Social Data Science with Python

Session 2: Data Structures, Functions, and Files

Ashrakat Elshehawy

Department of Politics and International Relations,
University of Oxford

February 7, 2023

Hello again!

Figure: Source: Reddit r/ProgrammerHumor

What have we covered last lecture?

- Introductions

What have we covered last lecture?

- Introductions
- Organizational

What have we covered last lecture?

- Introductions
- Organizational
- Why Python?

What have we covered last lecture?

- Introductions
- Organizational
- Why Python?
- What are Python and R good for

What have we covered last lecture?

- Introductions
- Organizational
- Why Python?
- What are Python and R good for
- GitHub and Stack Overflow

What have we covered last lecture?

- Introductions
- Organizational
- Why Python?
- What are Python and R good for
- GitHub and Stack Overflow
- **A Data Science Pipeline**

What have we covered last lecture?

- Introductions
- Organizational
- Why Python?
- What are Python and R good for
- GitHub and Stack Overflow
- **A Data Science Pipeline**
  - Get Data

What have we covered last lecture?

- Introductions
- Organizational
- Why Python?
- What are Python and R good for
- GitHub and Stack Overflow
- **A Data Science Pipeline**
  - Get Data
  - Clean Data

What have we covered last lecture?

- Introductions
- Organizational
- Why Python?
- What are Python and R good for
- GitHub and Stack Overflow
- **A Data Science Pipeline**
  - Get Data
  - Clean Data
  - Explore Data

What have we covered last lecture?

- Introductions
- Organizational
- Why Python?
- What are Python and R good for
- GitHub and Stack Overflow
- **A Data Science Pipeline**
  - Get Data
  - Clean Data
  - Explore Data
  - Modelling

What have we covered last lecture?

- Introductions
- Organizational
- Why Python?
- What are Python and R good for
- GitHub and Stack Overflow
- **A Data Science Pipeline**
  - Get Data
  - Clean Data
  - Explore Data
  - Modelling
  - Interpreting

What have we covered last lecture?

- Introductions
- Organizational
- Why Python?
- What are Python and R good for
- GitHub and Stack Overflow
- **A Data Science Pipeline**
  - Get Data
  - Clean Data
  - Explore Data
  - Modelling
  - Interpreting

What are we learning today?

- Representing Data on A Computer
- More about Variables, Data Types, and Data Objects
- **Structuring Observations and Storing Data in Python**

How do we think about Data?

Why are Data Structures helpful?

1. Speed up operations.

Why are Data Structures helpful?

1. Speed up operations.
2. Provides a specific way of sorting and organizing your data.

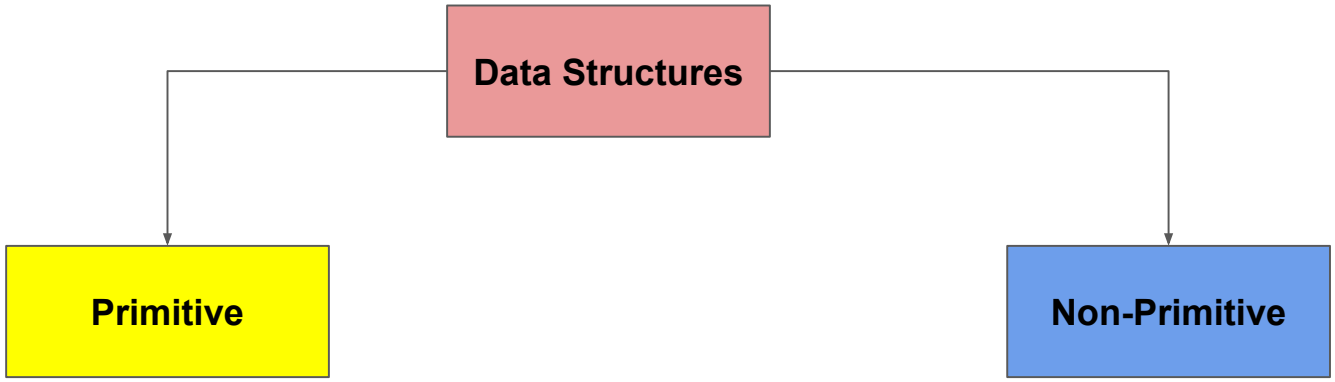
Why are Data Structures helpful?

1. Speed up operations.
2. Provides a specific way of sorting and organizing your data.
3. Primitive Structures: Simple ways of storing your values.

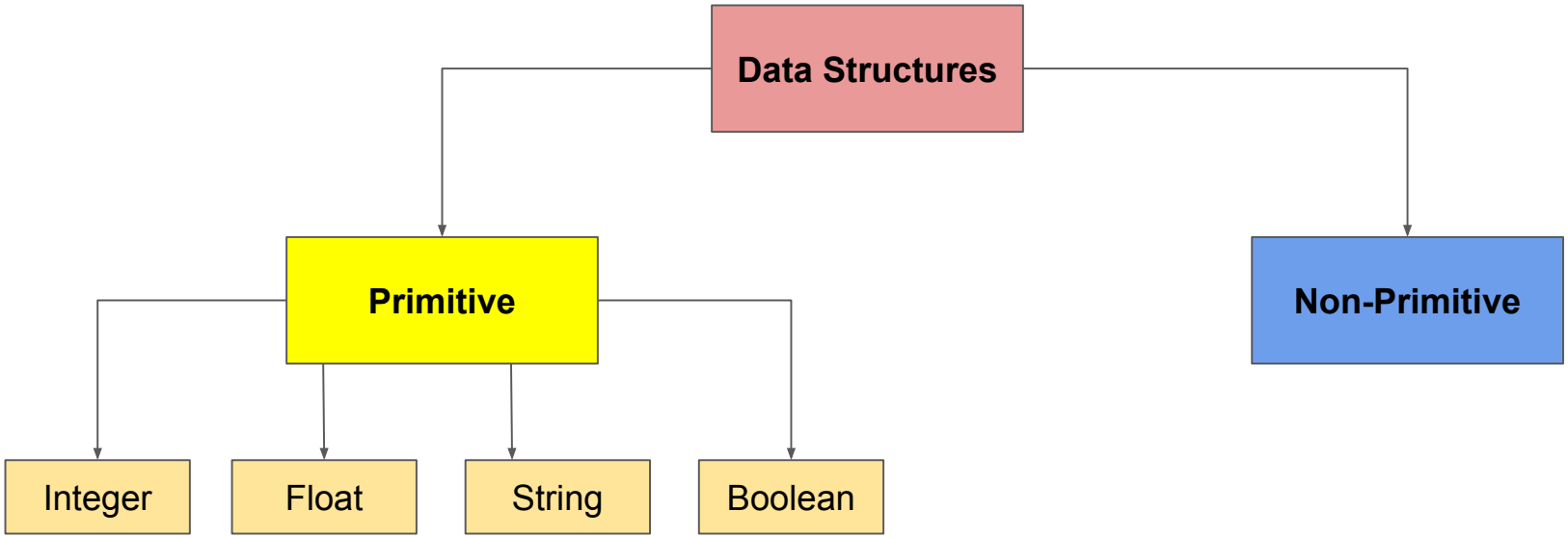
Why are Data Structures helpful?

