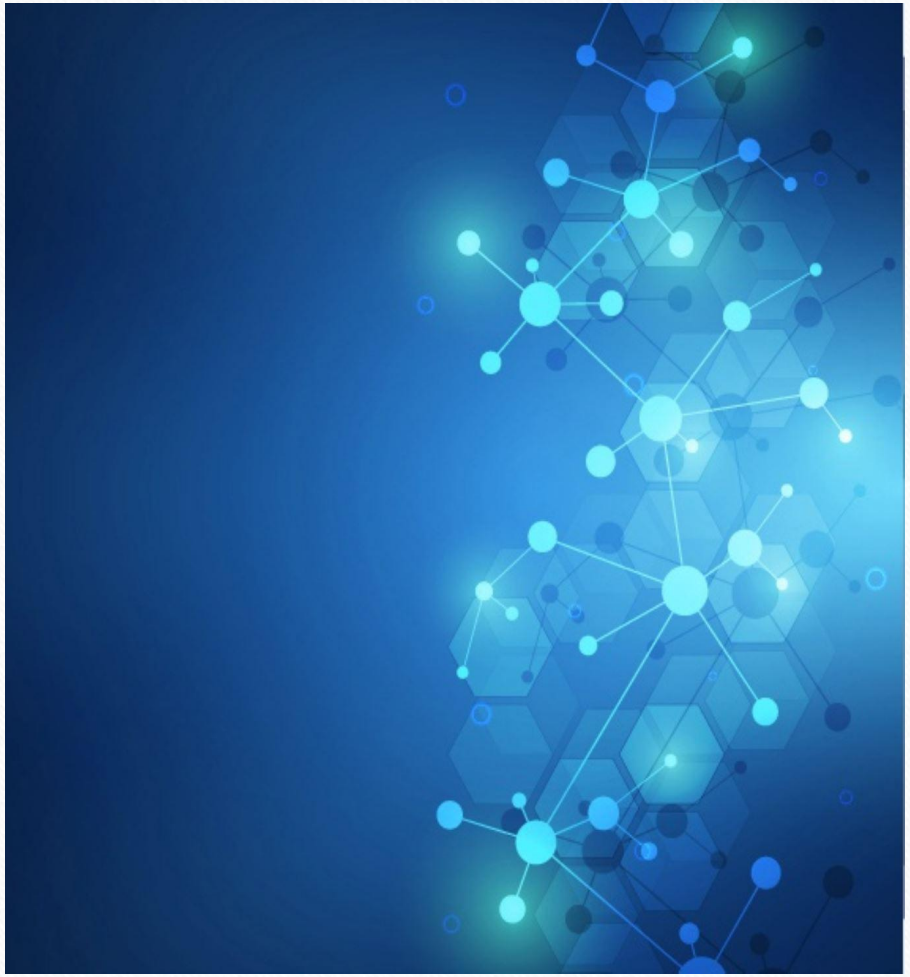
MSE 800

- Professional Software Engineering



Python Lists

- Many computer programs handle collections of data
  – a list of students, a sequence of temperature samples, an array of image pixels, set of university courses, a table of measurements ...

- Most such collections can be represented in Python by its list data type.
- A list is a sequence of objects that can be processed sequentially.
- The Python list also allows immediate access to any element by subscripting, for example, *marks[i]* for the *i*th mark
  – In maths notation, we would write this as $marks_i$
  – So a Python list is both a list and an array

Some examples of lists

```
days_in_month = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
guys = ["Freddie", "Brian", "Roger", "John"]
colours = ["Red", "Green", "Blue"]
squares = [0, 1, 4, 9, 16, 25, 36, 49]
square_roots = [1.0, 1.4142135623, 1.732050807, 2.0, 2.236067977]
intelligent_life_forms_on_earth = []                    # The empty list
personal_details = ["Erika Mandelbrot", 27, "5 Nowhere St, Christchurch"]
```

Length=8

A list of objects of different types.
Legal in Python but bad style.
We will see better ways of representing such “records” later.

