

Foundations of Data Science for Biologists

Introduction to Python

BIO 724D

10-JAN-2024

Instructors: Greg Wray and Paul Magwene

Announcements

Please welcome new students who will be joining us

Class schedule for the next two weeks:

Today is our first lecture/lab session (10 Jan)

We will *not* meet tomorrow for a lunch session (11 Jan)

No class on Monday (15 Jan)

We will meet for our first lunch session next Thursday (18 Jan)

After that, the regular pattern: lecture/lab on Mondays, lunch on Thursdays

Changes to assignments and grading:

No more exit tickets on Thursdays; instead, include in the weekly problem set

Earlier feedback on your notebooks and include in the weekly problem set

Why learn another programming language?

Why learn Python?

To expand your computing skills

- Work with non-tabular data (e.g., images, sound, text, results of simulations)

- Carry out modeling and simulations

- Use machine learning and artificial intelligence approaches to process data

- Automate tasks and instruments (e.g., use Arduino or Raspberry Pi to capture data)

- Carry out signal processing

- “Glue” code from multiple languages into a single workflow (bash, C, SQL, R, etc.)

To increase your value to future employers, collaborators, and mentees

What is Python?

A general programming language (GPL)

Python was designed by computer scientists to be as broadly applicable as possible

R is a more specialized language, designed by statisticians for statisticians

Python shares several familiar features with R:

very high-level language (VHLL): abstracted from hardware, highly portable

interpreted language: commands are read one at a time while the program executes

supports multiple programming paradigms: object-oriented, functional, and others

professionally maintained: ongoing bug fixes, feature expansions, library support

supports reproducible research: free, open-source, cross-platform, widely adopted

huge code base: covers a mind-boggling range of applications

extensive learning materials: videos, books, and websites

What makes Python different from R?

A general programming language

- Includes a greater diversity of data structures optimized for different purposes

- Object-oriented programming features are more extensive and especially powerful

- Scalable to enterprise level (used by Facebook, Instagram, YouTube, Reddit, etc.)

An emphasis on readability of code

- Syntax is very clear, more so than just about any other language

- Uses numerous functional programming features to simplify code

An emphasis on ease of writing and debugging code

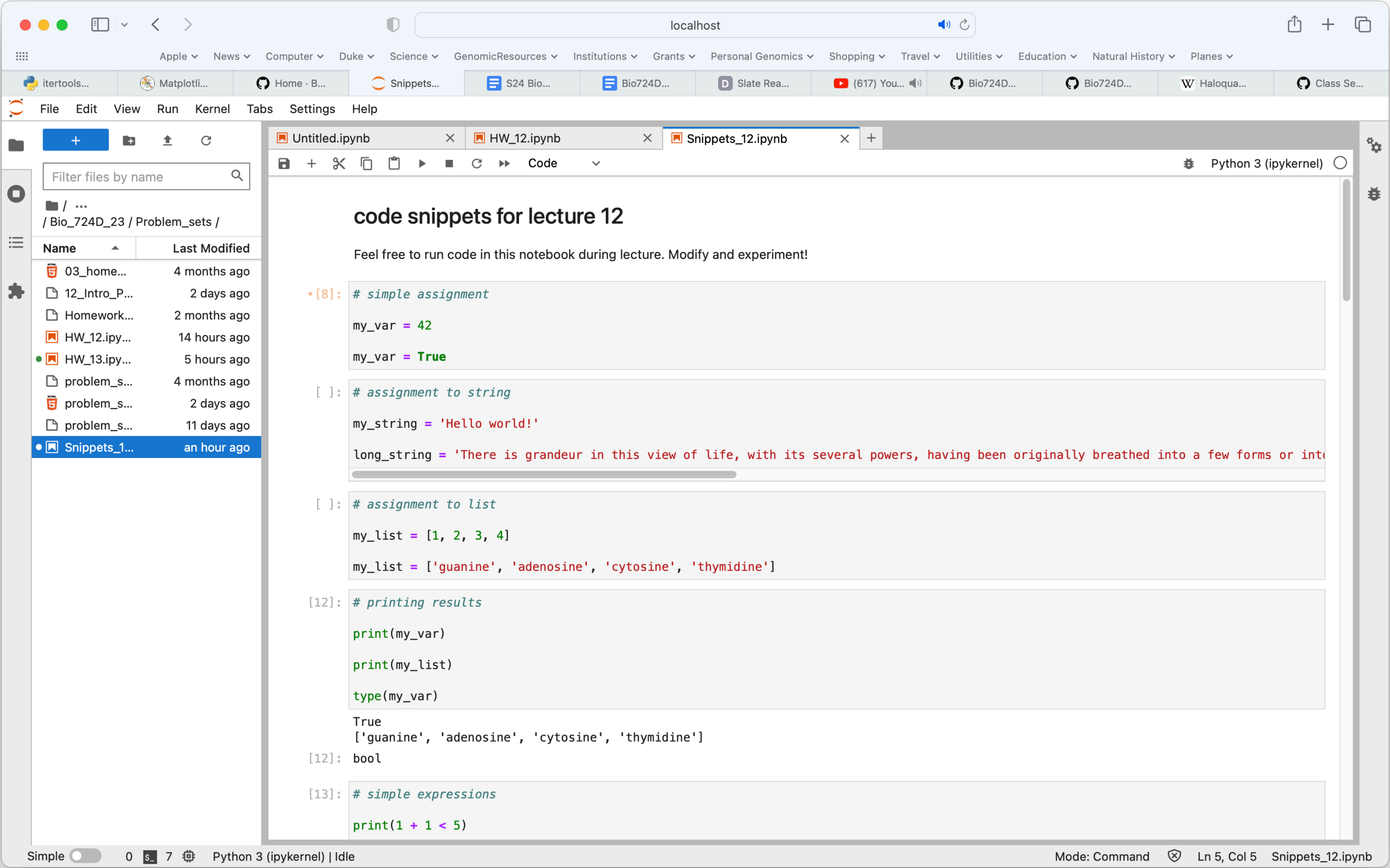
- Much larger set of standard libraries (“batteries included” philosophy)

