Introduction to Computer Programming Lecture 3.2:

Data Structures

Hemma Philamore

Department of Engineering Mathematics

Introduction to Computer Programming Lecture 3.2.a:

Data Structures: Lists

Hemma Philamore

Department of Engineering Mathematics

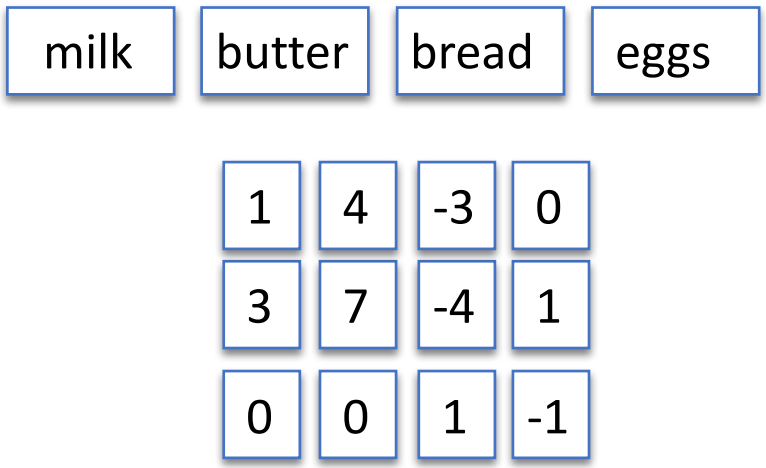
Limitations of simple variables

```
ShoppingItem = "milk"

XCoordinate = 3.5

StudentNumber = 123456
```

What if we need to deal with more complex data?



Learning Objectives

Data Structures

Lists

Tuples

Sets

Dictionary

Trees

Hash Tables

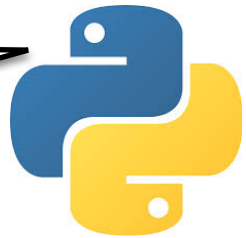
Queues

...

Why use different data structures

Why would you need more than lists?

Different problems, need different solutions!



Telephone Book

List 1

1	John Doe
2	Sarah Moor
3	Mary Salt
4	Kent Low

List 2

1	+555 7782
2	+555 4039
3	+555 9993
4	+555 1234

Dictionary

John Doe	+555 7782
Sarah Moor	+555 4039
Mary Salt	+555 9993
Kent Low	+555 1234

Learning Objectives

Data Structures

Lists

Tuples

Sets

Dictionary

Lists

Initialise with no elements

```
>>> ShoppingList = ["bread", "milk","eggs"]
>>> ShoppingList = []
>>> ShoppingList.append("bread")
>>> ShoppingList.append("eggs")
>>> ShoppingList.append("milk")
>>> ShoppingList
['bread', 'eggs', 'milk']
>>> ShoppingList[0]
'bread'
```

add to the end of the list

access individual items

```
>>> ShoppingList[0] = "rye bread"
>>> ShoppingList
['rye bread', 'eggs', 'milk']
```

