# Welcome to the CoGrammar Multidimensional Lists

The session will start shortly...

Questions? Drop them in the chat. We'll have dedicated moderators answering questions.

CoGrammar

## Software Engineering Session Housekeeping

- The use of disrespectful language is prohibited in the questions, this is a supportive, learning environment for all - please engage accordingly. **(Fundamental British Values: Mutual Respect and Tolerance)**
- No question is daft or silly - **ask them!**
- There are **Q&A sessions** midway and at the end of the session, should you wish to ask any follow-up questions. Moderators are going to be answering questions as the session progresses as well.
- If you have any questions outside of this lecture, or that are not answered during this lecture, please do submit these for upcoming Academic Sessions. You can submit these questions here: **Questions**

CoGrammar

## **Software Engineering Session Housekeeping** cont.

- For all **non-academic questions**, please submit a query: [www.hyperiondev.com/support]

- Report a **safeguarding** incident: [www.hyperiondev.com/safeguardreporting]

- We would love your **feedback** on lectures: **Feedback on Lectures**

## Skills Bootcamp
## 8-Week Progression Overview

### Fulfil 4 Criteria to Graduation

✅ **Criterion 1: Initial Requirements**

- ***Timeframe:*** First 2 Weeks
- ***Guided Learning Hours (GLH):*** Minimum of 15 hours
- ***Task Completion:*** First four tasks

**Due Date: 24 March 2024**

✅ **Criterion 2: Mid-Course Progress**

- ***Guided Learning Hours (GLH):*** 60
- ***Task Completion:*** 13 tasks

**Due Date: 28 April 2024**

CoGrammar

# CoGrammar

## Lists

April 2024

## Learning Objectives and Outcomes

- ❖ Understand a bit of memory management with respect to lists
- ❖ Understand differences between an array and a list
- ❖ Define a python list and index
- ❖ Understand the different lists operations
- ❖ Define and implement 1D and higher dimensional lists

CoGrammar

## Problem Statement

Picture organizing your bookshelf with various genres of books. In Python, lists act like shelves, helping you group similar items together. For instance, you can create a list of "fiction" books or "non-fiction" books. This makes it easy to manage and access your collection efficiently. Let's dive into organizing with Python lists!

