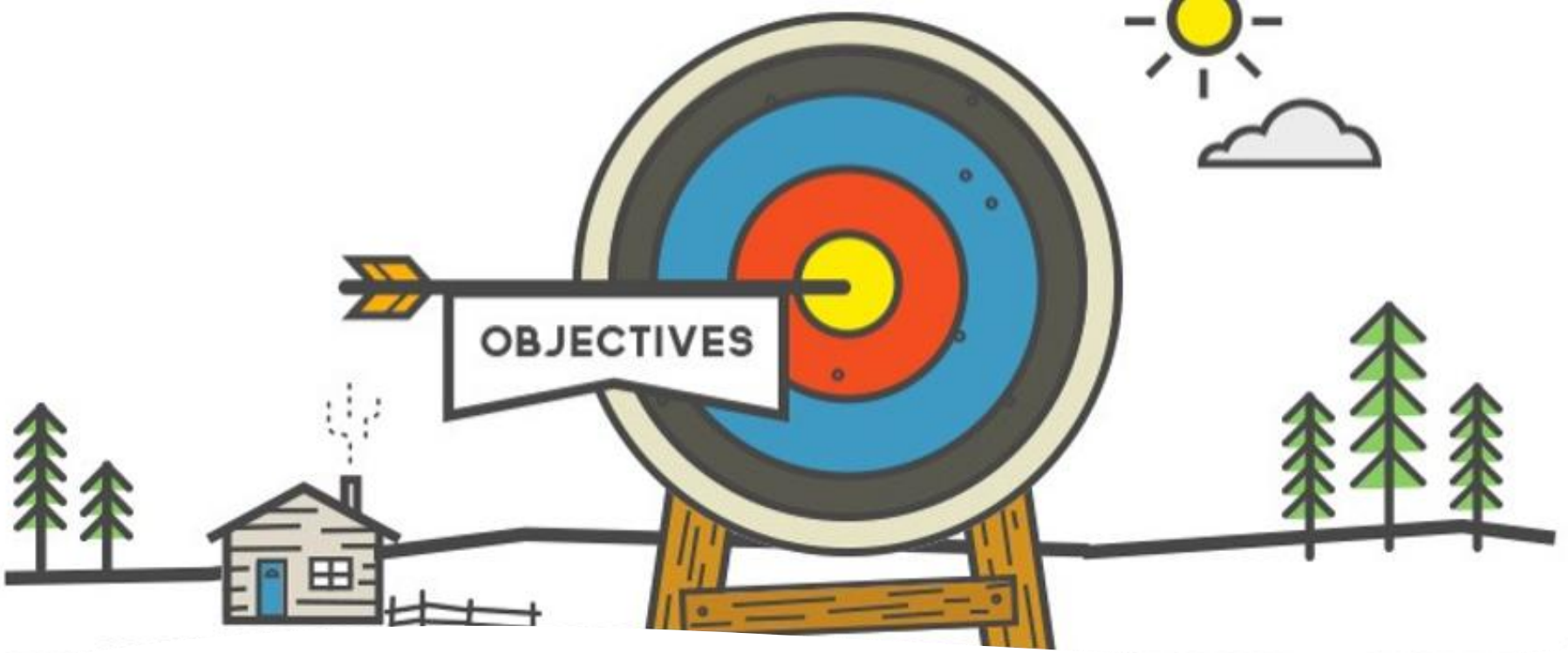
Lisbon School of Economics & Management
Universidade de Lisboa