edit (mutable), i.e. items can be changed

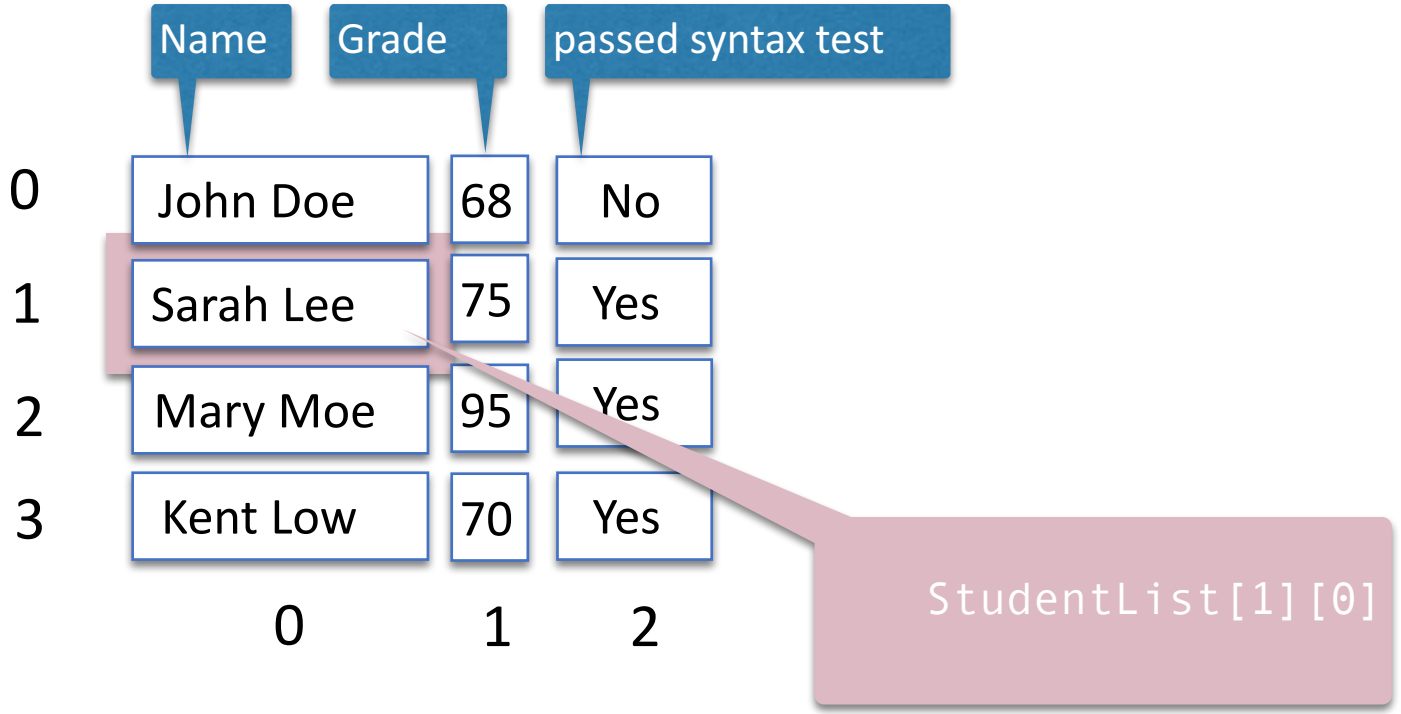
```
ShoppingList = ["butter","milk","bread"]
for item in ShoppingList:
    print(item)
```

you easily loop through a list (it's "iterable")

Nested Lists

List of lists *List* *List*

```
StudentList =  [["John Doe",68,False],["Sarah Lee", 75, True],....]
```




```
StudentList.append(["Mary Moe",95,True])
```

Nested Lists

```
>>> StudentList = [["John Doe", 68, False],["Sarah Lee", 75, True]]
>>> print(StudentList)
[['John Doe', 68, False], ['Sarah Lee', 75, True]]
```

```
>>> StudentList.append(["Kent Low", 70, True])
```

```
>>> print(StudentList[0][1])
68
```

"nested" index

```
for Entry in StudentList:
    for SubEntry in Entry:
        print(SubEntry)
```