- Exception handlers and error reporting are powerful and customizable

- Widely considered the best language for collaborative coding

JupyterLab notebooks

The JupyterLab interface



Introduction to Python for R programmers

Starting out with a new language

The good news: your experience with R will help (a lot!)

- Basic programming concepts remain the same

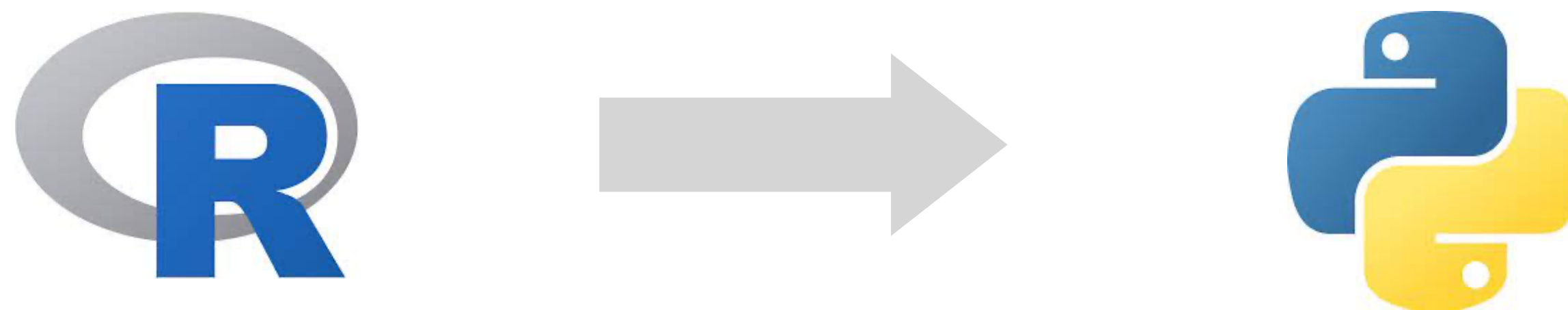
- Python and R share many features that some other languages do not

The challenge: differences between them can create unexpected errors and results

Today's goals:

- Get you working with Python code as quickly as possible

- Minimize the frustration caused by differences between the two languages



Similarities between R and Python

Many of the same **data types** are available: integer, float, boolean, string, list

Assignment and updating data objects is very similar

Square-bracket and name-based **indexing** are both available

Flow control works similarly, including how test conditions are defined

Defining and using basic **functions** is similar

Functional and object-oriented programming **paradigms** are available

However: there are important differences to be aware of in each case!