LISBOA
UNIVERSIDADE DE LISBOA

PYTHON DATA STRUCTURES

Carlos J. Costa



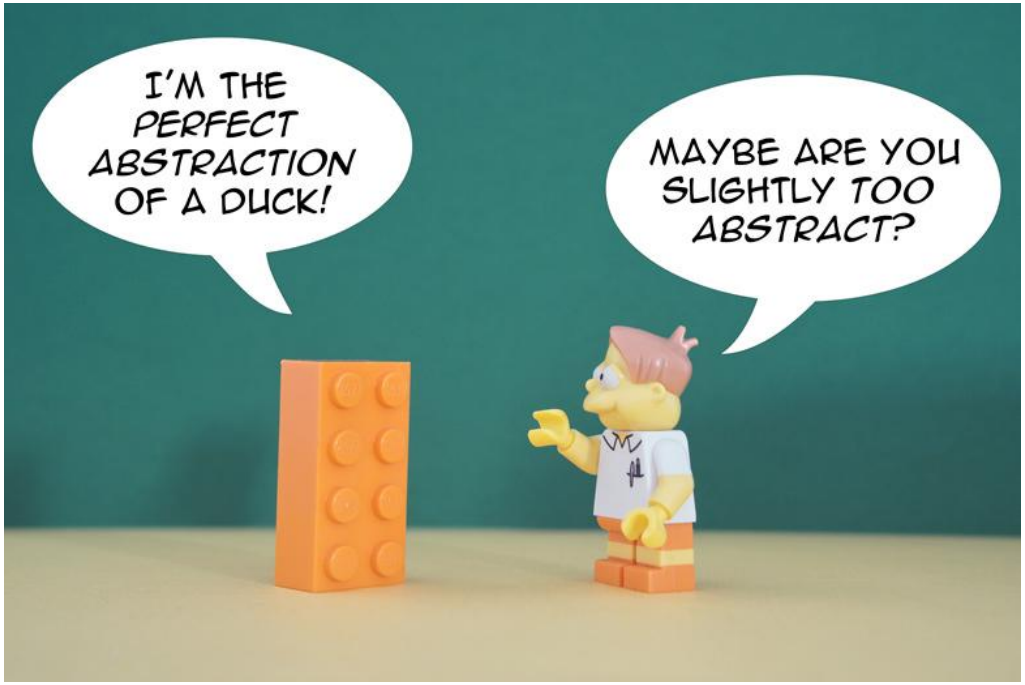
Learning Objectives

- Now the main built-ins Python data structures
- Understand how to manipulate data organized in lists, tuples, sets and dictionaries
- Use built-ins Python data structures to solve problems

Index

- Abstract Data Types
- Data Structure
- Lists
- Tuples
- Sets
- Dictionaries

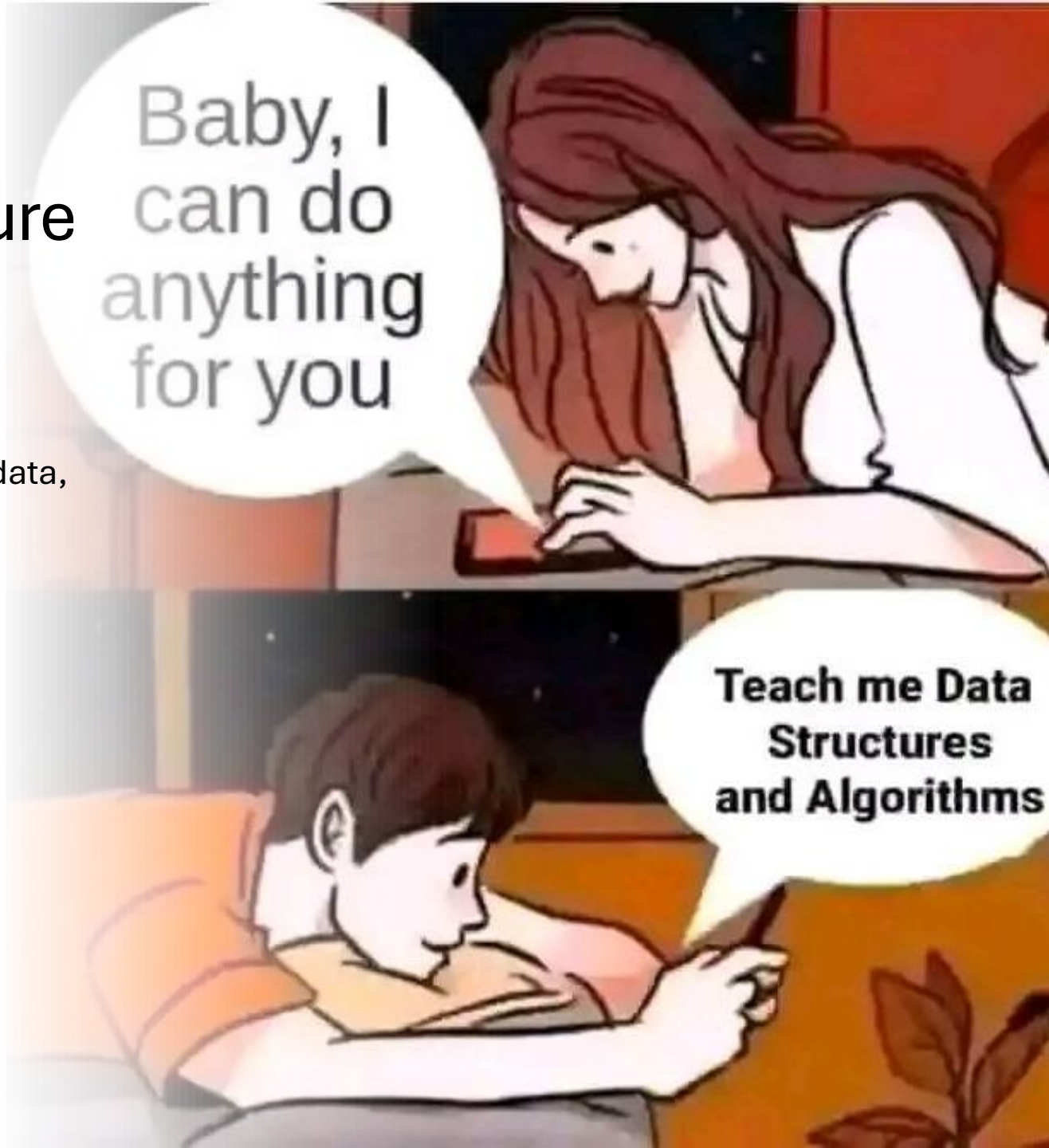
Abstract Data Type (ADT)



