Foundations of Data Science for Biologists

Python programming basics

BIO 724D

2025-MAR-04

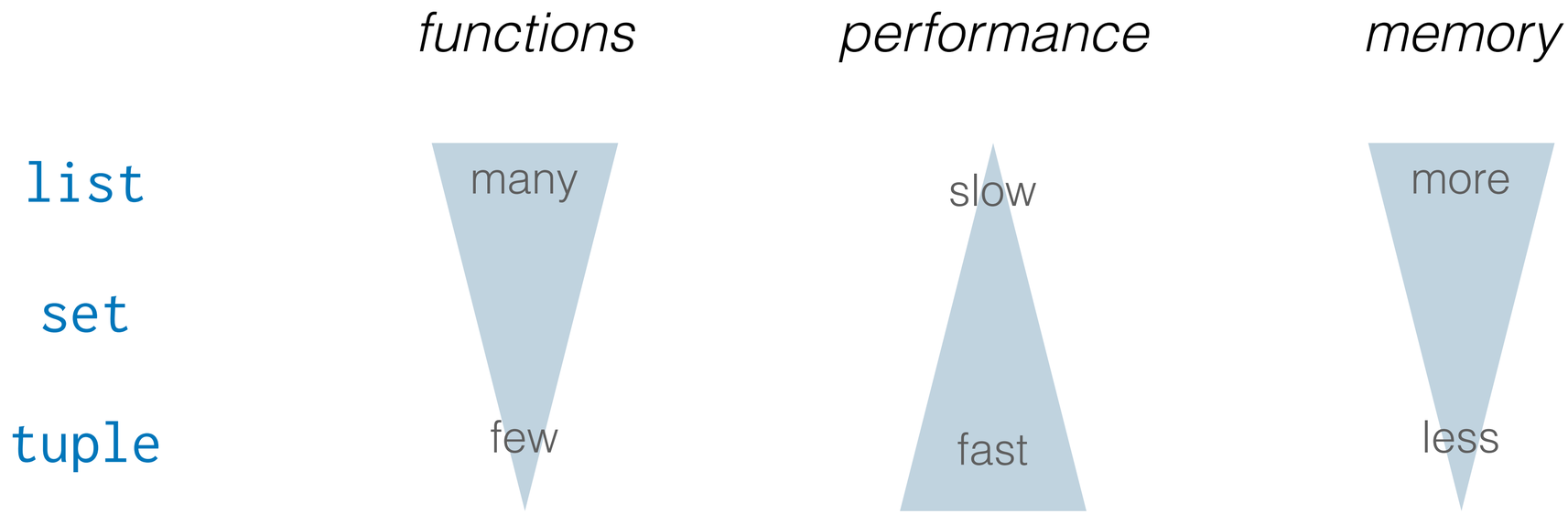
Instructors: Greg Wray and Paul Magwene

Data types and data structures

Why so many different data structures?