1. Speed up operations.
2. Provides a specific way of sorting and organizing your data.
3. Primitive Structures: Simple ways of storing your values.
4. Non-Primitive Structures: More complex and advanced ways of representing data.

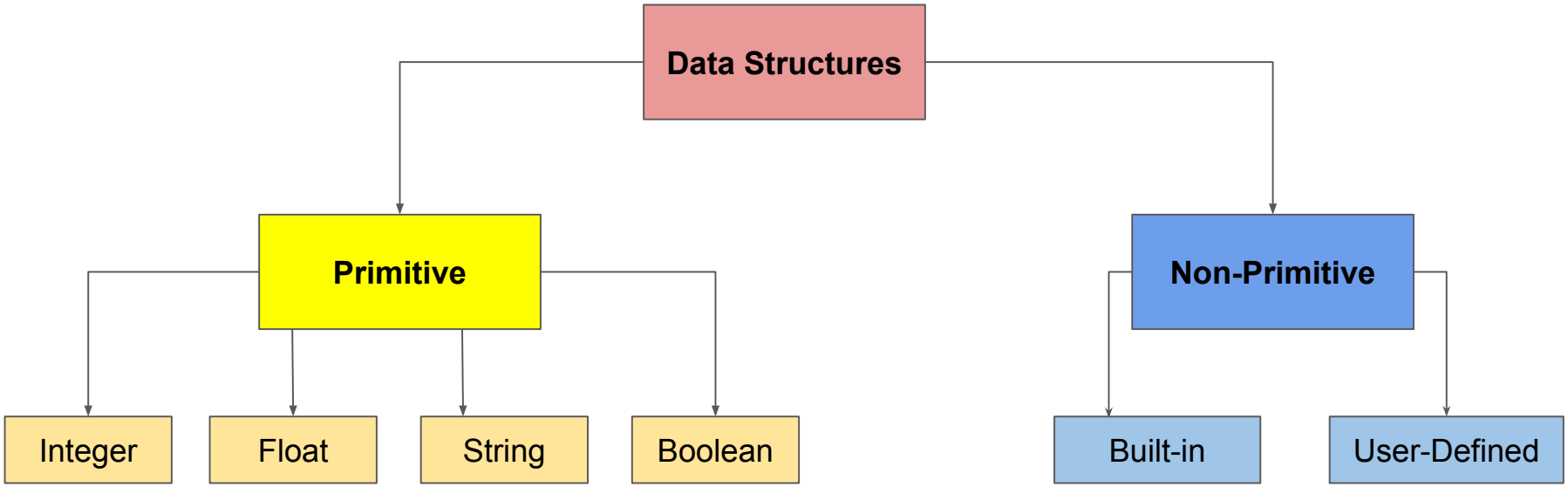
Python Data Structures



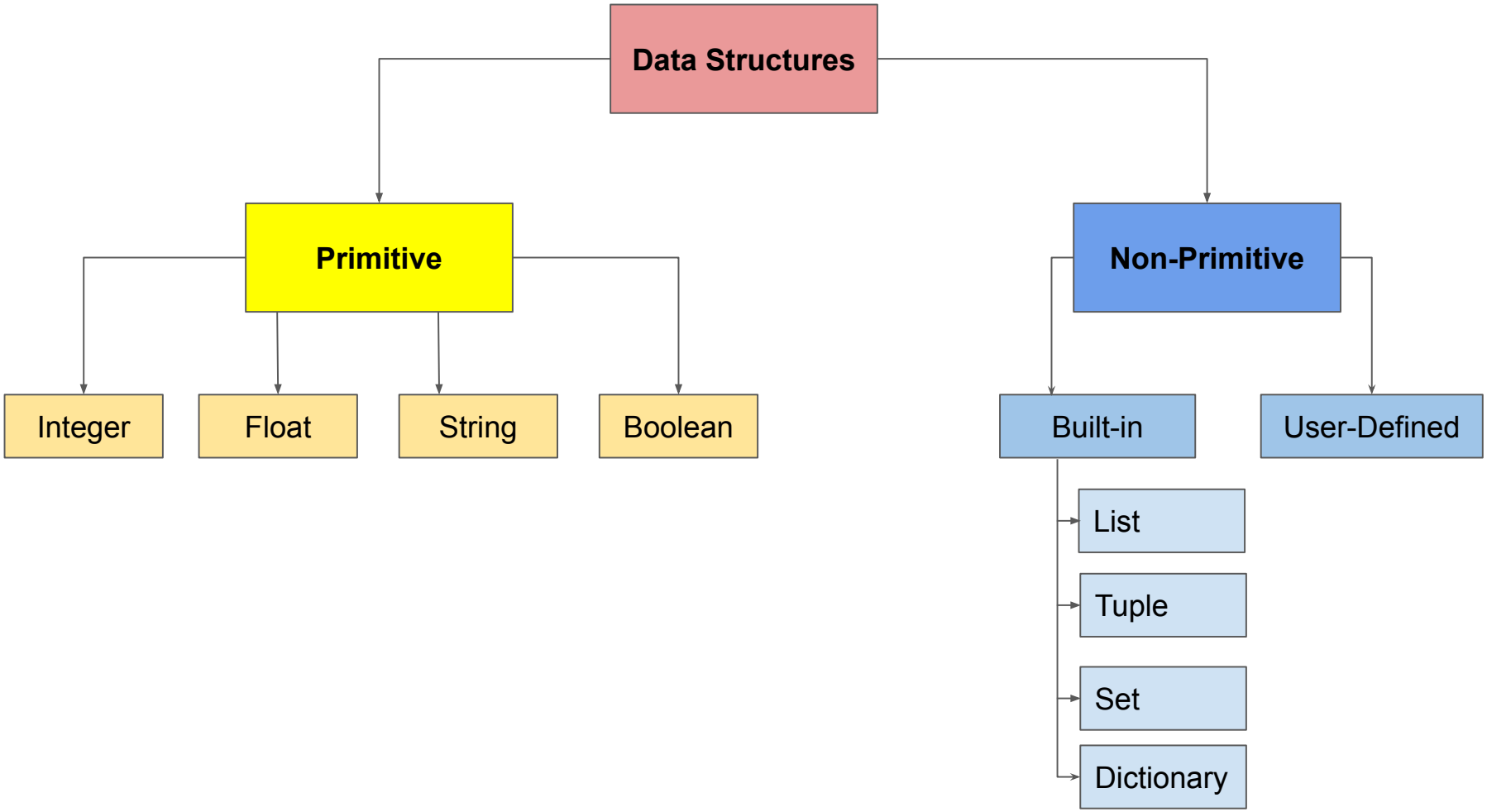
Python Data Structures



Python Data Structures



Python Data Structures



1. Lists

- Spot the squared brackets [,]

1. Lists

- Spot the squared brackets [,]
- Dynamic array: Can be modified.

```
arr = ["one", "two", "three"]
arr[1] = "hello"
>>> arr
['one', 'hello', 'three']
```

1. Lists

- Spot the squared brackets [,]
- Dynamic array: Can be modified.

```
arr = ["one", "two", "three"]
arr[1] = "hello"
>>> arr
['one', 'hello', 'three']
```

- Can hold arbitrary elements; everything is an object.

```
['one', 'three', 23]
```

1. Lists

- Spot the squared brackets [,]
- Dynamic array: Can be modified.

```
arr = ["one", "two", "three"]
arr[1] = "hello"
>>> arr
['one', 'hello', 'three']
```

- Can hold arbitrary elements; everything is an object.

```
['one', 'three', 23]
```

- Less tightly packed, takes space.

1. Lists

- Spot the squared brackets [,]
- Dynamic array: Can be modified.

```
arr = ["one", "two", "three"]
arr[1] = "hello"
>>> arr
['one', 'hello', 'three']
```

- Can hold arbitrary elements; everything is an object.

```