Assignment and operators

Basic assignment works similarly to R

Data types are inferred from values (*dynamic typing*)

Variable names can be re-assigned to new values / types (*re-binding*)

Data types can be converted where sensible (called *casting* in Python)

Most of the common **operators** work similarly to R

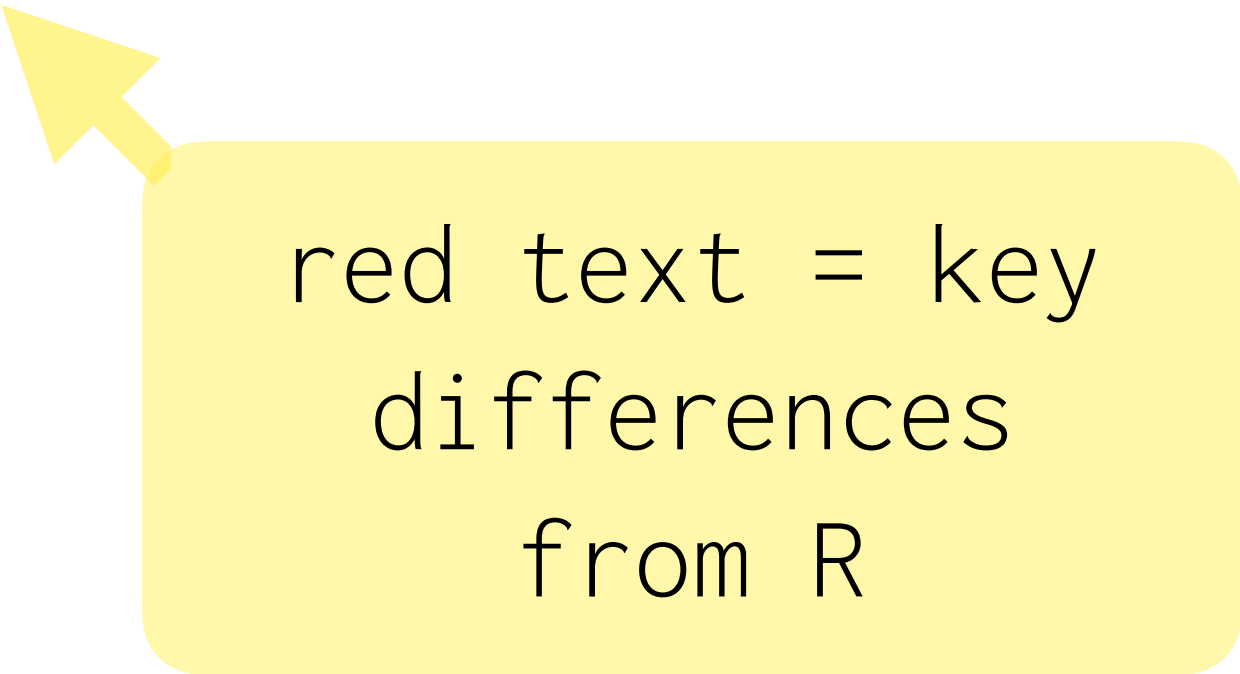
Arithmetic: `+`, `-`, `*`, `/`, `%`, `//` (but use `**` for exponentiation, not `^`)

String: `+` (concatenate)

Comparison: `==`, `!=`, `>`, `<`, `>=`, `<=`

Logical: `and`, `or`, `not`

Membership: `in`, `not in`



red text = key
differences
from R

Six key differences between R and Python

- 1 Larger set of standard data types and data structures
- 2 How data types and data structures are implemented
- 3 How indexing works
- 4 How some functions are implemented
- 5 How code is formatted
- 6 How libraries are used

These differences will cause most of your challenges in transitioning to Python

However: most are straightforward once you understand them

1 Diversity of data types and data structures

The **four common** data types in Python are: `int`, `float`, `str`, `bool`

Similar to R, except that they hold single values, not vectors of values

The **four common** data structures in Python are: `list`, `tuple`, `set`, `dictionary`

`list` is similar to R

The others are containers with no direct parallels in base R or the Tidyverse

Many **additional** standard data types / structures are available

A few commonly used data types are available through **third-party libraries**

NumPy: `ndarray` (n-dimensional arrays built from many different base data types)

Pandas: `Series` (named list) and `DataFrame` (enhanced data frame)