nested loops

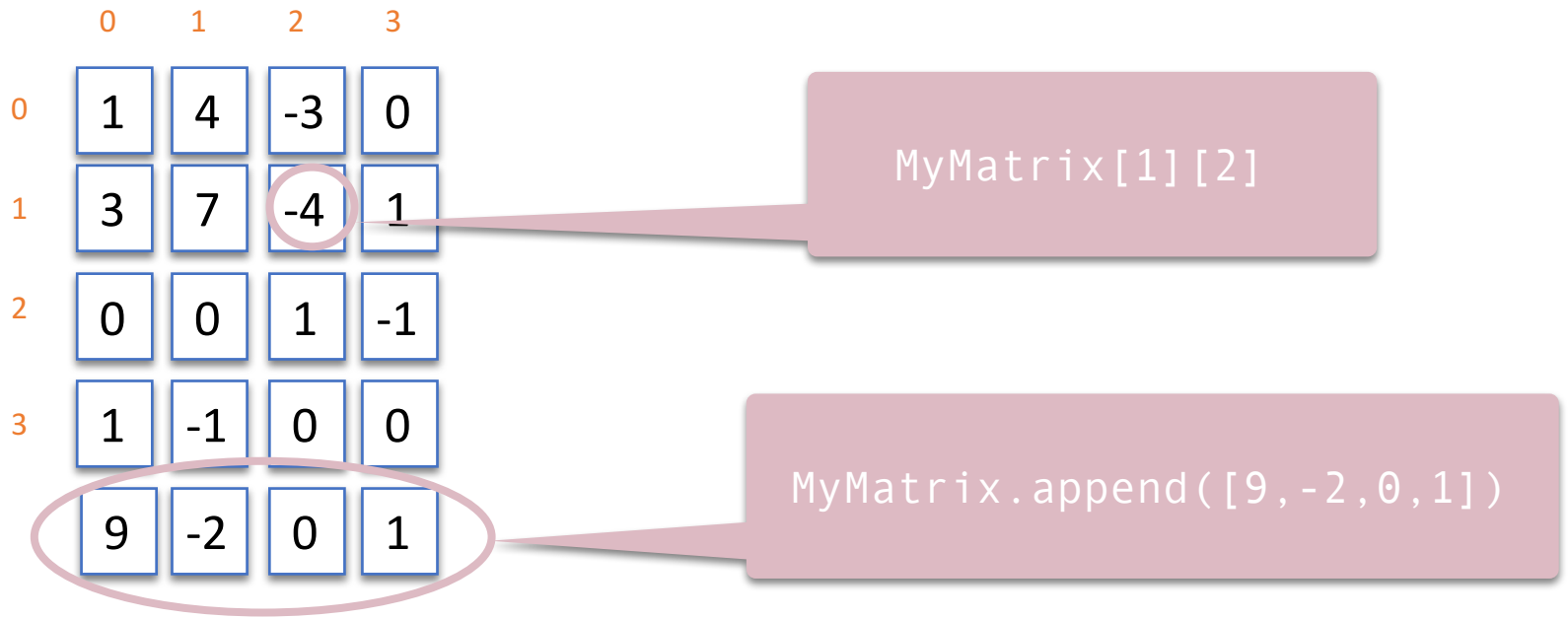
```
StudentList.append(45)
[['John Doe', 68, False], ['Sarah Lee', 75, True], 45]
```

Python lets you append anything.
You have to check as programmer!

Nested Lists

Application: Matrix

```
MyMatrix = [[1, 4, -3, 0], [3, 7, -4, 1], [0, 0, 1, -1], [1, -1, 0, 0]]
```



Nested Lists

Application: Matrix

```
MyMatrix = [[1, 4, -3, 0], [3, 7, -4, 1], [0, 0, 1, -1], [1, -1, 0, 0]]
```

```
>>> for Row in MyMatrix:
        print(Row)

[1, 4, -3, 0]
[3, 7, -4, 1]
[0, 0, 1, -1]
[1, -1, 0, 0]
```

```
>>> for Row in MyMatrix:
        for Item in Row:
            print(Item)

1
4
-3
0
3
7
-4
.
.
.
```

Introduction to Computer Programming Lecture 3.2.b:

Data Structures: List Comprehensions

Hemma Philamore

Department of Engineering Mathematics

List Comprehensions

Elegant way to create lists from sequences (rule-based).

1. **Make lists** where each element is the result of some operations applied to each member of the sequence.

2. **Create a subsequence** of those elements that satisfy a certain condition.

List comprehension is surrounded by brackets.