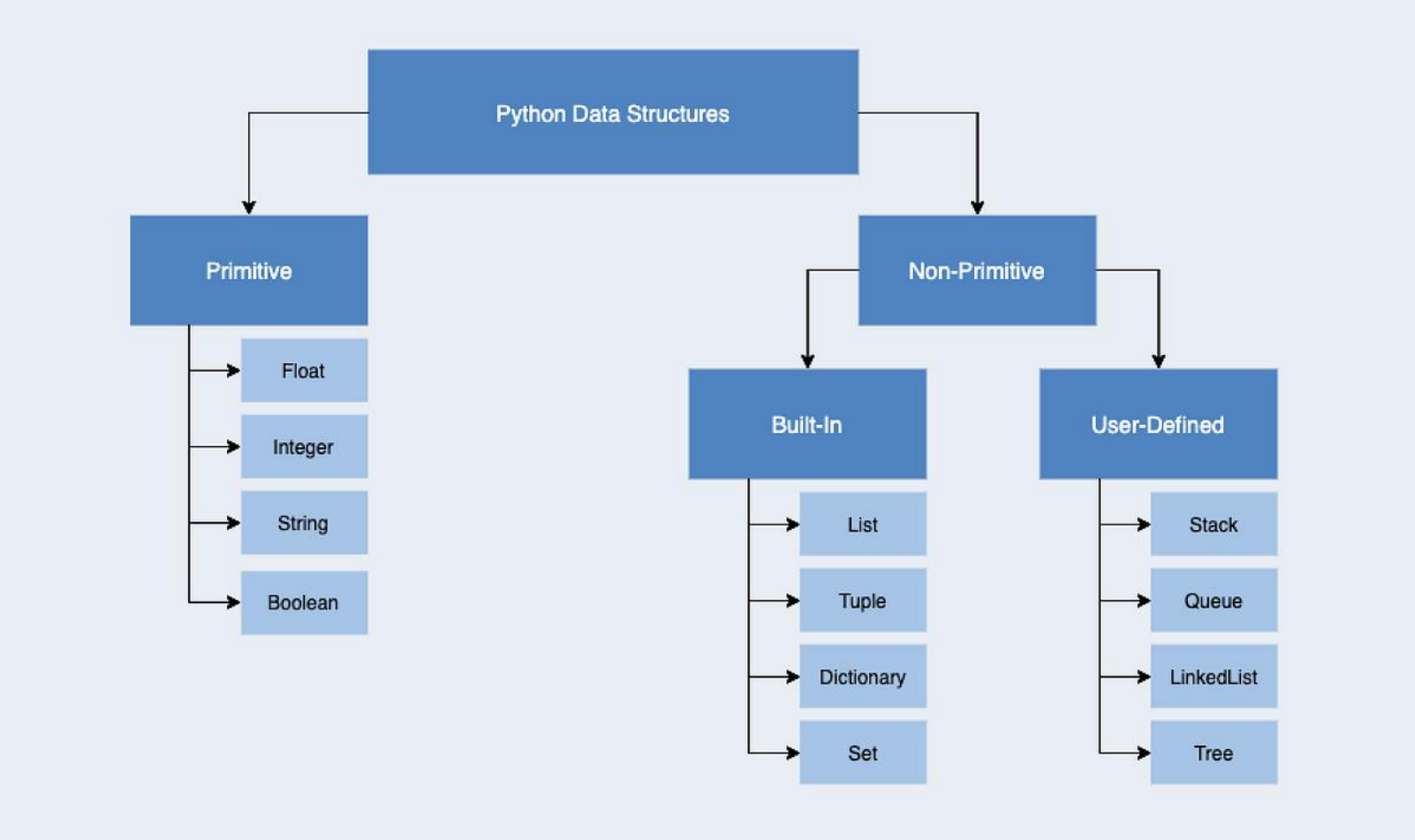
- is a type (or class) for objects
- behaviour is defined by a set of value and a set of operations.