CoGrammar

## Agenda

- ❖ Lists Fundamentals
- ❖ 1D Lists
- ❖ ND Lists

CoGrammar

## Lists Fundamentals

CoGrammar

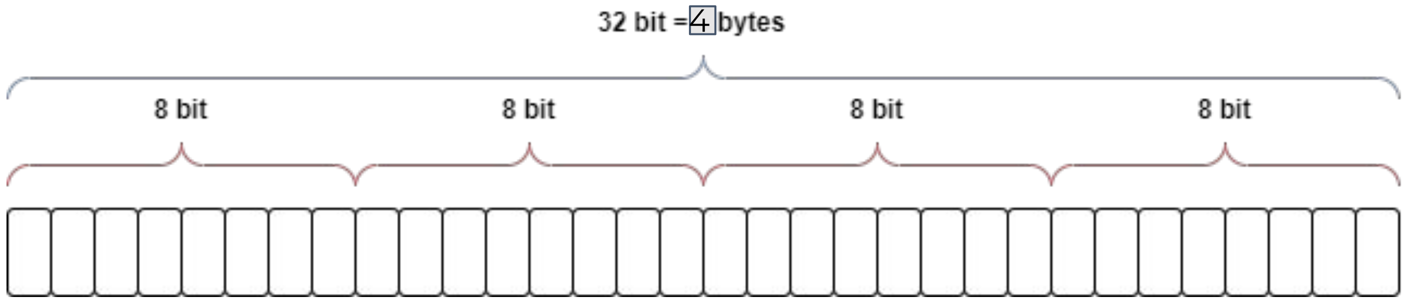
# Data Types and Sizes

| | | Empty | | | |
|---|---|---|---|---|---|
| | | **C++** | **Python** | **C++ Notes** | **Python Notes** |
| Varibles | boolean | 1 byte | 24 bytes | | But, **True (1)** has **28 bytes** and **False(0) has** 4 bytes less, **24 bytes** |
| | char | 1 byte | 2 bytes | | **But**, a character in python is already a string, no difference. **49 bytes** |
| | int | 4 bytes | 24 bytes | | |
| | float | 4 bytes | 24 bytes | | |
| | double | 8 byte | NA | | double is not primitive to python |
| | string | 32 bytes | 49 bytes | | +1 byte per additional character (49 + total length of characters) |
| | None (Null) | 8 bytes | 16 bytes | | |
| Containers | list (array) | 0 byte | 40 bytes | If array empty, size is 0 | +16 per additional item in a list ( 40 + 8*total length of items ) |
| | tuple | 0 byte | 24 bytes | If array empty, size is 0 | +8 per additional item in a list ( 24 + 8*total length of items ), including another container |
| | set | 0 byte | 200 bytes | If array empty, size is 0 | 0-4 takes a size of 200. 5-18 size will be 712. 19th will take 2248 and so on... |
| | dict (map) | 0 byte | 48 bytes | If array empty, size is 1 | Once filled is from 1 to 5 key-value pairs **216 bytes**, then, **344 bytes** (16 bytes more) |

CoGrammar

## Unit of measure

The primary memory of a computer (**RAM)** is composed of bits of information, and those bits are typically grouped into larger units that depend upon the precise system architecture. Such a typical unit is a **byte**, which is equivalent to **8 bits**



CoGrammar

## Arrays (C++)

Linear data structure that collects elements of the **same data type** and stores them in **contiguous** and **adjacent memory locations.**

In RAM

| | | | | |
|---|---|---|---|---|
| 0x7FFD90 | 0x7FFD91 | 0x7FFD93 | 0x7FFD94 | 0x7FFD95 |
| ‘A’ | 9 | | | |

Legend

| | | |
|---|---|---|
| Memory Location | Character | integer |

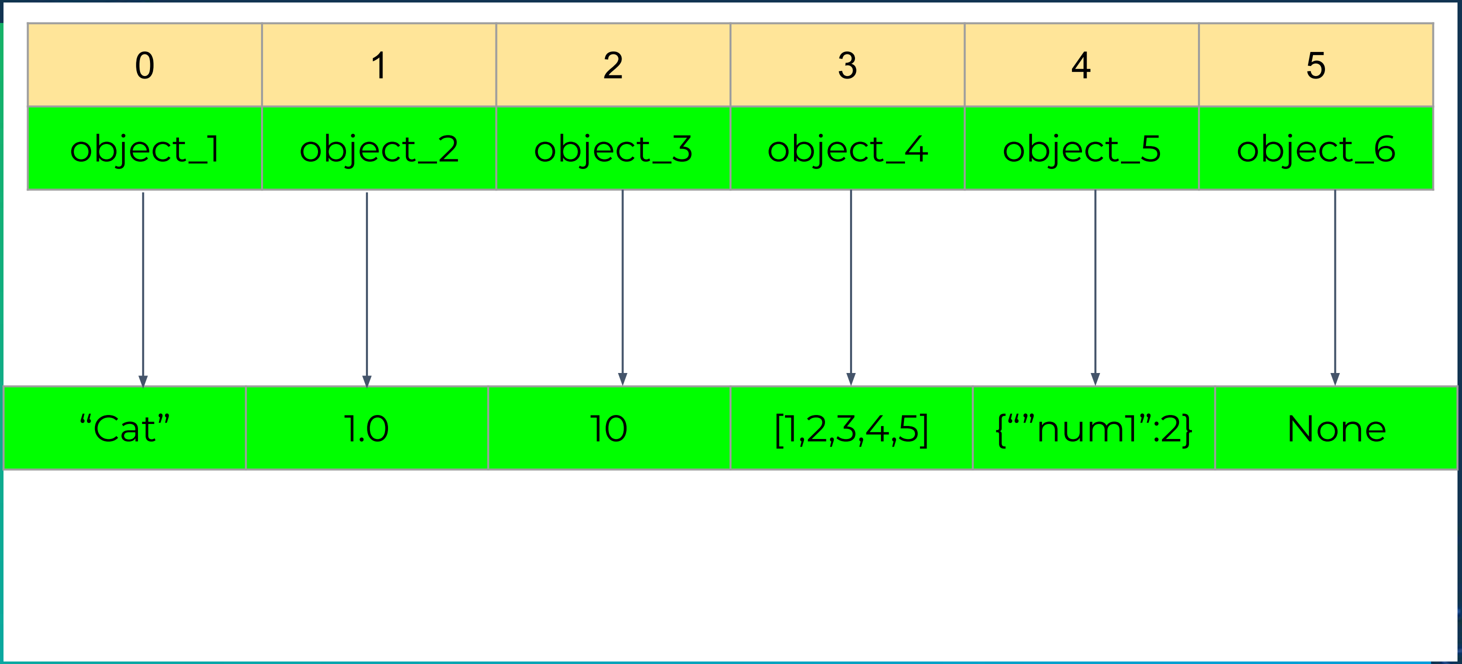
## Referential Arrays

- As opposed to other programming languages like C++ or Java, **Python can receive any type of variable types in a list.**
- Each cell in a list, stores the reference of each number item inserted in it. Then it terms of inserting, retrieval and removal are done is quicktime, or constant time (for later)

| 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| “Cat” | 1.0 | 10 | [1,2,3,4,5] | {“”num1”:2} | None |