Instead of the list of data inside it, enter an expression followed by for loop and if-else clauses.

```
>>> # define a range
>>> xRange = [x for x in range(10)]
>>> print(xRange)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

print(type(xRange))
list
```

```
>>> # let the range start at 1
>>> xRange = [x+1 for x in range(10)]
>>> print(xRange)
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

List Comprehensions

```
#define a range
xRange = [x for x in range(10)]
print(xRange)

# to let range start at 1
xRange = [x+1 for x in range(10)]
print(xRange)

# to define a funcion
# e.g.: y = 3x^2+5
y = [3*x*x+5 for x in xRange]
print(y)

# nested lists are possible as well
xRangeSmall = [x+1 for x in range(5)]
newMatrix = [[x, x**2, x**3] for x in xRangeSmall]
print(newMatrix)
```

any offset is possible

```
# time vector with time step of 1 ms
t = [x/1000 for x in range(10)]
y = [x**2 for x in t]
```

time vector for plotting

List Comprehensions

```
# to find subsets
# e.g., only even numbers
# module operation finds the remainder after division
# e.g. 9 mode 3 = 0 and 11 mod 3 = 3
# hence x mod 2 = 0 for even numbers
evenNr = [x for x in xRange if x % 2 == 0]
print(evenNr)
```

How would you get odd only numbers?

```
# works also for strings
# e.g., to make sure all characters are upper case
Names =["JAMEs", "claire", "ChLOe"]
upperCase = [x.upper() for x in Names]
print(upperCase)
```

Can also work as a filter

```
# filter out lower numbers
Numbers = [3, 0, 2, 10, 7]
LowerNumbers = [x for x in Numbers if x <= 5]
print(LowerNumbers)
```

Introduction to Computer Programming Lecture 3.2.c:

Data Structures: Tuples

Hemma Philamore

Department of Engineering Mathematics

Learning Objectives

Data Structures

Lists

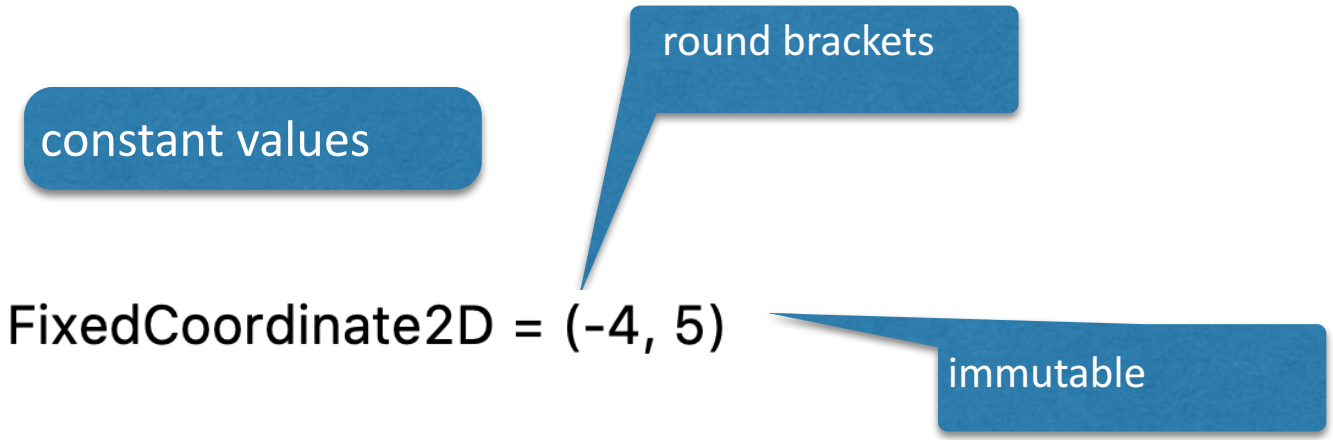
Tuples

Sets

Dictionary

Tuples

- Like lists, tuples are sequences of ’boxes’ that store data.
- Unlike lists, tuples are **immutable** (can **not** be changed)



Note: Tuples are lists with **functionality removed** — why do this?
It is an example of syntactic salt — “a feature designed to make it harder to write bad code”.

Tuples