Data Structure

- concrete representations of data,
- perspective of an implementer



DatabStructures



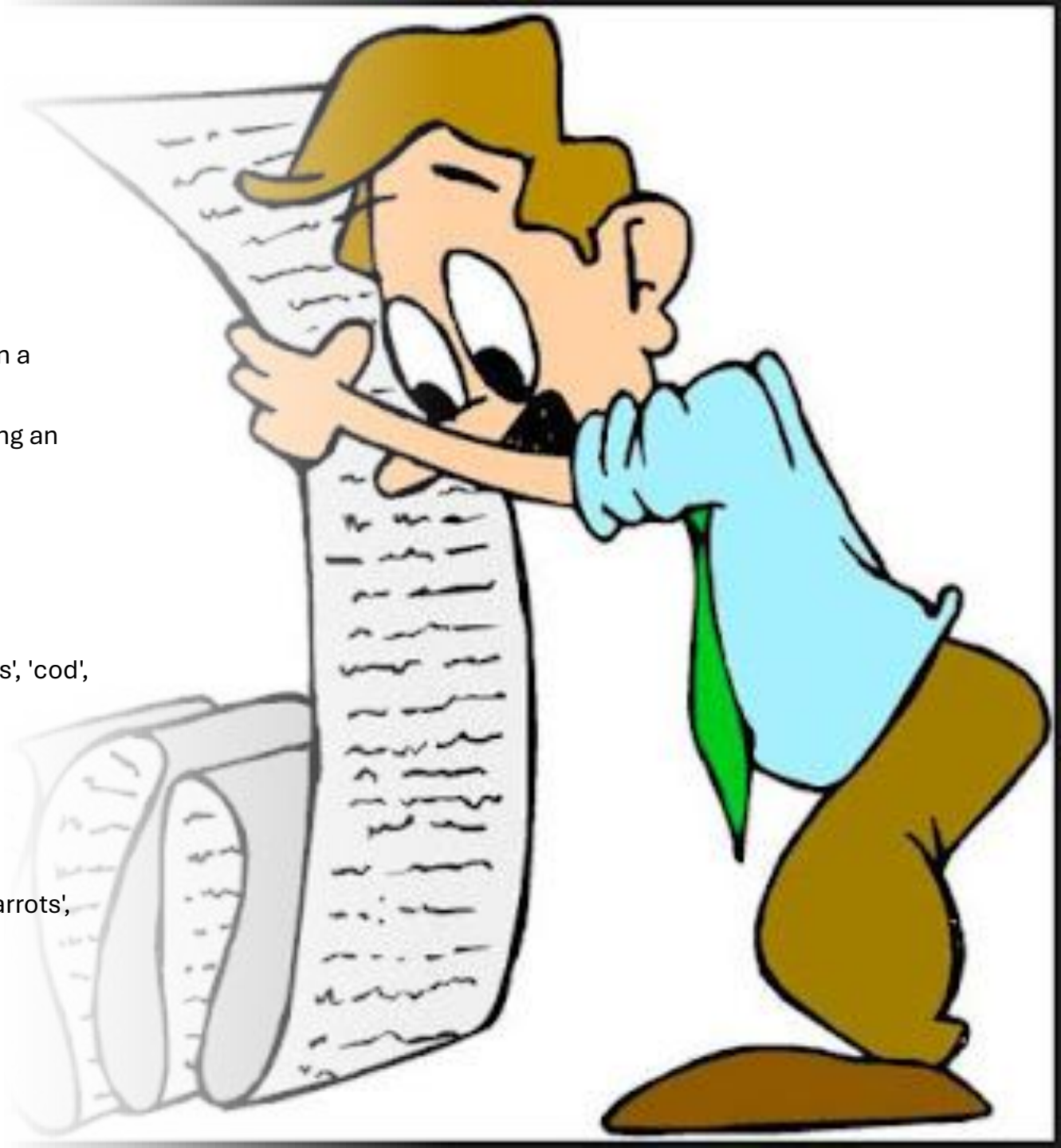
Lists

- A list stores a series of items in a specific order
- You can access each item using an index or loop
- The lists are mutable
- # Construct a list

```
shoppingList = ['potatoes', 'carrots', 'cod', 'sprouts’]
```

- Or else

```
shoppingList = list (('potatoes', 'carrots', 'cod', 'sprouts'))
```



Lists

- Operations with lists

- # Get the first element of the list
- shoppingList[0]
- # Get the last element from the list
- shoppingList[-1]