['one', 'three', 23]
```

- Less tightly packed, takes space.
- Sequenced: Indexed by position, starts counting at 0!

2. Tuples

```
("here", "is", "a" , "tuple")
```

2. Tuples

```
("here", "is", "a" , "tuple")
```

- Sequenced as Lists

2. Tuples

```
("here", "is", "a" , "tuple")
```

- Sequenced as Lists
- Immutable ($\neq$ Lists).

2. Tuples

```
("here", "is", "a" , "tuple")
```

- Sequenced as Lists
- Immutable ($\neq$ Lists).
  - No adding, deleting, or changing items

2. Tuples

```
("here", "is", "a" , "tuple")
```

- Sequenced as Lists
- Immutable ($\neq$ Lists).
  - No adding, deleting, or changing items
- Holds items of arbitrary data types.

2. Tuples

```
("here", "is", "a" , "tuple")
```

- Sequenced as Lists
- Immutable ($\neq$ Lists).
  - No adding, deleting, or changing items
- Holds items of arbitrary data types.
- Use less space.

3. Sets

```
myset1 = {"Python", "at", "DPIR","is","cool}
```

- Unordered.

3. Sets

```
myset1 = {"Python", "at", "DPIR","is","cool}
```

- Unordered.
- No duplicate items. Only unique values.

3. Sets

```
myset1 = {"Python", "at", "DPIR","is","cool}
```

- Unordered.
- No duplicate items. Only unique values.
- Mutable.

3. Sets

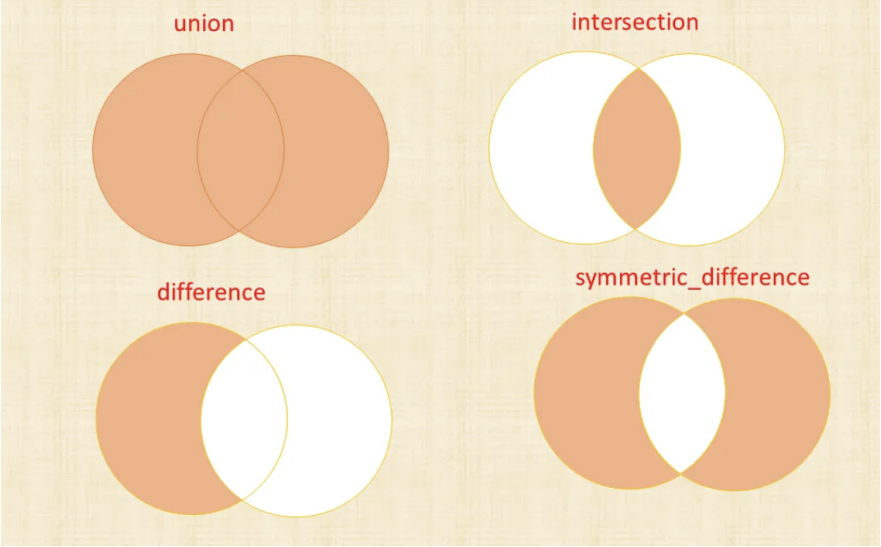
```
myset1 = {"Python", "at", "DPIR","is","cool}
```

- Unordered.
- No duplicate items. Only unique values.
- Mutable.
- No support for slicing or indexing operations.