```
FixedCoordinates2D = (-4,5)
print(FixedCoordinates2D)

print(FixedCoordinates2D[1])

FixedCoordinates2D[0] = 3

NestedTuple = ( (4,"Hello"), (7, "OK"))
print(NestedTuple)

TupleOfCharacters = tuple("Hello")
```

access item with squared brackets

produces error immutable

casting works here as well

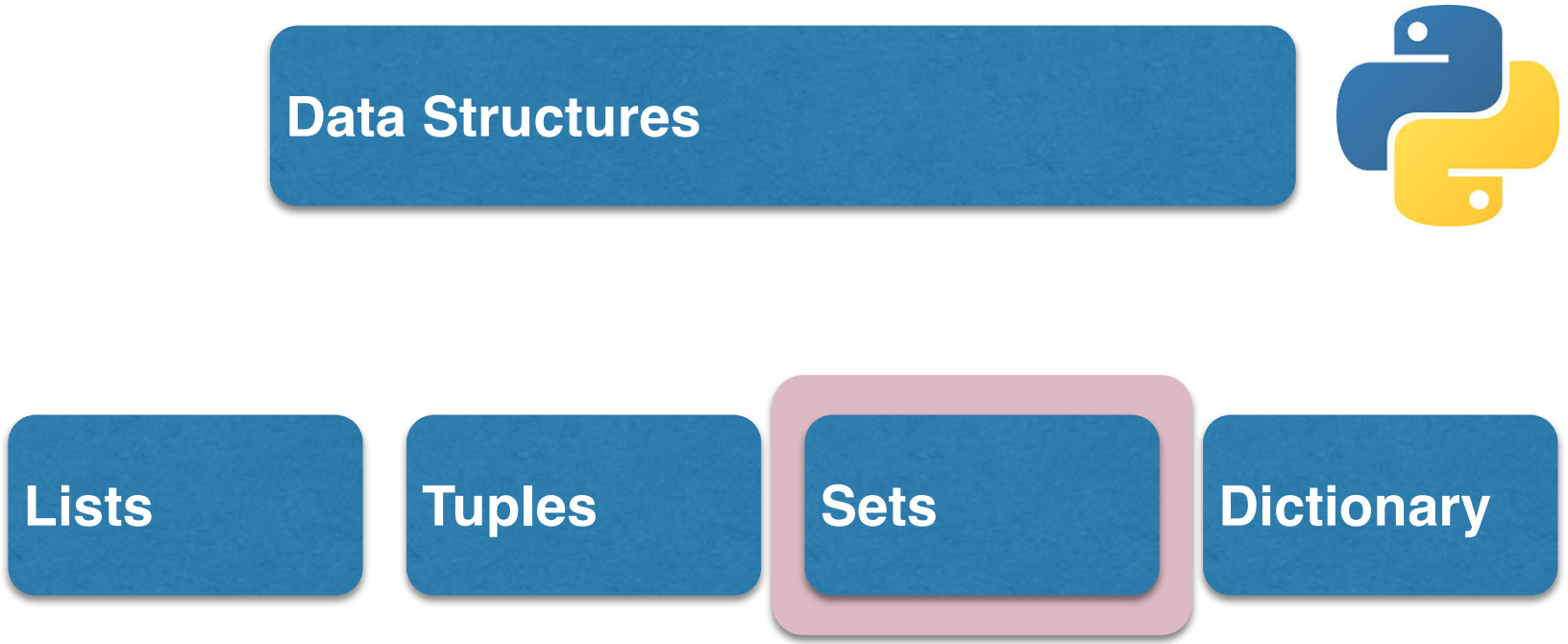
Introduction to Computer Programming Lecture 3.2.d:

Data Structures: Sets

Hemma Philamore

Department of Engineering Mathematics

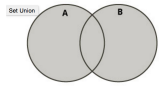
Learning Objectives



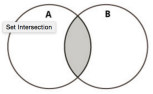
Sets

- Sets are **unordered** collections with **no duplicate** elements.
- You can **test membership** of a set (“is element in set”)
- Supports **mathematical operations**

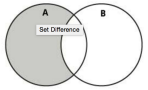
```
>>> FirstSet = {1, 2, 8, 6, 7}
>>> SecondSet = {5, 4, 6}
```



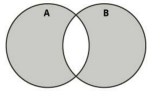
union (|) {1, 2, 8, 6, 7, 5, 4}



intersection (&) {6}



difference (-) {1, 2, 8, 7}



sym. diff (^) {1, 2, 8, 7, 5, 4}

Sets

order is irrelevant!

ignores double entries