Lists

- Operations with lists

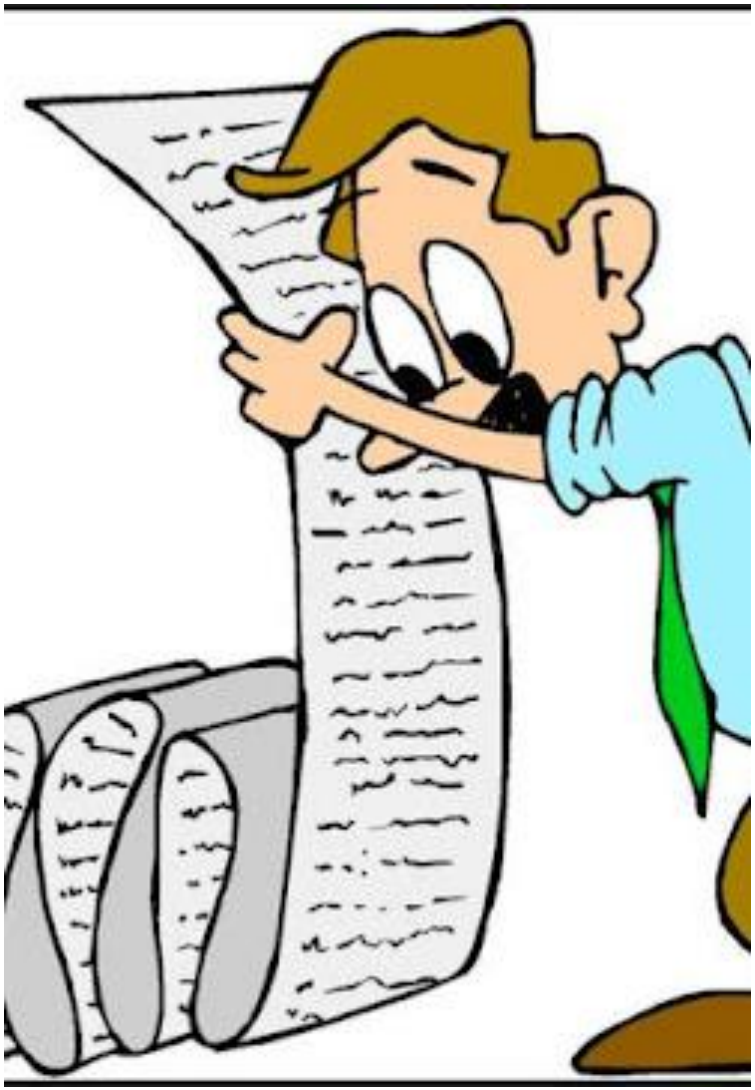
```
# Iterate though a list

for purchase in shoppingList:
    print(purchase)
```

Lists

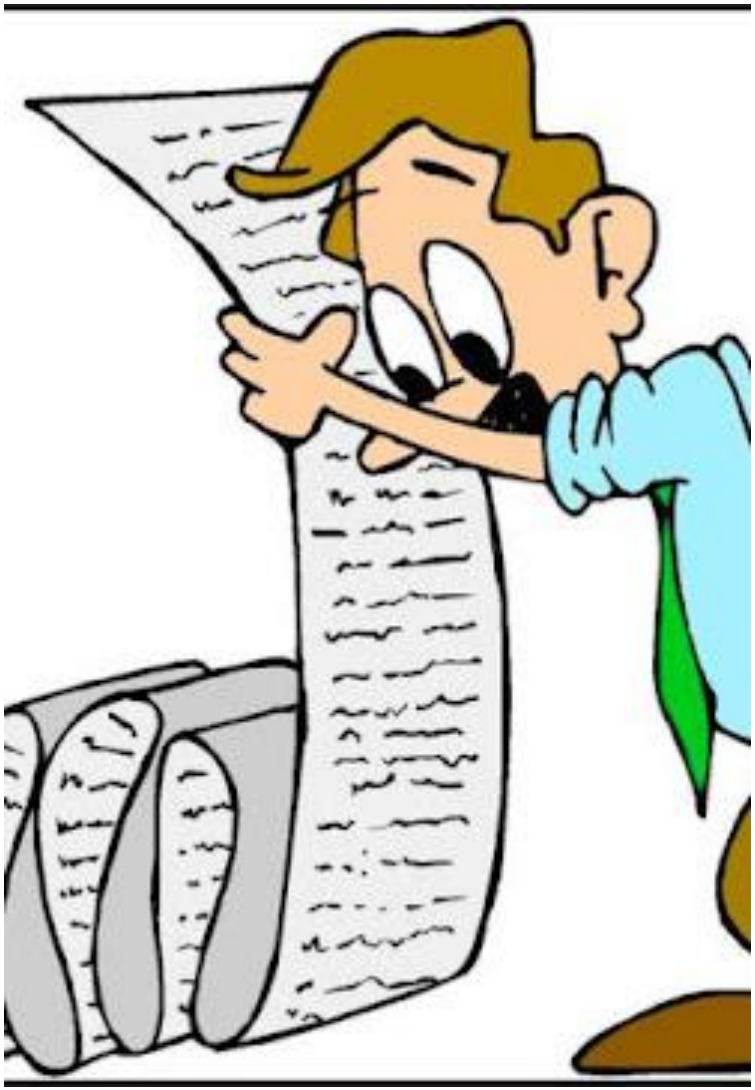
- Add Items

```
# Add an item to a list
films =[]
films.append('Vice')
films.append('Green Book')
films.append('Roma')
films.append('A Star Is Born')
print (films)
```



