Lecture 2: Python Programming Paradigms for R Novices

Programming in Psychological Science: Python

Thomas Pronk

Programming Paradigms

- Imperative (Structured) programming
  - If, loop, sub-routines
- Functional programming
  - Expressions, first-class functions
- Array programming
  - c(3, 4) + 1 -> c(4, 5)

Did you know… All these paradigms can be expressed as Turing Machines

Materials

- This presentation
- Three assignments to hand in
- Truckloads of support materials on the web

This is a nice basic tutorial:
http://www.djmannion.net/psych_programming

Imperative Programming

Using Statements to Change a Program State

Imperative Concepts

1. Importing packages
2. Variables (state)
3. Flow control (if, loop, function calls)
4. Blocks

This is what most people see as “programming”

Did you know… There was a time programming was done by punching holes in cards

Imperative Programming in R

```
library(ggplot2)                    # Import package
x <- 1                              # State
while (x <= 4) {                    # Loop
    if (x > 2) {                    # If
        print(mean(x))              # Function call
    }                               # End of if-block
    x <- x + 1                      # Update state
}                                   # End of loop-block
```

Imperative Programming in Python

```
import numpy as np                  # Import package
x = 1                               # State
while x <= 4:                       # Loop
    if x > 2:                       # If
        print(np.mean(x))           # Function call
    x = x + 1                       # Update state

# A block is defined by indenting 4 spaces
```

Basic Data Types in R and Python

R	Python	Properties
numeric	int	An Integer Number (-5, 0, 3)
numeric	float	A Natural Number (3.2, 0., 8.17), a.k.a. “floating-point number”
character	str	String of text (“hey”, “”, “a”)
logical	bool	Boolean (True or False)
	object	In Python, everything is an object in the end

Functional Programming

Treat Computation as the Evaluation of Mathematical Functions

Functional Concepts

- Expressions (many languages)
- First-class functions (modern languages: R, Python, JavaScript)

You can do a lot with functional programming; look for lambda calculus and closures

Expressions in R and Python

Type	R	Python
int & float	arithmetics (+ - * /)	arithmetics (+ - * /)
bool	**boolean logic (&& ||)**	**boolean logic (and or)**
str	**doesn't work**	**some work (try +)**
(all)	comparison (== != > >= < <=)	comparison (== != > >= < <=)

First-class Functions in R

```
# Adds one to number
add_one = function (number) {
    return (number + 1)
}
x = add_one                    # x refers to add_one
print(x(5))                    # Gives 6
numbers = c(1, 2, 3)           # Vector of numbers
# Apply add_one to each element
print(sapply(numbers, add_one))
```

First-class Functions in Python

```
# Adds one to number
def add_one(x):
    return x + 1
x = add_one                    # x refers to add_one
print(x(5))                    # Gives 6
numbers = [1, 2, 3]            # List of numbers
# Apply add_one to each element
print(map(add_one, numbers))
# In Python 3, use: list(map(numbers, add_one)
```

Array Programming

Apply Operations on an Arrays of Values

Array Concepts

1. Single value is Scalar; list of values is Array
2. Apply operations on whole arrays at one

Mainly provided in languages for data analysis:

- R
- Matlab
- Python with NumPy

Array Data Types in R and Python

R	NumPy	Properties
vector, matrix, array	ndarray (numpy)	Array with values of the *same* type; **record array**
data frame	ndarray (numpy)	Values whose type *vary* per column; **structured array**
list	list, dict, tuple (Python)	List of values of *mixed* types

Array Programming in R

```
numbers = c(1, 2, 3)      # Vector of numbers
# Get first element
print(numbers[1])
# Add one to each element
print(numbers + 1)
# Select last two elements
print(numbers[2:3])
# Single element via range; we get a scalar
print(numbers[2:2])
```

Array Programming in Python

```
import numpy as np # Need numpy
# ndarray of numbers
numbers = np.array([1, 2, 3])
# Get first element; note 0
print(numbers[0])
# Add one to each element
print(numbers + 1)
# Get last two elements; note 3
print(numbers[1:3])
# Single element via range; we get an array
print(numbers[1:2])
```

NumPy Caveats

- I’ve had trouble getting structured arrays to work with different data types (one column is a string, another int)
- Work-around; use arrays of **objects**; they work everywhere