3. Sets

```
myset1 = {"Python", "at", "DPIR","is","cool}
```

- Unordered.
- No duplicate items. Only unique values.
- Mutable.
- No support for slicing or indexing operations.
- Use it: find out if something exists, but nothing else.

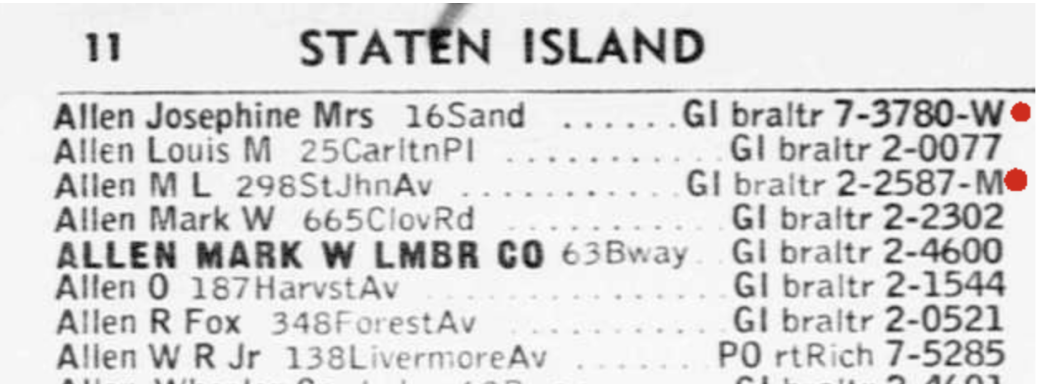


Source: Indhumathy Chelliah 2020 Medium

4. Dictionaries

- Key-Value pairs (Address Book).
- Keys must be unique.
- Mutable
- Tuples can be used as dictionary keys or values

Dictionaries

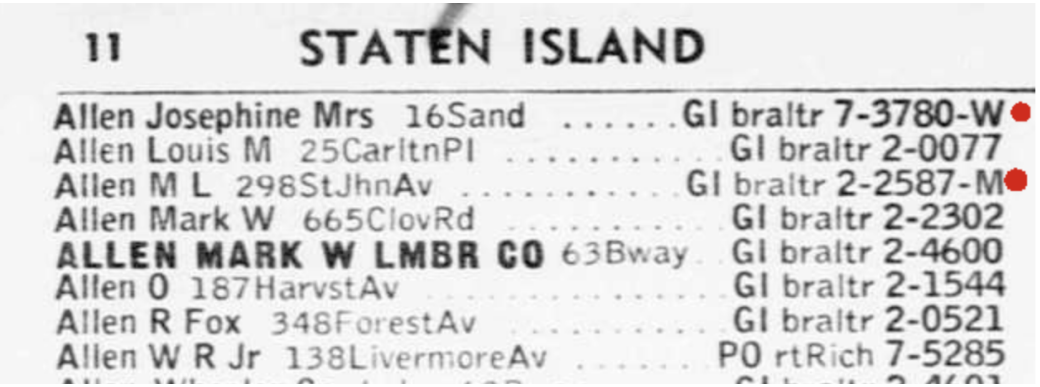

11 STATEN ISLAND

Allen Josephine Mrs 16Sand ..... GI braltr 7-3780-W
Allen Louis M 25CarltnPl ..... GI braltr 2-0077
Allen M L 298StJhnAv ..... GI braltr 2-2587-M
Allen Mark W 665ClovRd ..... GI braltr 2-2302
ALLEN MARK W LMBR CO 63Bway GI braltr 2-4600
Allen O 187HarvstAv ..... GI braltr 2-1544
Allen R Fox 348ForestAv ..... GI braltr 2-0521
Allen W R Jr 138LivermoreAv ..... PO rtRich 7-5285

Figure: Source: Internet Archive

Quiz: We want to create a telephone directory that maps from last-name, first-name pairs to telephone numbers. how could write a dictionary assignment statement?