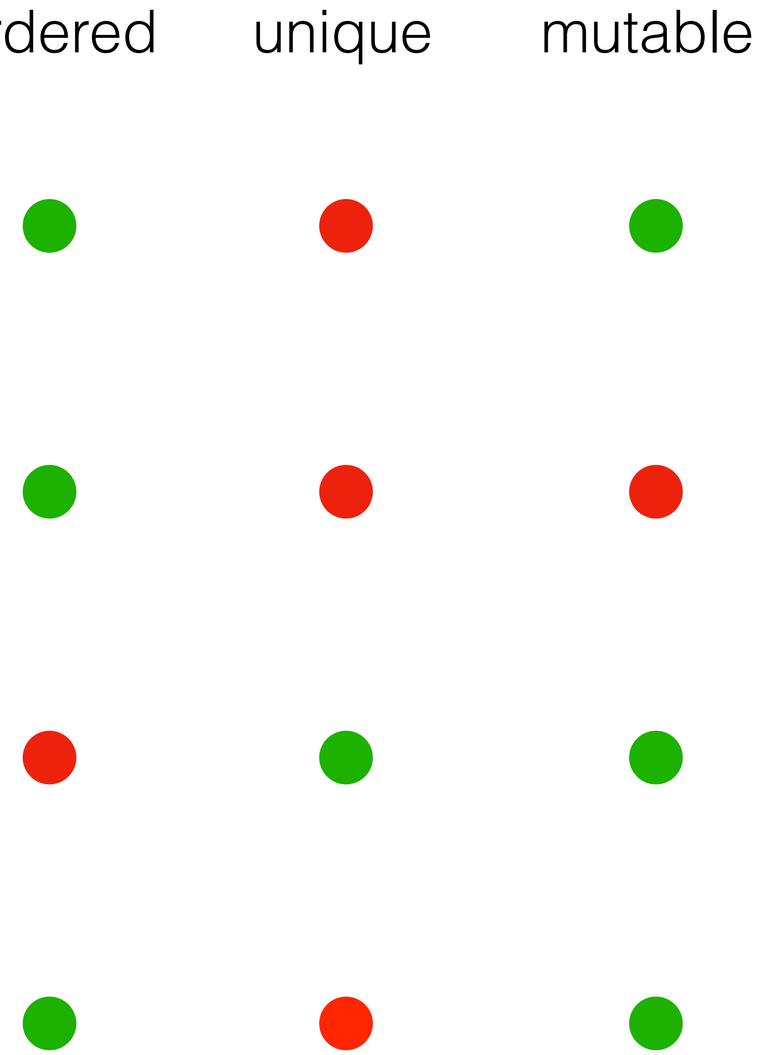
Data structures are not passive containers

- Each provides particular functions and prohibits other kinds of functions
- Each uses different organization behind the scenes to optimize distinct sets of tasks



Choosing the right data structure reduces coding errors and increases performance!