1.1 Identifying standard data types and data structures

Atomic types can be distinguished by their values (similar to R atomic vectors):

integer: number with no decimal (dot)

float: number with a decimal (dot), even if followed by zero or no decimal values

bool: `True`, `False` (no quotes)

string: quotes surrounding characters (including only numerals)

Data structures can be distinguished as follows:

list: item(s) enclosed in square brackets

`['a', 'b', 'c']`

tuple: item(s) enclosed in round brackets

`('a', 'b', 'c')`

set: item(s) enclosed in curly braces

`{'a', 'b', 'c'}`

dictionary: key:value pair(s) in curly braces

`{'a': 1, 'b': 2, 'c': 3 }`

Use `type()` to identify the type/class of any data object

1.2 List and dictionary

These two data structures are common in Python programs

Can hold a mix of any kind of objects and can be nested

Provide many functions

Primary difference is how information is accessed

Lists are indexed by **position**, dictionaries are indexed by **keys**

Keys can be strings or numbers; each key must be unique but values can repeat

my_obj	12	7	23	0	7	-2	4	5	← values
	0	1	2	3	4	5	6	7	← list index (positions)
	'A'	'B'	'C'	'D'	'E'	'F'	'G'	'H'	← dictionary index (keys)

2 Implementation of data types and structures

Differences in how Python implements data types and structures are based on:

- Iterable data objects

- Mutable and immutable data objects

2.1 Iterables in Python

Only containers and strings are iterable

Iterable means that the object can return one value or item at a time

Unlike R, where almost every data type is iterable (because atomics are vectors)

Basic data types (`int`, `float`, `str`, `bool`, etc.) hold single values

Unlike R, where all “atomic” variables are actually vectors (even if of length 1)

Strings contain a single sequence of characters (including whitespaces)

Strings work differently in Python

Indexing and iterating is with reference to individual characters (not items, as in R)

All iterables (except strings) can hold a mix of data types

Unlike R, where only lists can contain items of different types

2.2 Mutable and immutable objects

Mutable data objects can be modified, **immutable** objects cannot

Immutable data objects are useful as “read only” information

Immutable data objects occupy less memory and provide much faster performance

Immutable objects *can* be deleted

Re-binding the name of an immutable object to a different object deletes the data

Mutability only refers to whether data contained in an object can be changed

2.3 Mutable and immutable objects, continued

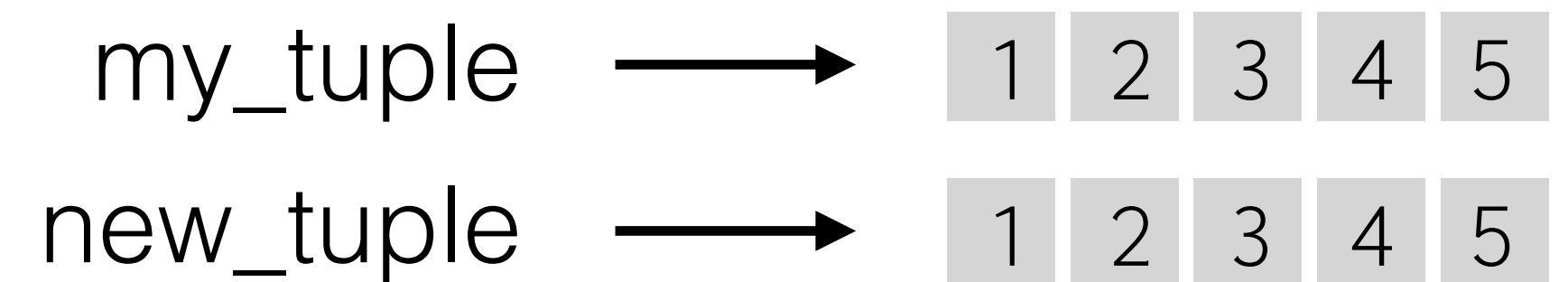
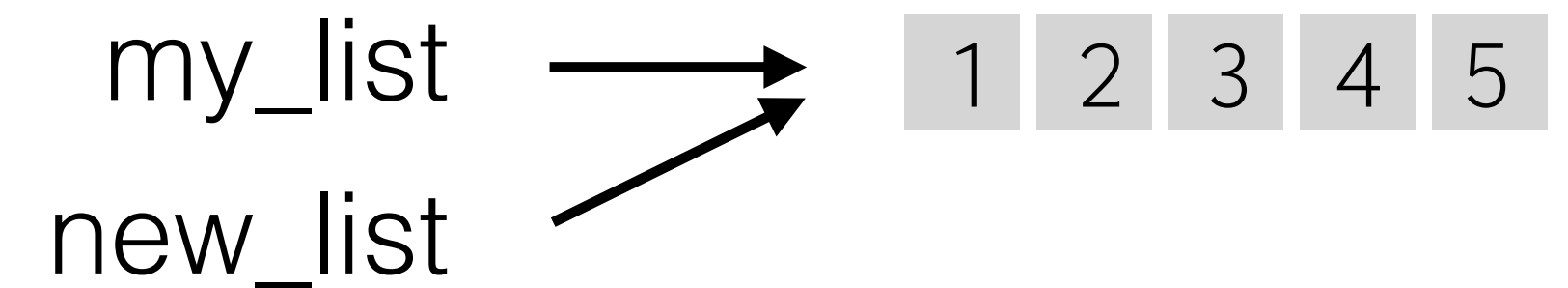
Assigning a mutable data object to a new identifier does not create a copy