Dictionaries


11 STATEN ISLAND

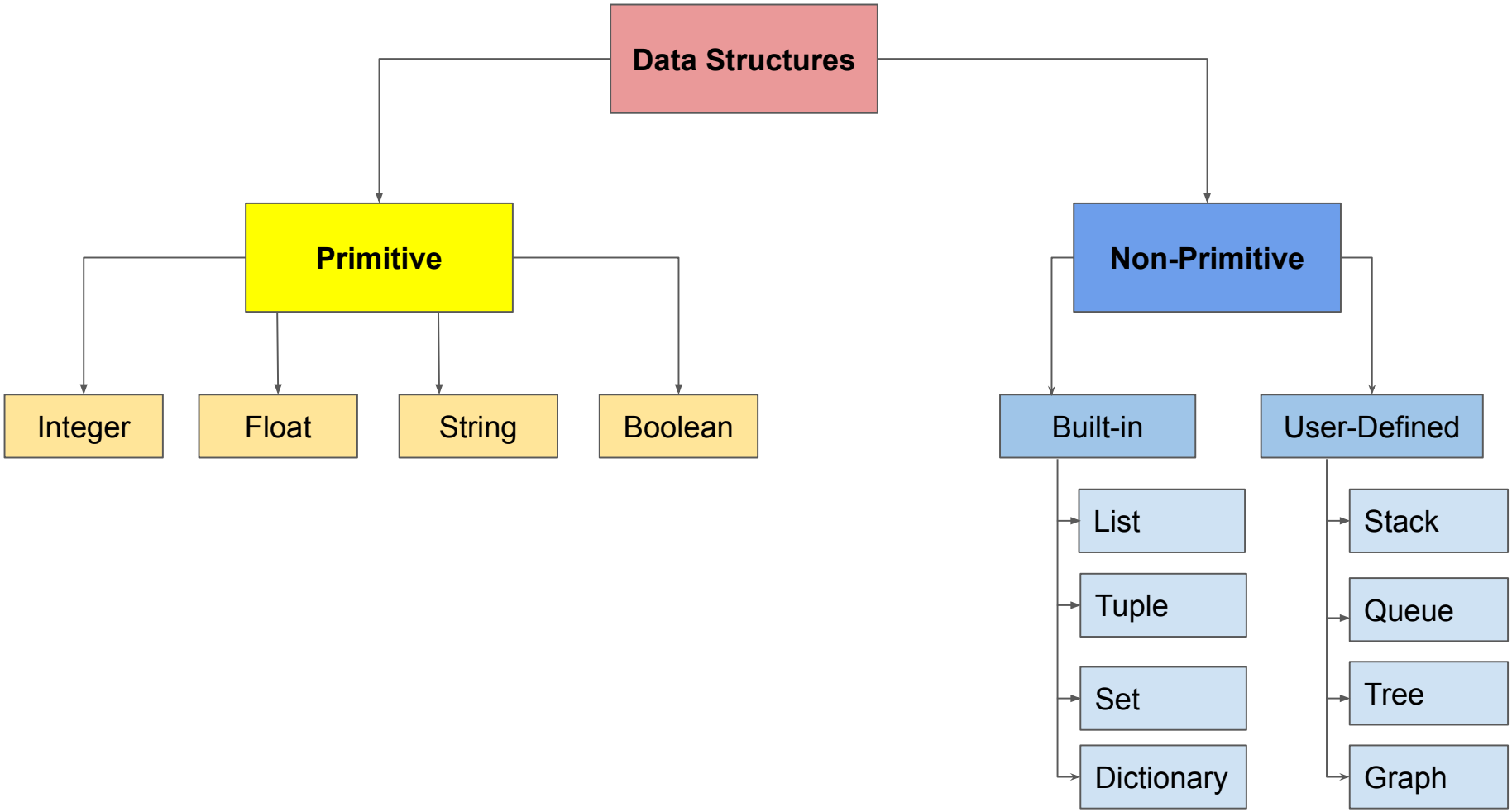
Allen Josephine Mrs 16Sand ..... GI braltr 7-3780-W
Allen Louis M 25CarltnPl ..... GI braltr 2-0077
Allen M L 298StJhnAv ..... GI braltr 2-2587-M
Allen Mark W 665ClovRd ..... GI braltr 2-2302
ALLEN MARK W LMBR CO 63Bway GI braltr 2-4600
Allen O 187HarvstAv ..... GI braltr 2-1544
Allen R Fox 348ForestAv ..... GI braltr 2-0521
Allen W R Jr 138LivermoreAv ..... PO rtRich 7-5285

Figure: Source: Internet Archive

Quiz: We want to create a telephone directory that maps from last-name, first-name pairs to telephone numbers. how could write a dictionary assignment statement?

```
my_dict =
        {(FirstName1, LastName1): Number,
        (FirstName2, LastName2): Number}
```

Python Data Structures



Stack

Implementation and deletions restricted to last item (Undo).