Contrasting the four most common data structures



`list`: items can be updated

By far the most versatile, but relatively slow

`tuple`: read-only version of a list

High performance, low memory, protects data

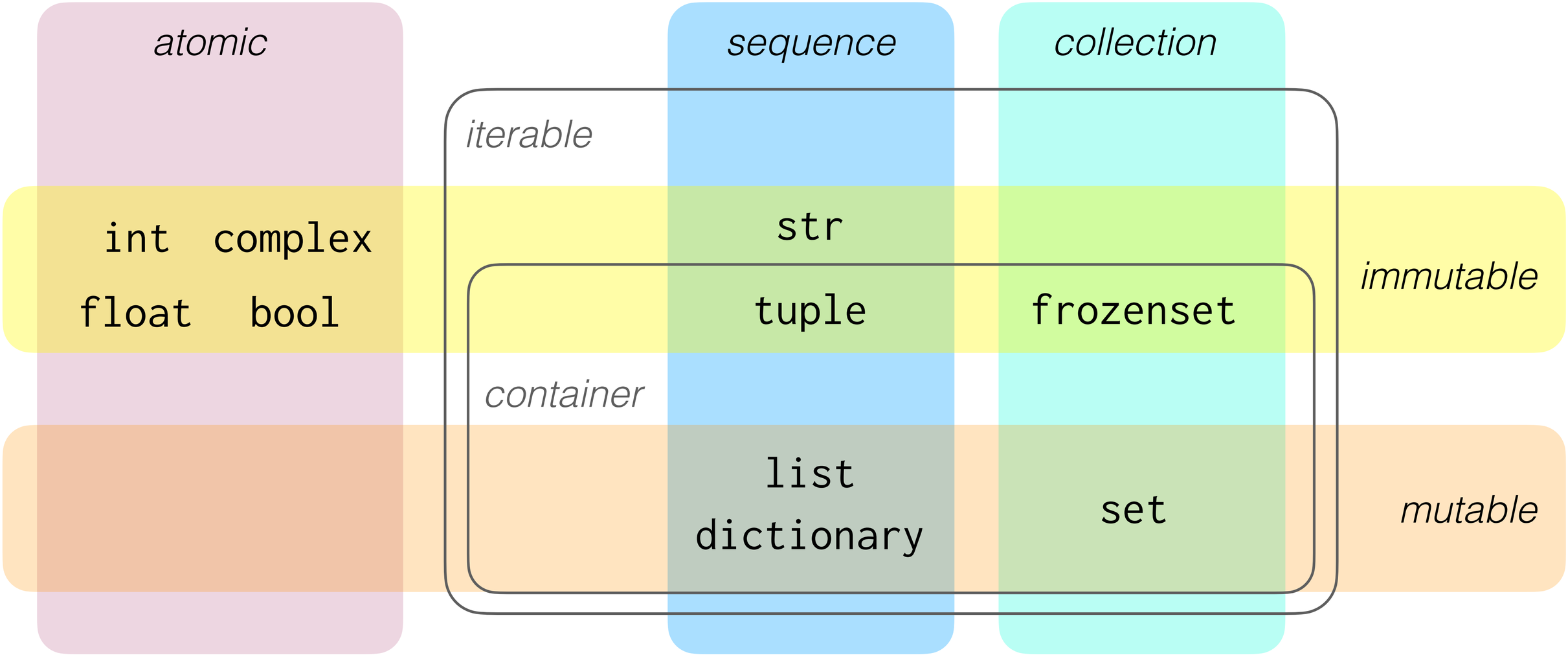
`set`: unordered, no duplicates

High performance, elegant set operations

`dictionary`: associative array or hash table

Powerful data structure indexed by names

Taxonomy of data structures in Python



Useful information for Python programming

Errors and exceptions

Syntax errors: code that does not conform to Python syntax

E.g.: missing comma, no closing bracket, = instead of ==

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[57], line 2
      1 s = "ATG"
----> 2 s[0] = "T"

TypeError: 'str' object does not support item assignment
```

Exceptions: errors detected during runtime

E.g.: divide by zero, import module does not exist, input file in wrong format

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[108], line 1
----> 1 l1.index(-99)

ValueError: -99 is not in list
```

Naming rules and conventions

Valid names:

- Must begin with a letter (or underscore, but reserve for special objects)
- May contain letters, digits, underscore, some additional Unicode
- Case matters (but avoid relying on case alone!)
- Can be arbitrarily long
- Cannot be keywords (but *can* be names of built-in functions: beware!) = “hiding”

Conventions:

- Use all-lowercase names for variables, functions, methods, modules, packages
- Use all-caps for constants
- Use nouns for variables and verbs for functions
- There are other common conventions, but these are the basic ones

Assignment

Plain assignment: create, modify, and access all of the standard data objects

Augmented assignment

```
i += 1                                  # i = i + 1; works with many operators
```

Simultaneous assignment

```
i, j = 'spam', 42
i, j = j, i                             # swaps values
```

Unpacking

```
i, j = my_tuple                         # assigns without need for indexing
i, _ = my_tuple                         # _ means ignore other items
i, j*  = my_tuple                       # * means remaining items are a list
i, _*, j  = my_tuple                    # assigns first and last items
```

Representing numbers

Plain numbers

Decimal point indicates float; no decimal indicates integer; `j` indicates complex

Scientific notation

```
35e3, 12E-4                    # valid floats (not integers)
e3, E-4                        # not valid; significand (mantissa) required
```

Long numbers

```
10_039_001                     # interpreted as integer 10039001
10,039,001                     # interpreted as tuple (1, 39, 1)
```

Integers in other bases

```
0b1, 0b110010                   # binary integers
0o23, 0h2A                      # octal and hexadecimal integers, respectively
```

Data types and casting

Data type for iterables:

```
my_list = [1, 2, 3]                 # implicit based on formatting
my_list = list(42)                  # explicit; class creator takes 1 argument
```

Data types can be converted when sensible:

```
my_list = list(my_tuple)            # tuple to list
my_integer = int(42.0)              # float to integer
my_str = str(42)                    # integer to string
```