This process creates an alias, but points to the same information in memory

Updating either the original or new identifier changes the data for *both*

This is not true for immutable objects; instead, a true copy is created

```
# assign an existing mutable object
my_list = [1, 2, 3, 4, 5]
new_list = my_list
# now try with an immutable object
my_tuple = (1, 2, 3, 4, 5)
new_tuple = my_tuple
```



2.4 Mutable and immutable objects, continued

Many functions work differently when applied to mutable and immutable objects

Mutable objects are typically altered *in place*

The function itself silently returns **None**

Assignment while applying a mutating function will create an undesirable result!

Any function that alters contents cannot be applied to an immutable object

```
# apply sort to a mutable object
my_list = [3, 1, 5, 2, 4]
my_list.sort()
output = my_list.sort()

# now apply sort to an immutable object
my_tuple = (3, 1, 5, 2, 4)
my_tuple.sort()
```

my_list	→	3	1	5	2	4
my_list	→	1	2	3	4	5
output	→	None				
my_tuple	→	3	1	5	2	4
error!						

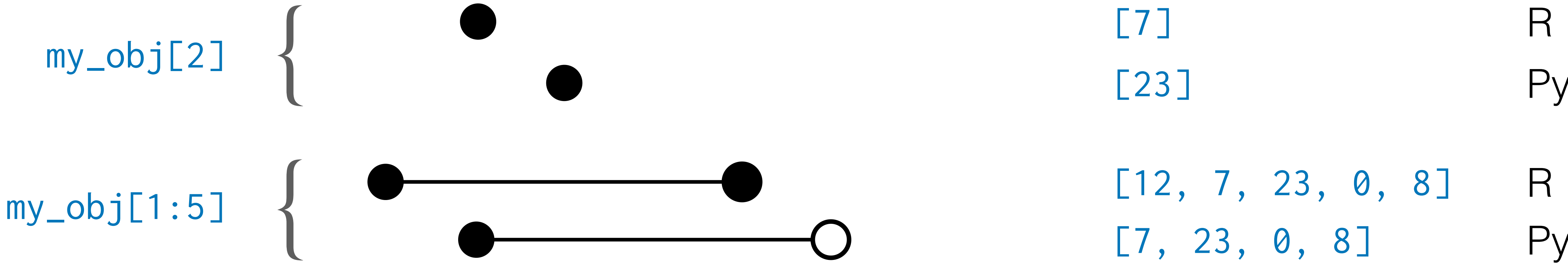
3 Indexing

Square-bracket indexing in Python is different from R in two ways:

- zero-based
- slices do not include the end number

my_obj	12	7	23	0	8	-2	4	5	← values
	1	2	3	4	5	6	7	8	← R index
	0	1	2	3	4	5	6	7	← Python index

Examples:



3.1 Examples of indexing

my_obj	12	7	23	0	8	-2	4	5	← values
	0	1	2	3	4	5	6	7	← index

index	return	notes
my_obj[1]	[7]	single item
my_obj[0:1]	[12]	slice of length 1
my_obj[3:]	[0, 8, -2, 4, 5]	open slice to end
my_obj[:2]	[12, 7]	open slice from start
my_obj[-1]	[5]	last item
my_obj[0:8:3]	[12, 0, 4]	slice with step = 3
my_obj[-1:0:-1]	[5, 4, -2, 8, 0, 23, 7]	slice with reverse step
my_obj[::-1]	[5, 4, -2, 8, 0, 23, 7, 12]	reverse (“Martian smiley”)