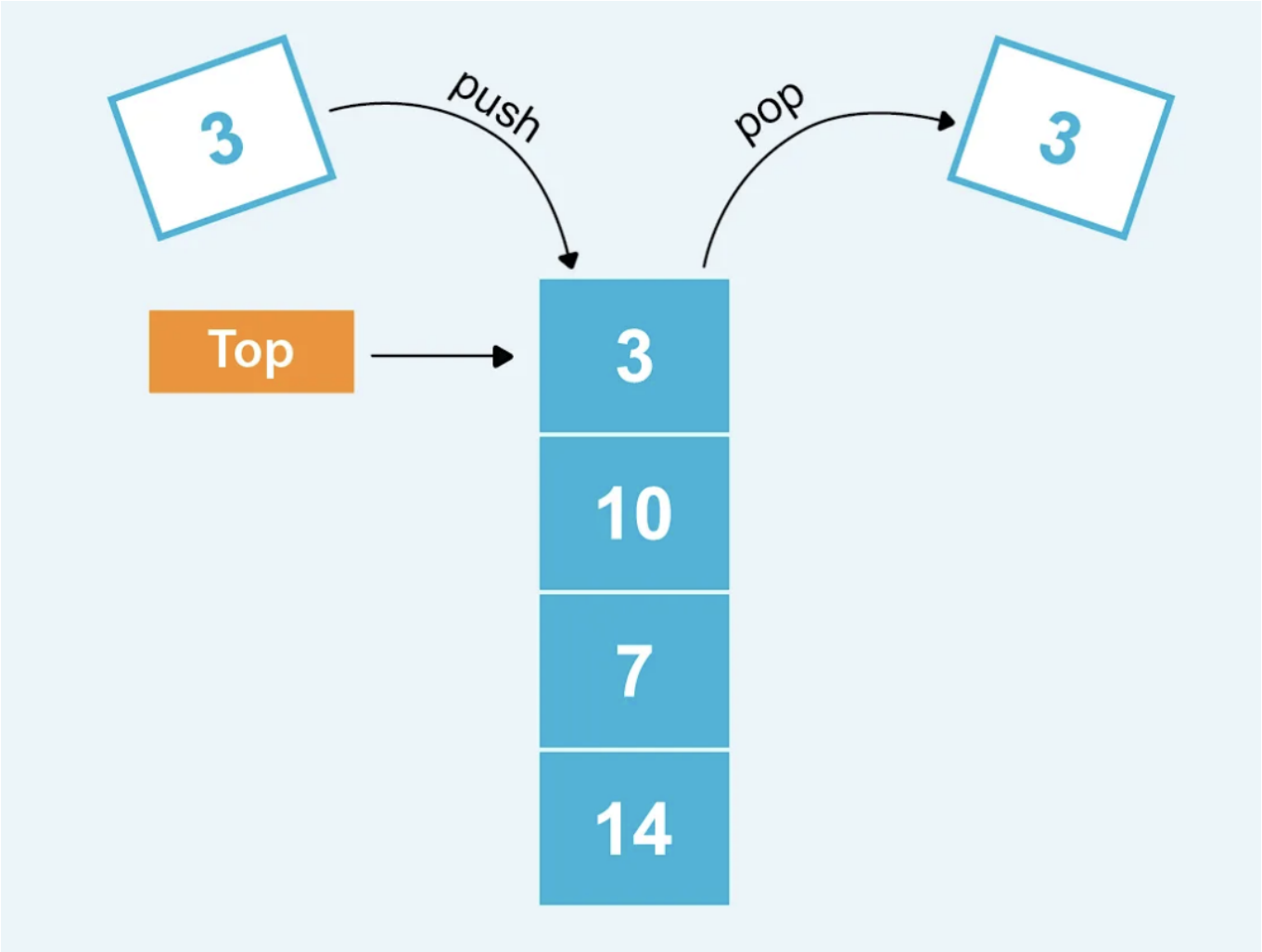


Figure: Source: Nikki Siapno 2022 Medium

Queue

First come, first serve!

Figure: Source: Link

Graph

Network Model

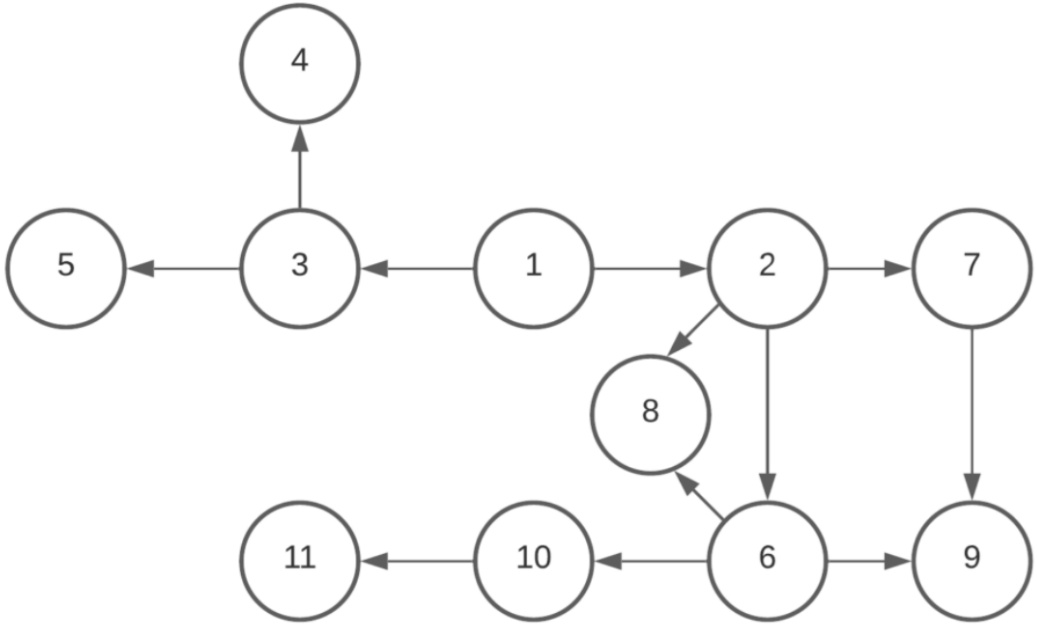


Figure: Source: Baeldung (Link)

Tree

Example: Family Tree

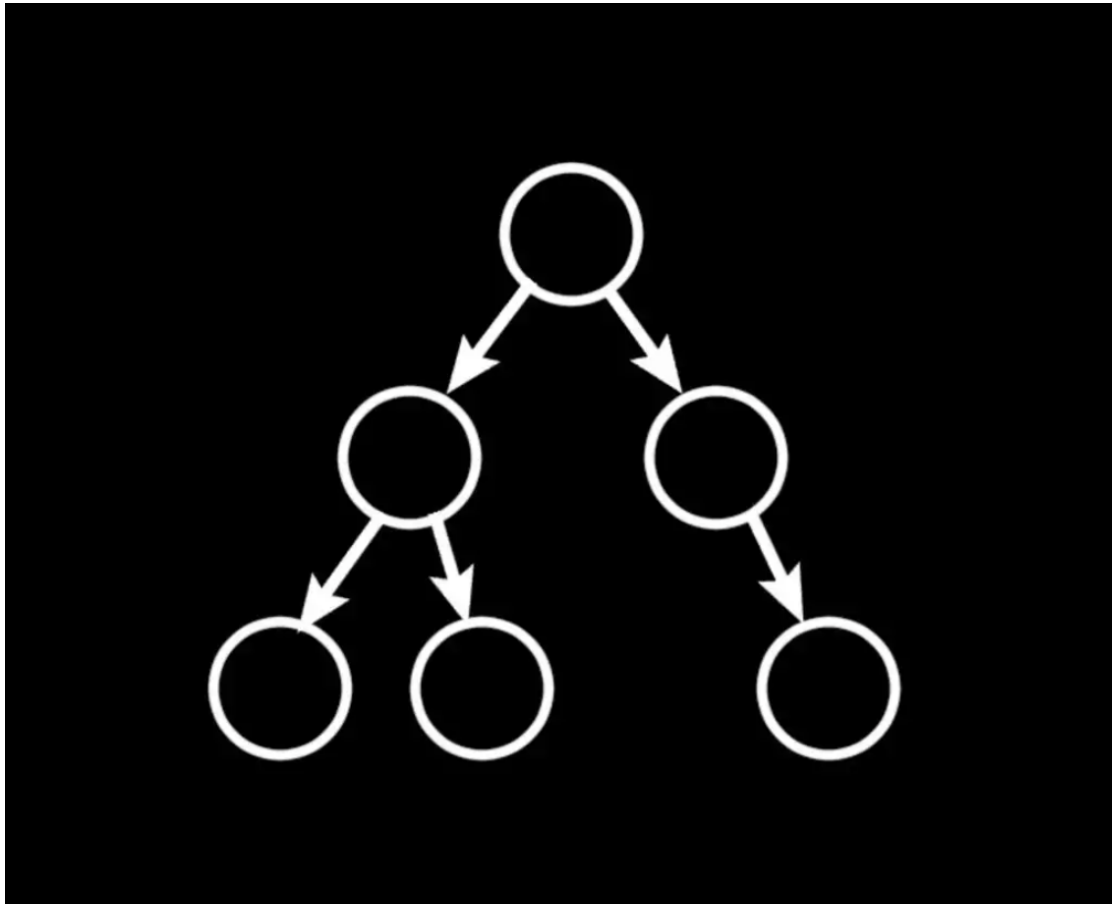


Figure: Source: Keno Leon (Link)

