sit *between* fence posts.

```
index: 0     1     2     3     4
       |     |     |     |     |     |
item:  | "A" | "B" | "C" | "D" | "E" |
       |     |     |     |     |     |
```

You slice at the *fence post* number to get elements *between* the posts.

```
x[0:1]
```

```
#> ['A']
```

```
x[0:3]
```

```
#> ['A', 'B', 'C']
```

Your turn

15:00

Write the following two functions in Python code:

1. `factorial(n)`: Returns the factorial of `n`, e.g. `3! = 3*2*1 = 6`. Note that `0` is a special case, and `0! = 1`. Assume `n >= 0`.

2. `nthHighestValue(n, x)`: Returns the nth highest value in a list of numbers. For example, if `x = [5, 1, 3]`, then `nthHighestValue(1, x)` should return `5`, because `5` is the 1st highest value in `x`, and `nthHighestValue(2, x)` should return `3` because it's the 2nd highest value in `x`. Assume that `n <= len(x)`.

Break

05:00

Week 8: *Python in R*

1. Getting started
2. Python basics
3. Functions & methods
4. Loops & lists

BREAK

5. Strings

Doing "math" with strings

Concatenation:

R

```r
paste("foo", "bar", sep = "")
```

```
#> [1] "foobar"
```

Python

```python
"foo" + "bar"
```

```
#> 'foobar'
```

Repetition:

R

```r
str_dup("foo", 3)
```

```
#> [1] "foofoofoo"
```

Python

```python
"foo" * 3
```

```
#> 'foofoofoo'
```

Using word commands with strings

Sub-string detection:

R

```r
str_detect('Apple', 'App')
```

```
#> [1] TRUE
```

Python

```python
'App' in 'Apple'
```

```
#> True
```

Most string manipulation is done with *methods*

R

```
str_function(s)
```

Python

```
s.method()
```

Case conversion

R

```r
s <- "A longer string"
str_to_upper(s)
```

```
#> [1] "A LONGER STRING"
```

```r
str_to_lower(s)
```

```
#> [1] "a longer string"
```

```r
str_to_title(s)
```

```
#> [1] "A Longer String"
```

Python

```python
s = "A longer string"
s.upper()
```

```
#> 'A LONGER STRING'
```

```python
s.lower()
```

```
#> 'a longer string'
```

```python
s.title()
```

```
#> 'A Longer String'
```

Trimming white space

R

```
s <- "     A string with space     "
str_trim(s)
```

```
#> [1] "A string with space"
```

Python

```
s = "     A string with space     "
s.strip()
```

```
#> 'A string with space'
```

Replacing strings

R

```
s <- "Hello world"
str_replace(s, "o", "a")
```

```
#> [1] "Hella world"
```

```
str_replace_all(s, "o", "a")
```

```
#> [1] "Hella warld"
```

Python

```
s = "Hello world"
s.replace("o", "a")
```

```
#> 'Hella warld'
```

Merge a vector / list of strings together

R

```r
s <- c("Hello", "world")
paste(s, collapse = "")
```

```
#> [1] "Helloworld"
```

Python

```python
s = ["Hello", "world"]
"".join(s)
```

```
#> 'Helloworld'
```

Python has some super handy string methods

Detect if string contains only numbers:

R

R doesn't have a function for this...
here's one way to do it:

```r
s <- "42"
! is.na(as.numeric(s))
```

```
#> [1] TRUE
```

Python

```python
s = "42"
s.isnumeric()
```

```
#> True
```

Getting sub-strings with `[]`

R

```r
s <- "Apple"
str_sub(s, 1, 3)
```

```
#> [1] "App"
```

Python

```python
s = "Apple"
s[0:3]
```

```
#> 'App'
```

Notes:

- Indexing is the same as lists

Getting sub-string indices

R

```r
s <- "Apple"
str_locate(s, "pp")
```

```
#>      start end
#> [1,]     2   3
```

Python

```python
s = "Apple"
s.index("pp")
```

```
#> 1
```

Note:

- Only returns the starting index

String splitting

Both languages return a list:

R

```r
s <- "Apple"
str_split(s, "pp")
```

```
#> [[1]]
#> [1] "A"  "le"
```

Python

```python
s = "Apple"
s.split("pp")
```

```
#> ['A', 'le']
```

Python can only split individual strings

R can split vectors of strings

```r
s <- c("Apple", "Snapple")
str_split(s, "pp")
```

```
#> [[1]]
#> [1] "A"  "le"
#>
#> [[2]]
#> [1] "Sna" "le"
```

Python

```python
s = ["Apple", "Snapple"]
s.split("pp")
```

```
#> Error in py_call_impl(callable,
dots$args, dots$keywords): AttributeError:
'list' object has no attribute 'split'
```

Need **numpy** package for this in Python

```
import numpy as np

s = np.array(["Apple", "Snapple"])
np.char.split(s, "pp")
```

```
#> array([list(['A', 'le']), list(['Sna', 'le'])], dtype=object)
```

You'll need to install **numpy** to use this:

```
py_install("numpy")
```

Your turn

15:00

Write the following two functions in Python code:

1. `sortString(s)`: Takes a string `s` and returns back an alphabetically sorted string. **Hint**: Use `list(s)` to break a string into a list of letters.

- `sortString("cba") == "abc"`
- `sortString("abedhg") == "abdegh"`
- `sortString("AbacBc") == "ABabcc"`

1. `areAnagrams(s1, s2)`: Takes two strings, `s1` and `s2`, and returns `True` if the strings are anagrams, and `False` otherwise. **Treat lower and upper case as the same letters**.

- `areAnagrams("", "") == True`
- `areAnagrams("aabbccdd", "bbccddee") == False`
- `areAnagrams("TomMarvoloRiddle", "IAmLordVoldemort") == True`

HW 8

I suggest starting with `reticulate::repl_python()` to work in Python from RStudio.

- Submit your "hw8.py" file to the autograder - it will (hopefully) work