4 Functions

Python has two representations of functions

Function: syntax and use are similar to functions in R

Method: function bound to a specific data class; they have a distinct syntax

```
# create a list
my_list = [5, 3, 1, 2, 4]
# call a function
function_result = len(my_list)
print(function_result)
# now call a method
my_list.sort()
print(my_list)
my_string = 'hello world'
my_string.sort()
```

functions take arguments
prints 5

methods are attached to objects
returns [1, 2, 3, 4, 5]

returns an AttributeError

5 Syntax and formatting

Python has a clear syntax

- Expressions often contain fewer characters than other programming languages

- Expressions mimic statements in English as much as possible

- Requires use of whitespace, unlike R and most other languages

Use indenting to indicate code blocks

- Best practice is to use 4 spaces

- Any number of tabs or space is allowed, but must be consistent

- Indent 4 additional spaces to indicate a nested code blocks (no limit to nesting)

Use # to indicate comments (very similar to R)

- # at the beginning of a line makes the entire line a comment

- # in the middle of a line makes everything after it a comment

5.1 Syntax for flow control

Python provides `if`, `for`, and `while` flow control structures

The basic syntax is similar to R, although a few additional options are available

Test conditions are expressed in a similar way

To create a control structure:

Use colon and indenting rather than curly braces to define the code block

Stop indenting to indicate the end of the code block

```
# test for life stage of frogs
if gills == True :
    cohort = 'tadpole'
else :
    cohort = 'adult'
print('Cohort is', cohort)
```

```
# print out names of study group
for x in study_group :
    print('name:', x)
num = len(study_group)
print('total individuals =', num)
```


5.2 Syntax for creating a function

Creating a function in Python uses different syntax from R

Use the `def` keyword rather than calling the `function()` function

Use colon and indenting rather than curly braces to define the code block

Do not use an assignment operator

```
# create a function to test whether 7 is a factor
def is_div_by_7(input_value) :
    if input_value % 7 == 0 : return True
    else: return False
# call the function
result_1 = is_div_by_7(3)
result_2 = is_div_by_7(42)
print(result_1, result_2)                                # returns False True
```

6 Libraries

A vast constellation of useful libraries is available for Python (as for R)

Standard modules: these are built and maintained by the Python Software Organization
Quite extensive and provide many functions and data classes (“batteries included”)
Do not need to be installed: distributed with Python installations and updates

Third-party packages: these are built and maintained by others (like packages in R)
Quality, reliability, and compatibility with updates varies enormously
Some are widely used and extremely reliable (e.g., SciPy, NumPy, Matplotlib, Pandas)
Packages need to be installed before use and manually updated

6.1 Importing libraries

To access a library, you first need to import it within your program:

```
import csv
```

Provides access to all data structures and functions in the `csv` module

Best practice: place import statements at the beginning of a program or module

Multiple libraries can be imported by separating names with commas:

```
import csv, math, time
```

Aliases provide a convenient way to abbreviate names:

```
import numpy as np, pandas as pd
```

6.2 Using functions or data structures in libraries

To use a function in a library, precede with the module name and a period:

```
my_result = factorial(3)          # generates NameError (factorial not defined)
import math                       # factorial() is part of the math module
my_result = factorial(3)          # still generates a NameError
my_result = math.factorial(3)     # correct syntax!
print(my_result)                  # now we have the result: 6
```

This requirement prevents “name collisions” between libraries

Resources for Python

Resources for learning Python

Textbooks

Downey (2015) *Think Python* (2ed). Green Tea Press: [here](#)

McKinney (2022) *Python for Data Analysis* (3ed). O'Reilly: [here](#) (NumPy and pandas)

Tutorials

Software Carpentry: [here](#)

Wikiversity: [here](#)

w3schools: [here](#)

Resources for Python programming

Official Python documentation

The Python Language Reference: [here](#)

The Python Standard Library: [here](#)

General Python programming reference

Martelli et al. (2017) *Python in a Nutshell* (3ed). O'Reilly: [here](#)