Indexing into lists

- To use lists we need to be able to get at the individual elements
- Do by indexing, for example:

```
print(days_in_month[0])                    # Prints 31
print(colours[2])                          # Prints "Blue“
```

  - subscripts start at 0!!

```
print(squares[len(squares) – 1])           # Prints 49
```

  - *len* function returns the number of items in a list

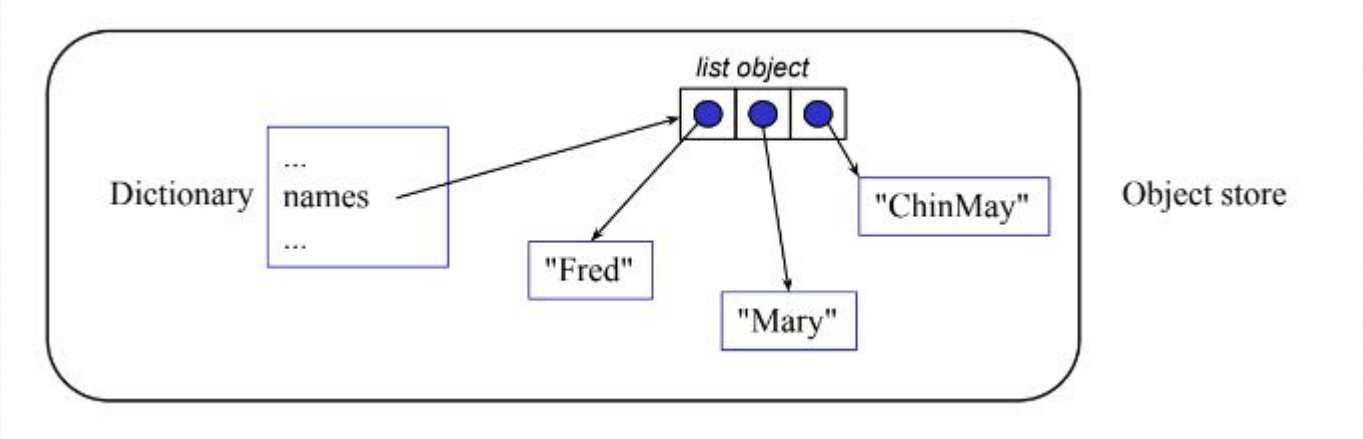
```
print(squares[-1])                         # Also prints 49
```

  - If subscript is negative, Python adds len(list) to it

```
print(squares[-2])                         # Prints 36
```

How lists are represented

- names = ["Fred", "Mary", "ChinMay"] results in:

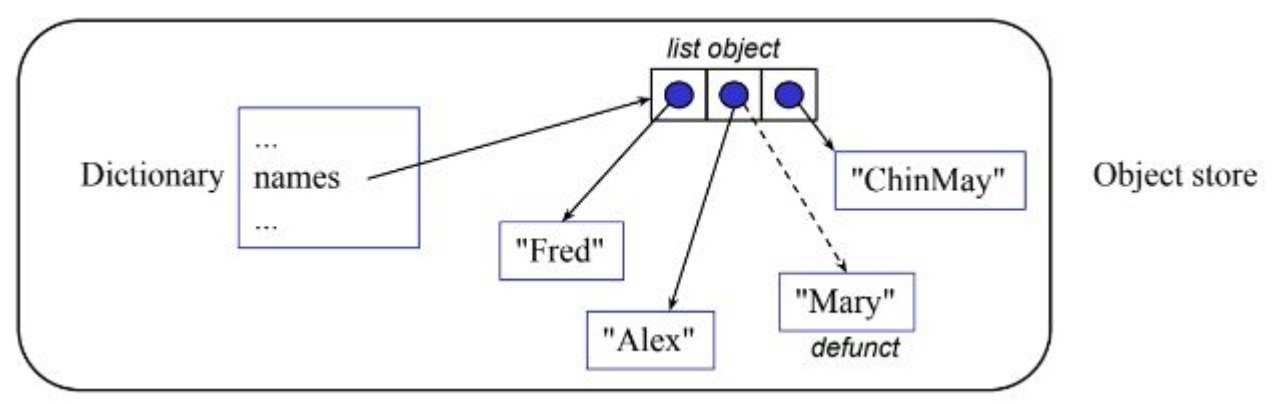


- The list object itself is just a list of references to the objects in the list.