Data Challenge:

How would you structure the information given by this family tree in a tabular data structure? What would be the columns (variables) and rows (observations)? (challenge by Harukawa 2020)

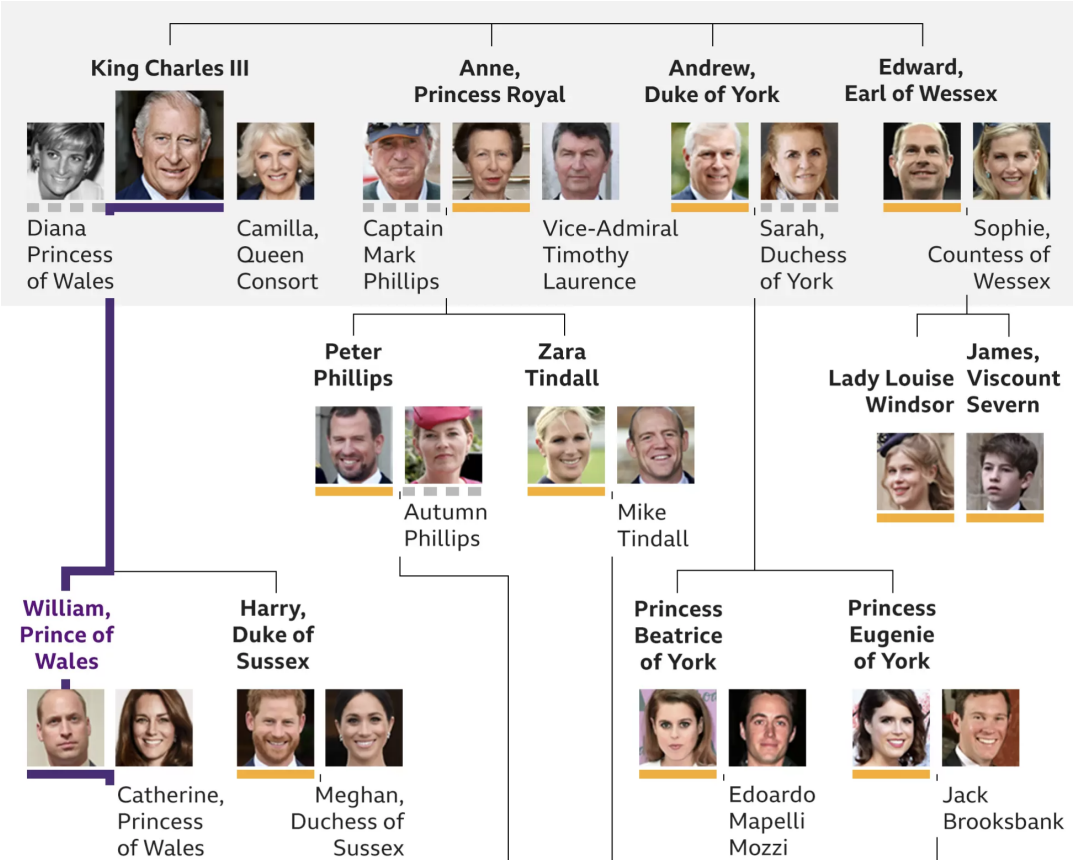


Figure: Source: BBC UK News 1.2.23

From Data Points to Data Frame: Tabular Data

More about this next time!

Creating Pandas DataFrames from Python Lists and Dictionaries

Dictionary

Row Oriented

```
sales = [{'account': 'Jones LLC', 'Jan': 150, 'Feb': 200, 'Mar': 140},
         {'account': 'Alpha Co', 'Jan': 200, 'Feb': 210, 'Mar': 215},
         {'account': 'Blue Inc', 'Jan': 50, 'Feb': 90, 'Mar': 95 }]
df = pd.DataFrame(sales)
```

default

List

```
sales = [('Jones LLC', 150, 200, 50),
         ('Alpha Co', 200, 210, 90),
         ('Blue Inc', 140, 215, 95)]
labels = ['account', 'Jan', 'Feb', 'Mar']
df = pd.DataFrame.from_records(sales, columns=labels)
```

from_records

	account	Jan	Feb	Mar
0	Jones LLC	150	200	140
1	Alpha Co	200	210	215
2	Blue Inc	50	90	95

Column Oriented

```
sales = {'account': ['Jones LLC', 'Alpha Co', 'Blue Inc'],
         'Jan': [150, 200, 50],
         'Feb': [200, 210, 90],
         'Mar': [140, 215, 95]}
df = pd.DataFrame.from_dict(sales)
```

from_dict

```
sales = [('account', ['Jones LLC', 'Alpha Co', 'Blue Inc']),
         ('Jan', [150, 200, 50]),
         ('Feb', [200, 210, 90]),
         ('Mar', [140, 215, 95]) ]
df = pd.DataFrame.from_items(sales)
```

from_items

When using a dictionary, column order is not preserved.
Explicitly order them:
`df = df[['account', 'Jan', 'Feb', 'Mar']]`

Practical Business Python - pbpython.com

Figure: Source: Chris Moffitt 2016

Coding Session Today:

- Data structures

Coding Session Today:

- Data structures
- Control Structures: Learn to iterate through data structures

Coding Session Today:

- Data structures
- Control Structures: Learn to iterate through data structures
  - Loops

Coding Session Today:

- Data structures
- Control Structures: Learn to iterate through data structures
  - Loops
  - If-Else Statement

Coding Session Today:

- Data structures
- Control Structures: Learn to iterate through data structures
  - Loops
  - If-Else Statement
  - List Comprehension

Coding Session Today:

- Data structures
- Control Structures: Learn to iterate through data structures
  - Loops
  - If-Else Statement
  - List Comprehension

If-Else Statement

```
if expression :
    statement 1
    statement 2
    ...
    statement n
```

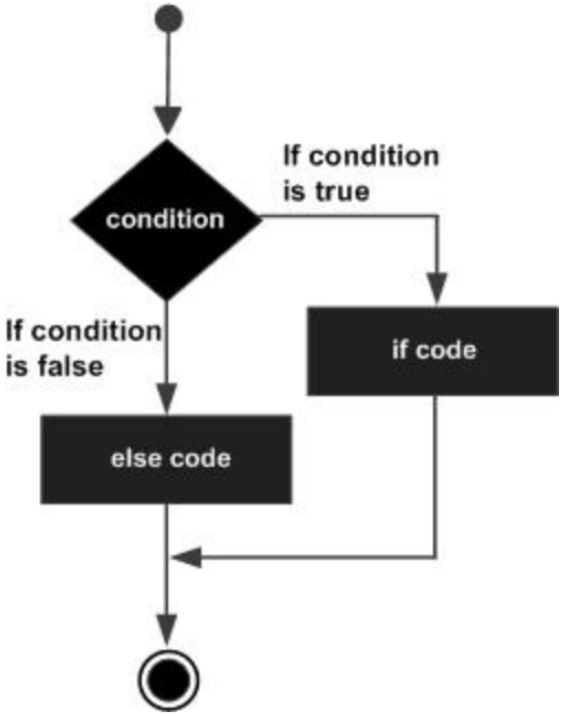


Figure: Source: Tutorials Point

List Comprehension

- Simple, compact, fast.

List Comprehension

- Simple, compact, fast.
- More Pythonic than loops:

List Comprehension

- Simple, compact, fast.
- More Pythonic than loops:
  - map + filter + list generation
  - no new strategy for each situation

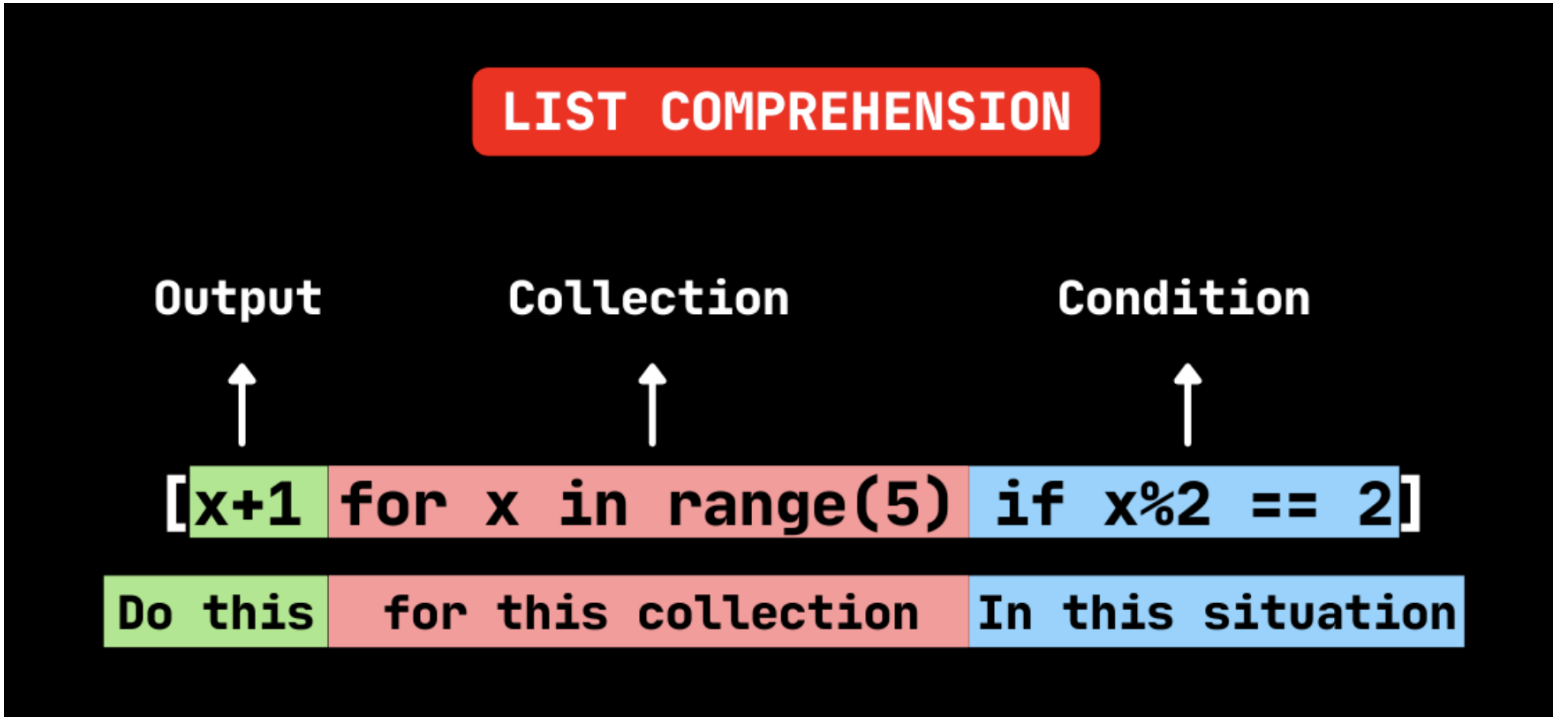


Figure: Source: Buggy Programmer