CoGrammar

# Referential Arrays



CoGrammar

## Definitions

- **A container** is a construct used to group related values together and contains references to other objects instead of data.
- **A list** is a container created by surrounding a dynamically typed array sequence of variables or literals with brackets [ ] or list().
- **An element** or a call is a list item
- **Index:** in a list refers to the position of an element within the list. Usually starts from 0
- **Mutability**: ability of modify a data structure at runtime. A list is mutable data structure in Python

CoGrammar

## 1D Lists

CoGrammar

# 1D Lists: Definition

**A list** is a container created by surrounding a dynamically typed array sequence of variables or literals with brackets [ ] or list().

```
>>> myList = [“cat”, 1.0, 10, [1,2,3,4,5], {“”num1”:2}, None]
>>> myList[2]
[1,2,3,4,5]
>>> my_list = [ ]          #or   my_list = list()    creates an empty list
```

10 is a position 2

CoGrammar

# 1D Lists: Operations

## Adding an element in a list: **append()**

my_list = list() → Empty list

| |
|---|
| |

To add **3** to the list, then **5**

my_list**.append(3)** → 3 added to list

| 0 |
|---|
| 3 |

my_list**.append(5)** → 5 added to list

| 0 | 1 |
|---|---|
| 3 | 5 |

CoGrammar

# 1D Lists: Operations

Removing an element in a list: **pop()**

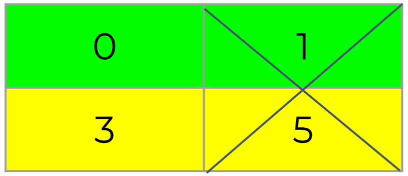
my_list = list()

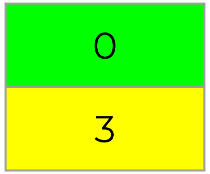
To add **3** to the list, then **5**

my_list**.append(3)**

my_list**.append(5)**

my_list.pop() # => returns **5**





CoGrammar

# 1D Lists: Operations

## Updating a cell in a list: **update**

my_list = list()

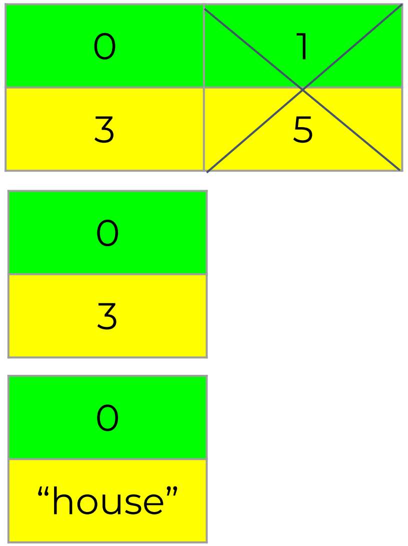
To add **3** to the list, then **5**

my_list**.append(3)**

my_list**.append(5)**

my_list.pop() # => returns **5**

My_list[0] = “house”



CoGrammar

# 1D Lists: Operations

## Extending the list: **extend()**

my_list[0] = “house”

your_list = [“Monday”, True]

| 0 |
|---|
| “house” |

Beware!

# extend() is an **inplace** function

my_list.extend(your_list)

| 0 | 1 | 2 |
|---|---|---|
| “house” | “Monday” | True |

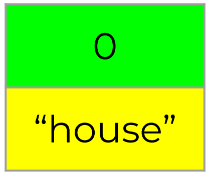
CoGrammar

# 1D Lists: Operations

## Extending the list: **+ (extend)**

my_list[0] = “house”

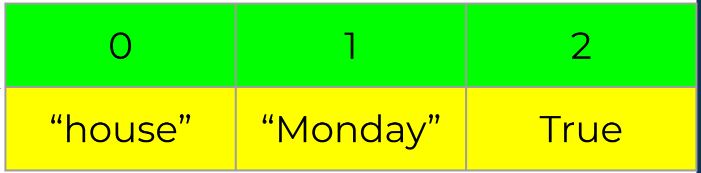
your_list = [“Monday”, True]



Beware!