Glossaries of Python terminology

Official Python Glossary: [here](#)

Python Lingo from Fluent Python: [here](#)

Python style guide

PEP 8: [here](#)

Example code chunk that does something useful

```
# import libraries
import csv

# open a csv file and read its contents into a list object
my_list = [] # create an empty list
with open ('data.csv', newline='') as f : # open the file for reading
    reader = csv.reader(f) # call the csv.reader() function
    for row in reader : # loop through each line
        my_list.append(row) # add each line to the list

# examine contents of my_list
print(my_list[0:9]) # print first 10 lines
print(len(my_list)) # print the number of lines
```

1.1 Become familiar with the data structures

Learn **when to use** each data structure

- Each data structure has a distinct set of capabilities

- Some functions return specific data structures or require them as arguments

Learn **how to recognize** each data structure

- Each has a unique syntax that makes it instantly identifiable

Two quotes (and an Easter egg) that encapsulate the Python philosophy

"Python is the second-best language for anything. That is an amazing aspiration."

Dan Callahan, PyCon 2018 keynote

"To describe something as 'clever' is not considered a compliment in the Python culture."

Alex Martelli, software engineer at Google, Fellow of the Python Software Foundation

Type `import this` at the Python console to see *The Zen of Python*

Data types available through libraries

Standard libraries are maintained by the Python organization (“batteries included”)

Provide dozens of additional data types, some of which are quite useful

Including: date-times, arrays, collections, enumerators, queues, decimal

Broadly-adopted third-party libraries are important for data science with Python

Math and data science: numpy, pandas (matrices and dataframes)

Data visualization: matplotlib, seaborn (plotting and visualization)

Other third-party libraries contain additional data types but are rarely needed

What is Python?

Most importantly: an outstanding **general programming language** (GPL)

Python was designed by computer scientists to be as broadly useful as possible

R is a specialized language, designed by statisticians for statisticians

A **very high-level language** (VHLL)

Abstracted from hardware, with a focus on logic rather than devices and memory

An **interpreted language**

Commands are read one at a time while the program executes

Easier to program and debug, but less efficient execution and memory use

A language that supports **multiple programming paradigms**

Including: procedural, structural, object-oriented, functional, aspect-oriented, reflective

Why learn Python?

Expand your capabilities: arguably the best GPL currently available

Future-proof your efforts: massive user base, widely adopted, under active development

Extensive standard libraries: developed by professionals (“batteries included” principle)

Adhere to open science / FAIR principles: free, open source, and cross-platform

Ready for serious programming: scalable, great “glue” language, ideal for team coding

Improve your marketability: high demand in industry, medicine, and basic research

Arguably the most popular and widely used programming language*

*ranked #1 in 2023 by: StackScale; TIOBE index; IEEE Spectrum; PYPL; #2 in 2023 in GitHub push/pull requests

A little history

Developed by Guido van Rossem beginning in late 1980s

Named for Monty Python's Flying Circus (classic comedy show of '70s-'80s)

Emphasis on ease of programming, readability, usability, scalability

A strong contender for the most widely used programming language, period

"Python is the second-best language for anything.
That is an amazing aspiration."

Dan Callahan, PyCon 2018 keynote



Guido van Rossem
“Benevolent Dictator for Life”

Built-in data types and data structures

The **eight common** data types/structures in Python are:

Atomic types: `int, float, str, bool` → similar to R

Container structures: `list, tuple, set, dictionary`

You can accomplish a wide range of tasks with just this set of data types / structures

Additional built-in data types are built in but much less commonly used:

Atomic types: `complex, NoneType`

Container structures: `byte, bytearray, frozenset, memoryview`

You will rarely, if ever, need to use any of these

Python provides the ability to create **custom data structures** (known as classes)

Provide enormous flexibility and power

Why learn Python?

Arguably the best GPL currently available

Huge user base: ranked #1 or 2 in data science, widely adopted in scientific computing

Under continuous, active, highly competent development by volunteers

Extensive standard libraries developed in parallel (“batteries included” principle)

Huge base of third-party libraries

Relatively easy to learn, yet seamlessly scalable to serious, massive applications

Ideal for open science / FAIR principles: free, open source, and cross-platform

Arguably the best language for collaborative coding; code relatively easy to maintain

High demand for Python programmers in industrial, biomedical, and basic research