Changing list elements

names = ["Fred", "Mary", "ChinMay"]
names[1] = "Alex"

results in:



- The list element is changed – we do not get a new list
- We say list objects are mutable (= "changeable")
  - c.f. string which are immutable

List slicing

- Often want sublists rather than individual items
- Done by extended indexing of the form "start:end+1"
  - Missing first subscript defaults to 0
  - Missing second subscript defaults to len(list)
- Called "slicing"
- Examples: squares = [0, 1, 4, 9, 16, 25, 36, 49]
  - print(squares[2:4]) # Prints "[4, 9]"
    - Note that slice is up to but not including the second subscript
  - print(squares[:4]) # Prints "[0, 1, 4, 9]"
  - print(squares[3:]) # Prints "[9, 16, 25, 36, 49]"

Assigning to slices

- my_list[start:end] = another_list replaces the elements *my_list[start]* up to but not including *my_list[end]* with the elements from *another_list*

- **Example:**

```
my_list = [1, 3, 5, 7, 9, 11]
my_list[2:4] = [-3, -9, -11, -13]
print(my_list)
```

This prints [1, 3, -3, -9, -11, -13, 9, 11]

- Can do insertion too (but insert method easier to read?):

- **Example:**

```
my_list = [1, 3, 5]
my_list[1:1] = [-3, -9] # my_list is now [1, -3, -9, 3, 5]
```

assigning to such an empty slice range, actually insert rather than replacing

List operators

- list1 + list2 is a list of all the elements from list1 followed by all the elements from list2
  - Called concatenation
  - for example, [1, 2, 3] + [7, 8] is [1, 2, 3, 7, 8]

- my_list * n or n * my_list, where n is an int, is a new list containing n repetitions of the sequence of items in my_list
  - 3 * ['Max', 'Amy'] is ['Max', 'Amy', 'Max', 'Amy', 'Max', 'Amy']

- object in list evaluates to True if the object is in the list
  - for example, 3 in [1, 3, 5] is True and 2 in [1, 3, 5] is False

List functions

- len(my_list) is the length of my_list
  - for example, print(len([1, 2, 3])) prints 3

- sum(my_list) sums the elements of my_list
  - for example, print(sum([1, 2, 3])) prints 6
  - List items must be numeric
    - Cannot do string concatenation this way
      - But str.join can be used. Look it up!

- min(my_list) and max(my_list) return min and max elements in a non-empty numeric list
  - for example, max([-3, 13, 5]) is 13

```
strings = ['Hello', 'World', 'Python']

print(' '.join(strings))          # Output: Hello World Python

print(','.join(strings))          # Output: Hello,World,Python

print('\n'.join(strings))
# Output:
Hello
World
Python

print('-'.join(strings))          # Output: Hello-World-Python

print(''.join(strings))           # Output: HelloWorldPython
```

List methods

If **L** is a list:

```
L.append(object)          # Adds object to end of L.
L.count(value)            # Returns count of items in L equal to value
L.extend(L2)              # Appends all the items from L2 onto L.
L.index(value)            # Returns the index of the first occurrence of value in L
    o Gives an error if value not found
L.insert(index, object)   # Insert object into L before index.
L.pop([index])            # Remove and return object at index (defaults to last)
L.remove(value)           # Remove first occurrence of value.
L.reverse()               # Reverse list L.
L.sort()                  # Sort L in ascending order.
```

```python
L = [1, 2, 2, 3]
L2 = [4, 5]

L.append(4)  # L is now [1, 2, 2, 3, 4]
last_item = L.pop() # L is now [1, 2, 2, 3]
index = L.index(2)  # index is 1
count = L.count(2)  # count is 2
L.extend(L2)  # L is now [1, 2, 2, 3, 4, 5]
combined_list = L + L2  # L is now [1, 2, 2, 3, 4, 5]
# combined_list = [1, 2, 2, 3, 4, 5, 4, 5]
L.insert(1, 2) # L is now [1, 2, 2, 2, 3, 4, 5]
L.remove(2)  # L is now [1, 2, 2, 3, 4, 5]
L.pop(2)    # L is now [1, 2, 3, 4, 5]
```

A trap!

- What will the following output?

```
names = ["Fred", "Mary", "ChinMay"]
other_names = names
names.append("Angus")
print("Names:", names)
print("Other names:", other_names )
```

- Answer:

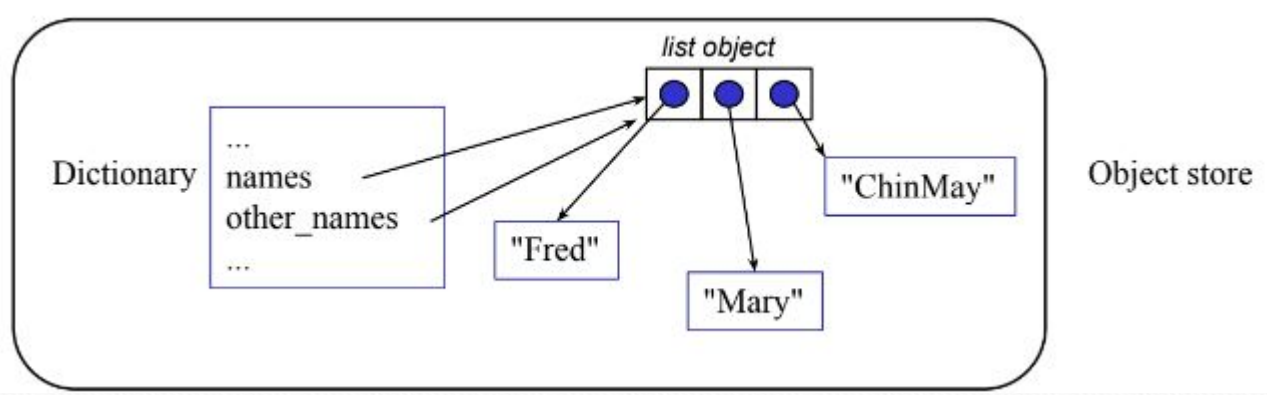
  Names: ['Fred', 'Mary', 'ChinMay', 'Angus']

  Other names: ['Fred', 'Mary', 'ChinMay', 'Angus']

- Both lists were altered!

Why it happened: aliasing

- Assignment of one object to another just copies the reference.
- So after other_names = names we have:



both actually reference the same object in memory.

- So names and other_names are just aliases for the same object.

Whenever one changes, the other changes too.

Avoiding aliasing problems

- Be wary of assignments of the form a = b when b is a mutable object, that is, one whose value can be changed, as any changes will apply to all aliases.
  - Not a problem with ints, floats, strings, and tuples as they are all immutable.

- If you want to make a copy of a list, use slicing, for example, other_names = names[:]
  - This constructs a new list containing copies of all the references. Called a shallow copy.
    - There can still be aliasing problems if the referenced objects are mutable but we will not worry about that for now!

Lists and Tuples (recap)

Lists are mutable collections of objects (in brackets)

```
fruits_list = ["apple", "banana", "cherry"]
print(fruits_list)

# You can modify the list after creation
fruits_list.append("orange")  # Add a new element
print(fruits_list)

# Output:
# ["apple", "banana", "cherry"]
# ["apple", "banana", "cherry", "orange"]
```

Tuples are immutable collections of objects (in parentheses)

```
colors_tuple = ("red", "green", "blue")
print(colors_tuple)

# Trying to modify the tuple will result in a TypeError
# colors_tuple.append("yellow")  # This will raise an error
```

Dictionaries

- Dictionaries define key/value pairs
- The keys form a set
  - Any key can appear once at most
  - Keys must be immutable
  - Ordered by insertion-time (since Python 3.6)
- Values can change
- Construct with curly braces { }, colons, and commas

```
>>> bird_counts = {'kiwi' : 3, 'weka': 1, 'kereru': 7}
>>> bird_counts['kiwi']
3
```

- Type name is 'dict'

```
>>> bird_counts = {'kiwi' : 3, 'weka': 1, 'kereru': 7}
```

Dictionaries – Basics

- {} is an empty dictionary (not an empty set!)
- Accessing a non-existent key is an error

```
>>> bird_counts['puffin']            # OOPS!
Traceback (most recent call last):
  File "<string>", line 1, in <fragment>
KeyError: 'puffin'
```

- Is the key in the dictionary? Use in

```
>>> if 'kiwi' in bird_counts:
     print('kiwi have been seen')
```

- Adding a key/value pair, or reassigning a value

```
>>> bird_counts['piwakawaka'] = 42
```

- Deleting a key (and its value). Use del or pop

```
>>> del bird_counts['piwakawaka']
>>> bird_counts.pop('kiwi')
```

Dictionary methods

- clear Empties the dictionary

```
>>> d.clear()
```

- get Returns the value associated with the key, or an optional default if the key is not present

```
>>> bird_counts = {'kea':42, 'weka':14, 'kiwi':56}
>>> bird_counts.get('kea')
42
>>> bird_counts.get('kereru', 99)
99
```

Dictionary methods (cont'd)

- keys Returns a list-like object of the dictionary keys

```
>>> bird_counts.keys()
dict_keys(['kiwi', 'weka', 'kea'])
```

- items Returns a list-like object of key/value pairs

```
>>> bird_counts.items()
dict_items([('kiwi', 56), ('weka', 14), ('kea', 42)])
```

- Break

- Q & A
  - (warm up)