Operators (for reference)

Arithmetic: `+`, `-`, `*`, `/`, `%`, `//`; but use `**` for exponentiation

String: `+` (concatenate), `*` (repeat)

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

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

Membership: `in`, `not in`, `<=` (subset), `>=` (superset)

Sets: `|` (union), `&` (intersection), `-` (difference), `^` (symmetric difference)

Bitwise: `|` (or), `^` (exclusive or), `&` (and), `<<` (shift left n bits), `>>` (shift right n bits)

Working with the common data structures

Two things to be aware of when working with data structures

The distinction between mutable and immutable objects:

- Immutable objects cannot be updated (but identifiers can be reassigned)
- Assignment of a mutable object does not create a true copy

The distinction between functions/methods that raise errors and those that don't:

- Some functions are "silent" if they are not able to do anything
- Other functions raise an error and halt execution of the program

Useful and common general-purpose functions

The following can be applied to any data object:

```
type()                          # returns type
isinstance()                    # tests for type; returns a boolean
dir()                           # returns valid methods for an object
```

The following can be applied to all iterable data objects:

```
len()                           # number of items (characters for string)
min(), max(), sum()             # applies to non-numeric values as well
all(), any()                    # returns bool; tests whether items are True
map()                           # applies function, returns altered items
filter()                        # applies condition, returns True items
```

The following can be applied to mutable sequence data objects:

```
reversed(), sorted()            # do not alter original object
```

Strings

Some useful string manipulation methods (use assignment to keep the result):

`.upper()`, `.lower()`, `.title()`, `.rjust()`, `.center()`

`.strip()`, `.lstrip()`, `.rstrip()`, `.removeprefix()`, `.removesuffix()`

`.replace()`

`.zfill()`

Concatenating and slicing:

`+`

`[]`, `.split()`, `.splitlines()`

Strings, continued

Testing for properties:

```
.isalpha(), isdigit(), .isupper(), .islower()    # and many others
.isidentifier()                                  # is a variable name valid?
in, not in, startswith(), endswith()
```

Searching:

```
.find(), .count()
```

Printing: f-string formatting

```
my_string = f'The planet {var_1} has a radius of {var_2} km'
```

Lists

Basic list content manipulation methods (most operate in-place, “mutating”):

```
.append(), .insert(), .remove(), .pop(), .extend()
.sort(), .reverse()                    # non-mutating alternatives mentioned earlier
.copy(), .clear()
```

Retrieve information about the contents of a list:

```
in,  not in, .count(), .index()
```

Two useful things to know:

```
my_list = list(set(my_list))     # removes duplicates from a list
zip()                            # conjoins two equal-length lists
```

Tuples

Only 2 methods because — immutable!

Retrieve information about the contents of a tuple:

```
in, not in, .count(), .index()
```

Extract the contents of a tuple so they can be manipulated (leaving the original intact):

```
my_list = list(my_tuple)
```

Sets

Basic set content manipulation methods (most are mutating):

`.add()`, `.update()`, `.remove()`, `.discard()`, `.pop()`

`.copy()`, `.clear()`

Set operations:

`|`, `.union()`, `&`, `.intersection()`, `-`, `.difference()`, `^`, `.symmetric_difference()`

`in`, `not in`

`.isdisjoint()`, `.issubset()`, `.issuperset()`

Dictionaries

Two common ways to create a dictionary:

```
my_dict = {‘A':1, 'B':2, 'C':3}
my_dict = dict(zip(['A', 'B', 'C'], [1, 2, 3]))
```

Basic dictionary content manipulation methods (most are mutating):

```
[], .add(), .update(), .remove(), .discard(), .pop()
.sort(), .reverse()                    # non-mutating alternatives mentioned earlier
.copy(), .clear()
```

Extract information from a dictionary:

```
[], .values(), .keys(), .items()
.get()                                 # no run-time error if key does not exist
```