```
>>> FirstSet = {1, 2, 3, 4, 4, 5}
>>> FirstSet
{1, 2, 3, 4, 5}
>>> SecondSet = {1, 2, 2, 7, 8}
>>> SecondSet
{8, 1, 2, 7}
>>> 3 in FirstSet
True
>>> 3 in SecondSet
False
>>> FirstSet - SecondSet
{3, 4, 5}
>>> FirstSet | SecondSet
{1, 2, 3, 4, 5, 7, 8}
>>> FirstSet & SecondSet
{1, 2}
>>> FirstSet ^ SecondSet
{3, 4, 5, 7, 8}
>>> FirstSet[2]
Traceback (most recent call last):
  File "<pyshell#62>", line 1, in <module>
    FirstSet[2]
TypeError: 'set' object does not support indexing
```

no order - no index

Introduction to Computer Programming Lecture 3.2.e:

Data Structures: Dictionaries

Hemma Philamore
Department of Engineering Mathematics

Learning Objectives

Data Structures

Lists

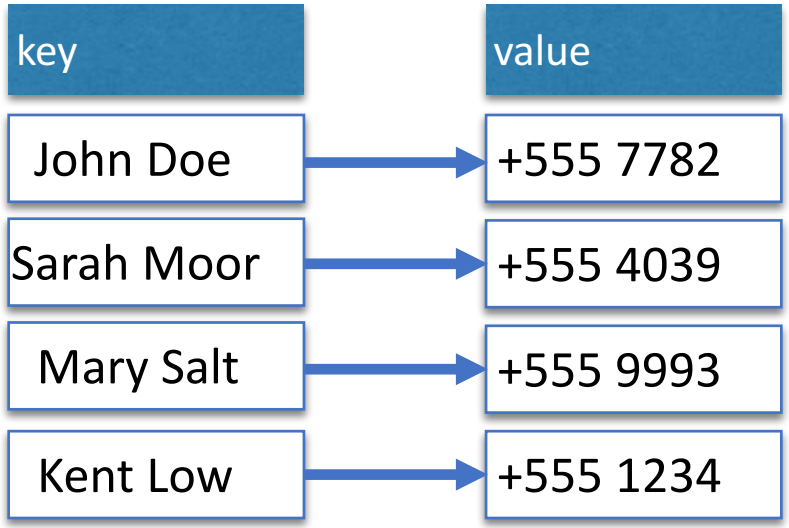
Tuples

Sets

Dictionary

Dictionary

Dictionaries are **unordered sets** of data that have a **key** and a **value** (an important idea in databases).



- Unlike sequences (lists, tuples), which are accessed by numerical indices (0,1,2,3...), dictionaries are **indexed by keys**.

- Keys must be **unique** and **immutable** (often numbers or strings).

Dictionary

```
States = {
  "Oregon": "OR",
  "Florida": "FL",
  "California": "CA",
  "New York": "NY"
  }

# add a new entry
States["Michigan"] = "MI"
print(States)

# delete an entry
del States["New York"]
print(States)

# list the keys
print(States.keys())

# you can even sort them
print(sorted(States.keys()))

# check for entries
print("Florida" in States)
print("Texas" in States)

# loop through dictionary
for Name, Abbr in States.items():
  print(Name, Abbr)
```

True or False –
can be used for conditions

You get both key
and values