1. What will be the data type of the variable x after this assignment: x = 3.5?

A int

B float

C str

D complex

2. Pseudocode:

Variable
x = "Python";
Check if x is a string,
if yes
print "String",
otherwise
"Not a String"

A String

B Not a String

C Error

D None

3. Evaluate this pseudocode:
Set x = [1, 2, 3];
If x is a list, print length of x, else print "Not a list"

A 3

B Not a list

C Error

D 0

4. What is the purpose of the end parameter in the print() function?

A To add a space at the end

B To end the script

To specify the string appended after the last value

D To break the line

5. What is the purpose of an if statement in Python?

A To loop through a sequence

B To execute a block conditionally

C To define a function

D To handle exceptions

6. In Python, which keyword is used to check additional conditions if the previous conditions fail?

A elif

B else if

C then

D switch

7. Which of the following is a valid conditional statement in Python?

A if a = 5:

B if a == 5:

C if a <> 5:

D if (a = 5):

8. How do you access the last element of a list named myList?

A myList[0]

B myList[-1]

C myList[len(myList)]

D myList[-2]

9. In Python, how can you combine two lists, list1 and list2?

A list1 + list2

B list1.append(list2)

C list1.combine(list2)

D list1.extend(list2)

10. What does myList[::-1] do?

A Reverses myList

B Copies myList

C Removes the last element from myList

D Sorts myList in descending order

- Exercises

Rules

- No Chatgpt
- No questions and No assistance from others
  - Self-learning capability
  - You need to learn how to solve complex problems on your own when faced with complex problems. For example, how to quickly find solutions online
  - The task may beyond the scope of your knowledge, Try.
- Can check online resources or lecture notes
- Solutions will be given later

BMI Calculator and Interpretation

Requirements:

- Build a BMI (Body Mass Index) calculator that computes the BMI score based on a person's weight and height.
- Use conditional statements to interpret the BMI score into categories such as Underweight, Normal weight, Overweight, and Obese.
- Set the BMI classification thresholds as follows:
  - Underweight: less than 18.5
  - Normal weight: 18.5 to 24.9
  - Overweight: 25 to 29.9
  - Obese: 30 or more
- Print out the person's BMI score and interpretation.

•Exercise2 : **Grade Classifier**

Objective: Create a program that takes students' scores as input and assigns a grade based on the score. The grades should be A, B, C, D, or F.

Requirements:

- Ask for user input(format: [score1,score2,score3,…])
- Utilize a list to store scores and their corresponding grades.
- Iterate over the list of scores using a loop.
- Use comparison operators within conditional statements to determine the appropriate grade for each score.
- Print each student's score (keep 1 place after point) along with their respective grade.

A: 90 and above

B: 80 to 89

C: 70 to 79

D: 60 to 69

F: below 60

Exercise3 : Simple Book Management System

Objective: Write a program to help users manage their personal book collection. The program should allow the user to add, remove, and search for books.

```
"ADD The Great Gatsby, F. Scott Fitzgerald"
```

Requirements :

- **User Input**: The user will input commands like "ADD title, author", "REMOVE title", or "SEARCH title".
- **Book List**: The program should maintain a list of books, where each book is represented by a dictionary containing the book's title and author.
- **Adding Books**: When adding a book, the program should check to see if the book already exists in the collection.
- **Removing Books**: When removing a book, the program should verify that the book is in the list.
- **Searching for Books**: When searching for a book, if found, the program should display "Book found: title by author". If the book is not found, it should display "Book not found".
- **Error Handling**: If the user enters an incorrect command format, the program should prompt them with "Invalid input. Please use ADD, REMOVE, or SEARCH followed by the book title and author."

Exercise4: Expense Tracker

Objective: Create a program to help users manage and analyze their personal expenses by categories over a month.

Refined Requirements:

- The program should have predefined categories: 'Food', 'Utilities', 'Entertainment', 'Transportation', 'Healthcare'.
- The user can add expenses by specifying a category and an amount.
- The user can request the total expenses for a specific category.
- The user can request the average expense for each category.
- The program should prevent the user from entering expenses into undefined categories.

Features to Use:

- Dictionary with predefined categories as keys, and the values as lists that store expenses.
- Functions for:
  - Adding expenses to categories
  - Calculating total expenses for a specific category
  - Calculating total and average expenses for all categories
- Input validation to ensure correct category usage.
- Exception handling for invalid inputs (e.g., non-numeric expense amounts).

- Thank you