Lists

- Remove elements
- films.pop()
- films.remove('A Star Is Born')
- del films[1]
- print (films)
- Delete List elements:
- films.clear()
- Delete list
- del films



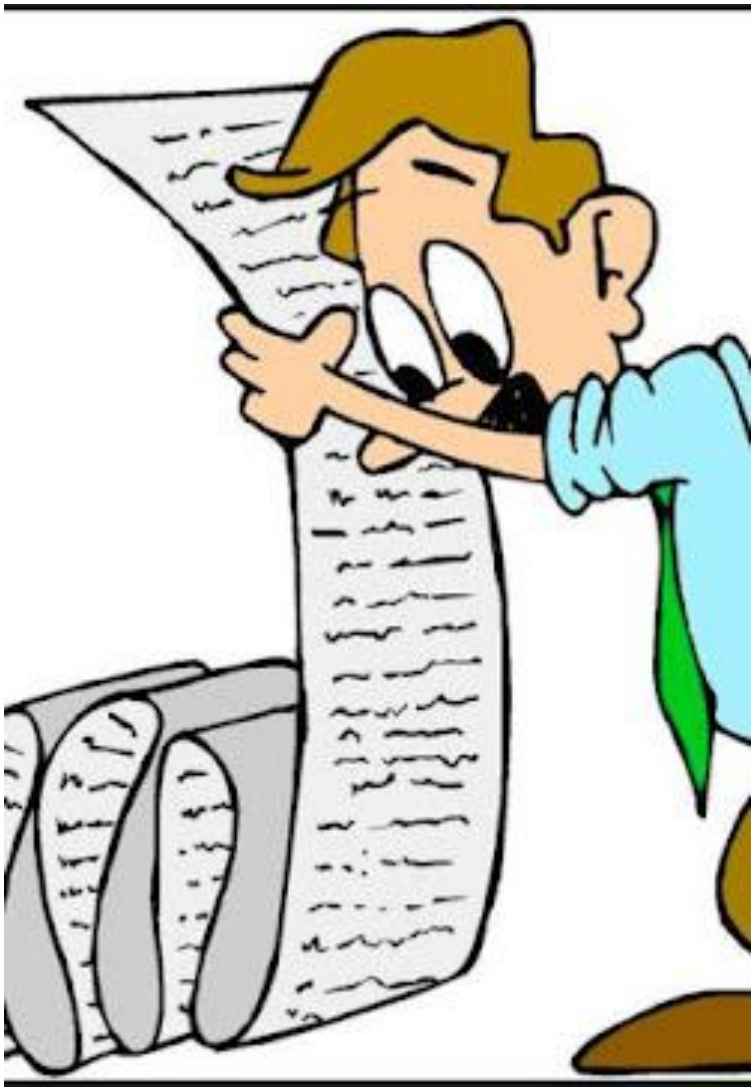
Lists

- List comprehension
- # Compress list
- squares = [x**2 for x in range(1, 11)]