# + not is an **inplace** operation

new_list = my_list **+** your_list



CoGrammar

# 1D Lists: Operations

## * (Repeat) and List Comprehension

**Repeat**

```
let new_list = [None]

let counter = 10

# To have 10 slots in new_list with None

>>> new_list * counter
[None, None, None, None, None, None, None, None, None]
```

CoGrammar

# 1D Lists: Operations

## * (Repeat) and List Comprehension

**List Comprehension** (Compressed **for loop**)

```
>>> counter = 10

>>>let new_list = [x for x in range(counter)]
```

result

iterator

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

## 1D Lists: Operations

### Slicing

```
>>> new_list  = [start:end:step]

>>> new_list = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

>>> new_list[0:5:2]

[0, 2, 4]

>>> new_list[::-1] # reversing the list
[9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
```

CoGrammar

## 2D Lists

CoGrammar

# 2D Lists: Definitions

## Definitions

- 2D Lists extension of 1 D List
- Each cell is an object referring to another Python list
- Two-dimensional lists, often referred to as 2D lists or matrices
- Nested Lists

CoGrammar

# 2D Lists: Operations

## Access

```
>>> new_list  = [[1.0,”cat”,3], [4,”fish”,6], 7,”hen”,9.0]
```

- As in a 1D list, we have indices
- 1 index for the row
- 1 index for the column

| | 0 | 1 | 2 |
|---|---|---|---|
| 0 | 1.0 | “cat” | 3 |
| 1 | 4 | “fish” | 6 |
| 2 | 7 | “hen” | 9.0 |

To access “fish”

```
>>>new_list[1][1]
“fish”
```

CoGrammar

## 3D Lists

CoGrammar

## 3D Lists: Definitions

### Definitions

- 3D Lists extension of 2D List
- Each cell is an object referring to another Python list, which also refers to another list
- Three-dimensional lists, often referred to as 3D lists or **matrices**
- Nested Lists

CoGrammar

## 3D Lists: Operations

### Access

```
>>> new_list  = [[['#', '#', '#'], ['#', '#', '#'], ['#', '#', '#']],
                 [['#', '#', '#'], ['#', '#', '#'], ['#', '#', '#']],
                 [['#', '#', '#'], ['#', '#', '#'], ['#', '#', '#']]]
```

- As in a 2D list, we have indices
- 1 index for the row
- 1 index for the column
- 1 index for the third axis

**matrix_item = [row_index][column_index][last_index]**

CoGrammar

## Summary

- **A list** is a container created by surrounding a dynamically typed array sequence of variables or literals with brackets [ ] or list().
- **Lists** operations include:
  - pop()
  - append()
  - extend()
  - The rest can be Data Structures
- Dimensionality can be 1D, 2D or 3D

CoGrammar

# Practical

**Problem Statement:** High Scores Tracker

You're tasked with creating a high scores tracker for a video game. The tracker should store the top 5 scores achieved by players. Players can submit their scores, and the tracker should update accordingly, ensuring only the highest scores are kept.
Write a Python program that implements this high scores tracker using lists. Your program should include the following functionalities:

- Initialize an empty list to store the top 5 scores.
- Implement a function **submit_score(score)** that takes a player's score as input and updates the list of top scores if the score is among the top 5.
- Implement a function **display_scores()** that displays the current top 5 scores in descending order.

Ensure your program handles edge cases such as ties in scores and maintains the order of submission for players with the same score.

CoGrammar

## Homework

```
# Initialize high scores tracker
initialize_high_scores()

# Submit scores
submit_score(800)
submit_score(600)
submit_score(1000)
submit_score(700)
submit_score(900)

# Display current top 5 scores
display_scores()
```

## Practical

```
# Initialize high scores tracker
initialize_high_scores()

# Submit scores
submit_score(800)
submit_score(600)
submit_score(1000)
submit_score(700)
submit_score(900)

# Display current top 5 scores
display_scores()
```

**Output:**

```
Top 5 Scores:
1. 1000
2. 900
3. 800
4. 700
5. 600
```

CoGrammar

## Questions and Answers

CoGrammar

## Let's take a short break

CoGrammar

# Thank you for attending

SKILLS FOR LIFE
SKILLS BOOTCAMPS

Department for Education

CoGrammar