Python in R Markdown Files

Andrew Lau

12/01/2019

Python in R and R Markdown Files

What a world we live in... Now, not only can we call Python from within R, we can pass data back and forth between R and Python and knit Python up into an R Markdown file. We can have our cake and eat it too! (Check out https://rstudio.github.io/reticulate/index.html for more information about the reticulate package)

Here we focus on how to call Python from an R Markdown file.

Step 1: Get everything ready

Before we begin:

- Have your Python version of choice installed
- Update R Studio. To be able to pass data structures between R and Python it's best to have 1.2.1206 or later installed. Note that as at December 2018, version 1.2.1206 is still in "preview release".
- Install/update the R reticulate package then load the library

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

Step 2 (optional): Point to the correct version of Python

You may have multiple versions of Python installed (if you're a Mac user you will have a version of Python 2 installed by default as the OS uses it for some of its programs). To find out how many Pythons are slithering inside your machine enter "which -a python python3" into your terminal (Mac OS) and this will give you a list of the paths of all the versions of Python on your machine.

In the below R chunk, I tell R to use my Python 3 intepreter, import the sys Python module and find out what version of Python R is using. Make sure you run this R chunk before you do anything else.

```
use_python("/usr/local/bin/python3", required = T)
sys <- import("sys")
sys$version
```

```
## [1] "3.7.1 (v3.7.1:260ec2c36a, Oct 20 2018, 03:13:28) \n[Clang 6.0 (clang-600.0.57)]"
```

Step 3: Calling Python

Importing Python modules within an R chunk
We can import a Python module *inside an R chunk* and use the $ operator to access functions, constants and classes from the module (e.g. instead of math.pi in Python we can use math$pi to access the pi constant from the Python math module *inside an R chunk*)

```
# Python module in an R chunk
print("We are calling Python from R in an R Chunk!")
```

```
## [1] "We are calling Python from R in an R Chunk!"
```

```
math <- import("math")
math$pi
```

```
## [1] 3.141593
```

```
math$sqrt(2)
```

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

Calling Python using a Python chunk

We can also insert a Python chunk into our R Markdown file using (Insert > Python) or typing:
“‘{python}“‘
Python code here
“‘

```
# Python in a Python chunk
import random
print("We are calling Python from R in a Python Chunk!")
```

```
## We are calling Python from R in a Python Chunk!
```

```
def my_func():
    my_num = random.randrange(1,10)
    return my_num

for i in range(10):
    print("Number " + str(i) + " is " + str(my_func()))
```

```
## Number 0 is 2
## Number 1 is 5
## Number 2 is 5
## Number 3 is 5
## Number 4 is 4
## Number 5 is 9
## Number 6 is 3
## Number 7 is 6
## Number 8 is 2
## Number 9 is 7
```

Calling a Python script with a GUI

I’m testing Python out in R with a simple memory game I made that calls uses a GUI (I wanted to see how R would respond to Python opening up a window with a GUI). It works a treat!

```
# implementation of card game - Memory
# import libraries
try:
    import simplegui
except ImportError:
    import SimpleGUICS2Pygame.simpleguics2pygame as simplegui
```

```
## pygame 1.9.4
## Hello from the pygame community. https://www.pygame.org/contribute.html
```

```
import random
# globals
HEIGHT = 100
WIDTH = 800
```

```python
state = 0
turn_counter = 0
# helper function to initialize globals
def new_game():
    global cards, state, paired, exposed, turn_counter
    cards = list(range(0, 8)) + list(range(0, 8))
    print(cards)
    random.shuffle(cards)
    print(cards)
    state = 0
    exposed = [False] * 16
    paired = [False] * 16
    print(exposed)
    turn_counter = 0
    label.set_text("Turns = " + str(turn_counter))
# define event handlers
def mouseclick(pos):
    """ returns the card index number based on the position of the mouse click """
    global selected_card, state, card1, card2, turn_counter
    # just base it on the x-coordinates
    selected_card = pos[0] // (WIDTH // 16)
    print(str(selected_card) + " selected")
    # exposed[selected_card] = not exposed[selected_card]
    if not exposed[selected_card]:
        if state == 0:
            print("state is " + str(state))
            card1 = selected_card
            print("card1 is " + str(card1))
            exposed[card1] = True
            state = 1
            print("state is 1")
        elif state == 1:
            card2 = selected_card
            print("card2 is " + str(card2))
            exposed[card2] = True
            state = 2
            print("state is 2")
        else:
            if cards[card1] == cards[card2]:
                print("card1 and card2 form a pair")
                card1 = selected_card
                exposed[card1] = True
                print("card1 is " + str(card1))
            else:
                exposed[card1], exposed[card2] = False, False
                card1 = selected_card
                exposed[card1] = True
                print("card1 is " + str(card1))
            state = 1
            turn_counter += 1
            print("state is 1")
            print("turns - " + str(turn_counter))
            label.set_text("Turns = " + str(turn_counter))
```

```python
# cards are logically 50x100 pixels in size
def draw(canvas):
    for i in range(len(cards)):
        if exposed[i]:
            canvas.draw_text(str(cards[i]), (WIDTH / 16 * (i + 0.3), HEIGHT * 0.575), 50, "white")
        elif not exposed[i]:
            p1 = [WIDTH // 16 * i, 0]
            # print(p1)
            p2 = [WIDTH // 16 * (i + 1), 0]
            # print(p2)
            p3 = [WIDTH // 16 * (i + 1), HEIGHT]
            # print(p3)
            p4 = [WIDTH // 16 * i, HEIGHT]
            # print(p4)
            canvas.draw_polygon([p1, p2, p3, p4], 6, "red", "green")
# create frame and add a button and labels
frame = simplegui.create_frame("Memory", WIDTH, HEIGHT)
frame.add_button("Reset", new_game)
label = frame.add_label("Turns = " + str(turn_counter))
# register event handlers
frame.set_mouseclick_handler(mouseclick)
frame.set_draw_handler(draw)
# get things rolling
new_game()
```

```
## [0, 1, 2, 3, 4, 5, 6, 7, 0, 1, 2, 3, 4, 5, 6, 7]
## [1, 2, 7, 4, 6, 0, 2, 3, 4, 0, 1, 6, 3, 7, 5, 5]
## [False, False, False, False, False, False, False, False, False, False, False, False, False, False, Fa
```

```python
frame.start()
```