Lists

- Slicing the list

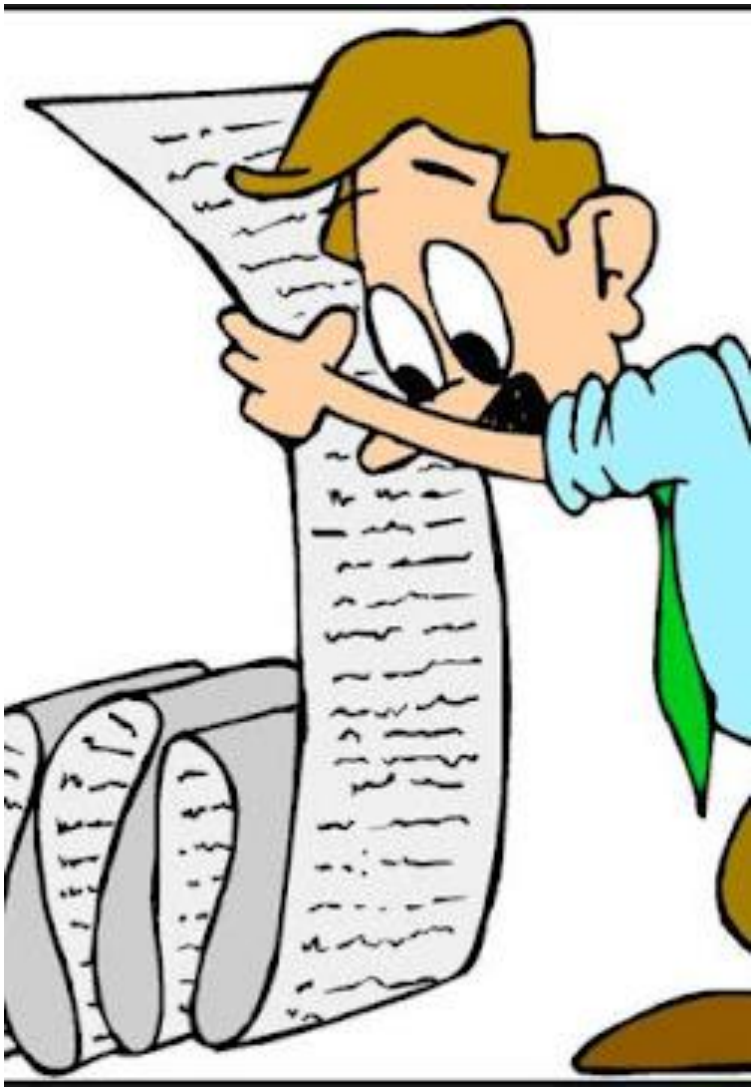
```
# Obtain the first 3 elements of the list
shoppingList = ['potatoes', 'carrots', 'cod', 'sprouts']
firstThree = shoppingList [: 3]
print (firstThree)
```



Lists

- What's the result of?

```
shopping = shoppingList
shoppingList.append(“orange")
print(shopping)
```

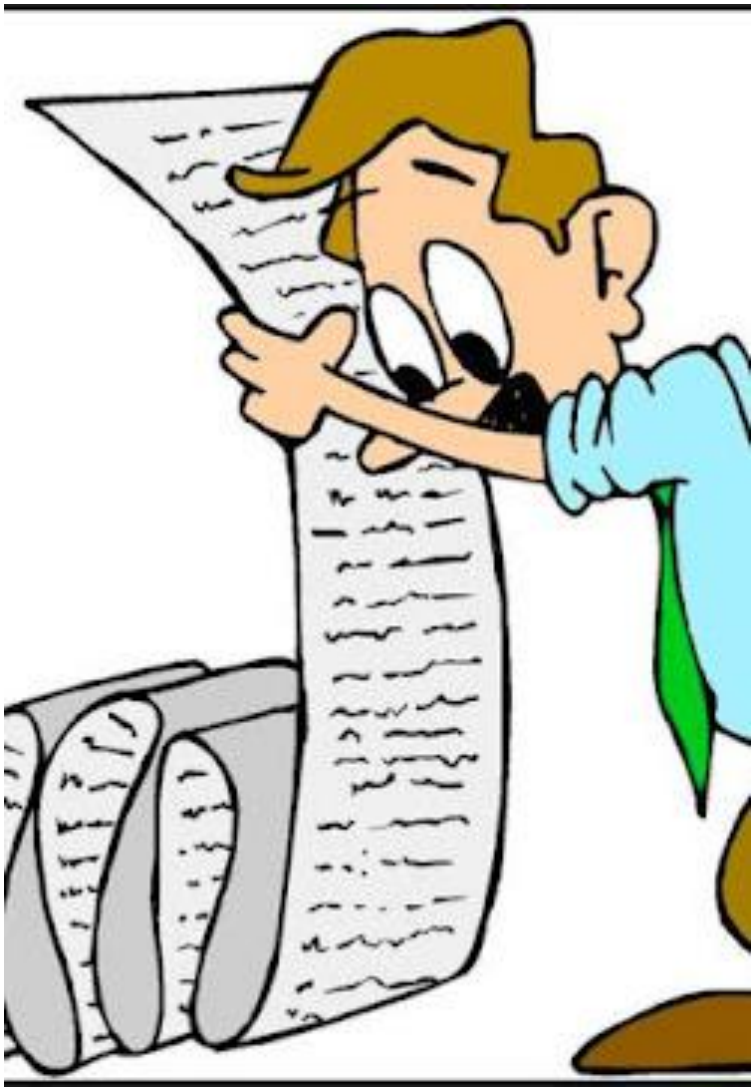


Lists

- Yes.. Do not copy… To copy, you make:

```
# copy a list

shoppingListCopy = shoppingList[:]
```



Tuples

- The tuples are identical to the lists, but tuples cannot be modified
- Tuples are immutable

```
newPurchases= (“bananas”, “beans”, “rice”)

print (newPurchases [1])

newPurchases [0] = “apple”
```

- What is the output?

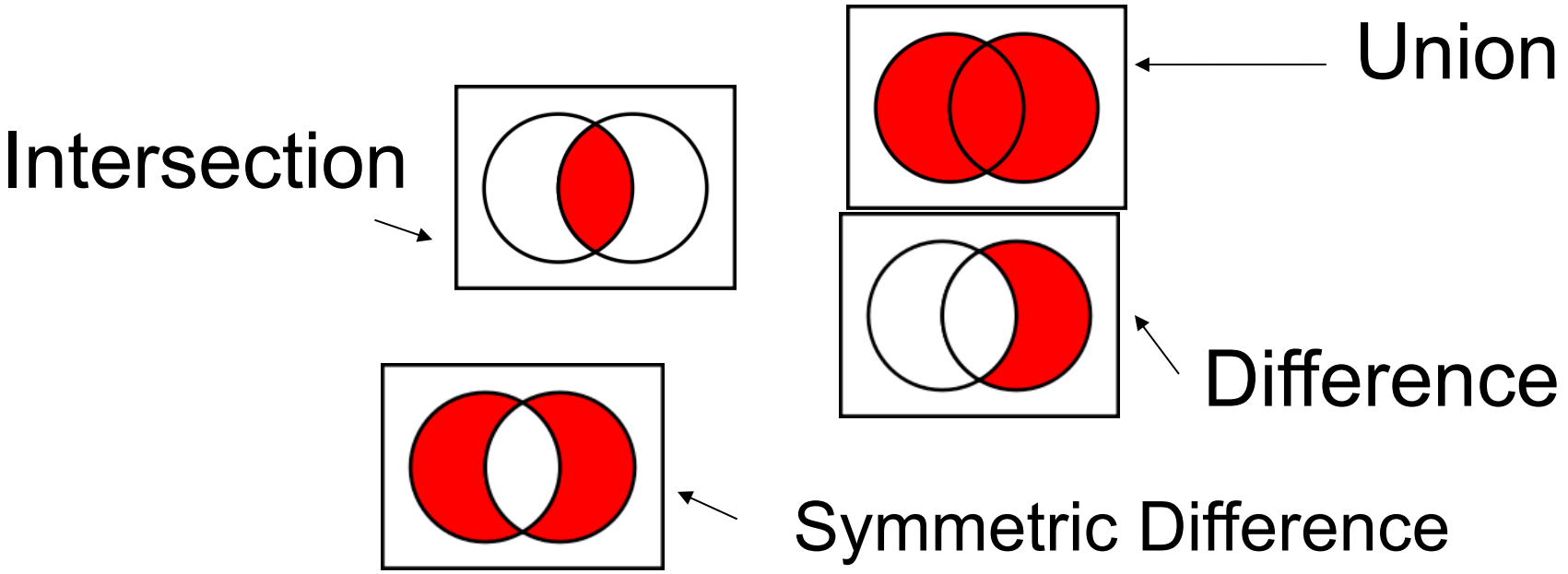
Sets

- Sets
- These are structures available in Python, used to represent unordered collections of unique elements

- s = {1, 2, 3, 4}
- print (s)

Sets

- In sets, a set of typical operations of mathematical set theory can be performed, such as:



Dictionary

- Add a new key-value pair

```
fruit[10] = 'pomegranate ’
```

Iterating through key-value pair

```
for key , value in fruit.items():
   print(‘The fluit' + str(key) +’ is ' + value)
```

Dictionaries

- Dictionaries store links between pieces of information
- Each item in a dictionary is a key-value pair
- Keys are not repeatable

```
fruit = {1: 'orange', 2: 'apple', 3: 'pear', 4: 'grape', 5: 'peach'}
```

Dictionary

- Iterating through the key

```
for key in fruit.keys():
    print(str(key) + ‘ is fluit’)
```

- Iterating through the values

```
for value in fruit.values():
    print(value + ' is fluit ')
```

Summary

Data Structure	Ordered	Mutable	Constructor	Example
List	Yes	Yes	`[ ]` or `list()`	`[5.7, 4, 'yes', 5.7]`
Tuple	Yes	No	`( )` or `tuple()`	`(5.7, 4, 'yes', 5.7)`
Set	No	Yes	`{}`* or `set()`	`{5.7, 4, 'yes'}`
Dictionary	No	Yes**	`{ }` or `dict()`	`{'Jun': 75, 'Jul': 89}`