CoGrammar

Sequences: Revision

SKILLS FOR LIFE
SKILLS BOOTCAMPS

Department for Education

Software Engineering Lecture Housekeeping

- The use of disrespectful language is prohibited in the questions, this is a supportive, learning environment for all - please engage accordingly. **(FBV: Mutual Respect.)**
- 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 Open Classes. You can submit these questions here: **Open Class Questions**

Software Engineering Lecture 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**

Progression Criteria

✓ **Criterion 1: Initial Requirements**

- Complete 15 hours of Guided Learning Hours and the first four tasks within two weeks.

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

- Software Engineering: Finish 14 tasks by week 8.
- Data Science: Finish 13 tasks by week 8.

✓ **Criterion 3: Post-Course Progress**

- Complete all mandatory tasks by 24th March 2024.
- Record an Invitation to Interview within 4 weeks of course completion, or by 30th March 2024.
- Achieve 112 GLH by 24th March 2024.

✓ **Criterion 4: Employability**

- Record a Final Job Outcome within 12 weeks of graduation, or by 23rd September 2024.

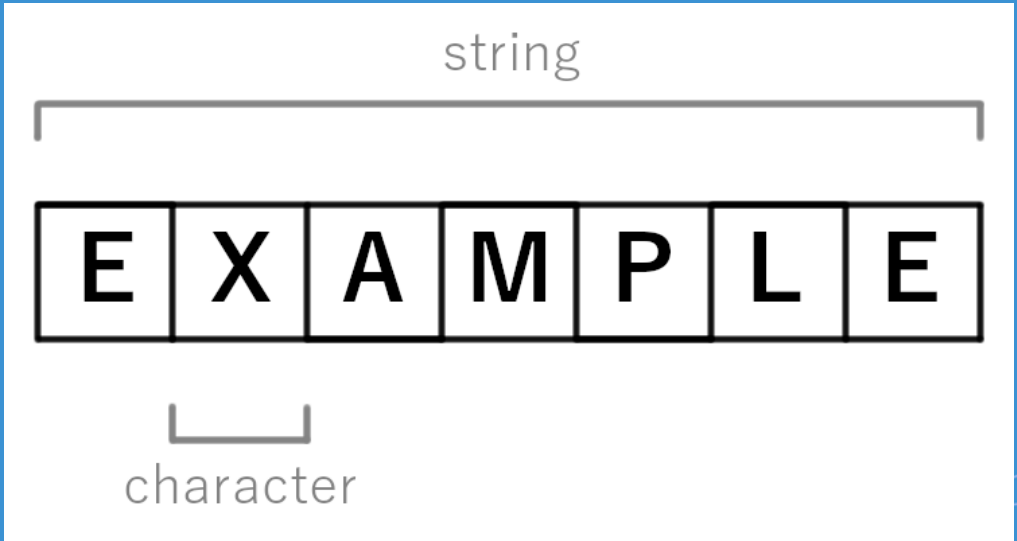
Lecture Objectives

1. String Fundamentals
2. Lists Fundamentals
3. List Comprehension
4. 2D Lists
5. Fundamentals of Dictionaries

What are Sequences?

- ★ A sequence is a collection of elements that are ordered and indexed, allowing for efficient access to individual elements by their position.
- ★ Sequences maintain the order in which elements are added and can contain duplicate values.

Strings



String Creation and Initialization

```
message = "This is a string"
print(message)
```

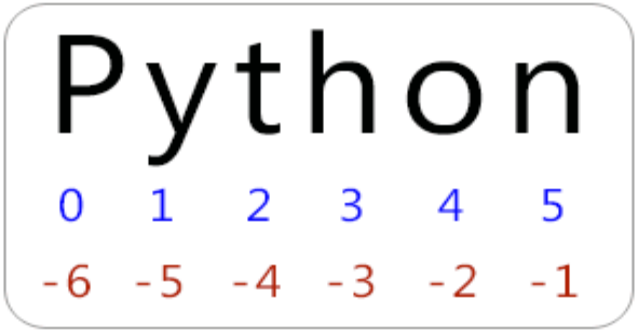
Python String Methods

Input	Method	Output
"Hello World"	.endswith("by")	False
"Hello World"	.startswith('Hello')	True
"hello world"	.captalize()	"Hello World"
"13/11/2021"	.split("/")	["13", "11", "2021"]
" Hello "	.strip()	"Hello"
"Hello A"	.replace("A","B")	"Hello B"
"hello world"	.count("o")	2
"Hello World"	.find("o")	4
"123456"	.isnumeric()	True
"HELLO"	.lower()	"hello"
"hello"	.upper()	"HELLO"

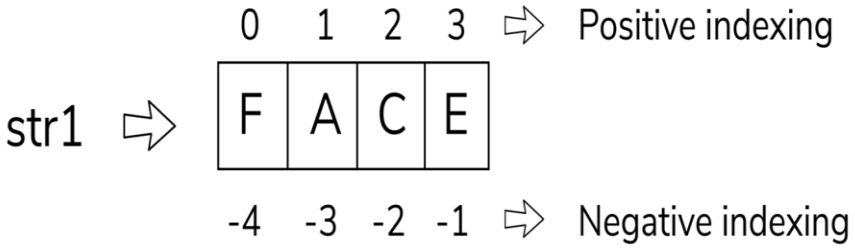
Strings Are Immutable

- When an object is immutable it means the object cannot be changed.
- When we apply methods to a string that appear to make changes, they are actually creating and returning new string objects.
- This means we have to store the changes we make in a variable to be reused.

String Indexing



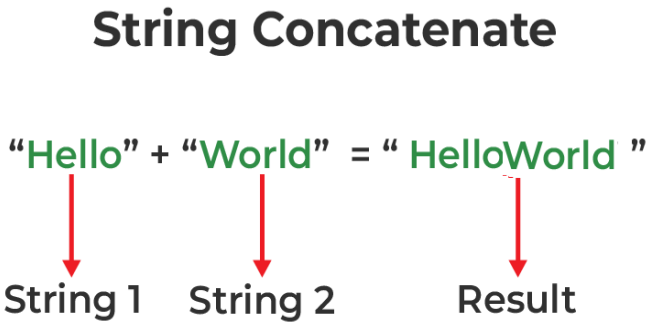
String Slicing



str1[1:3] = AC

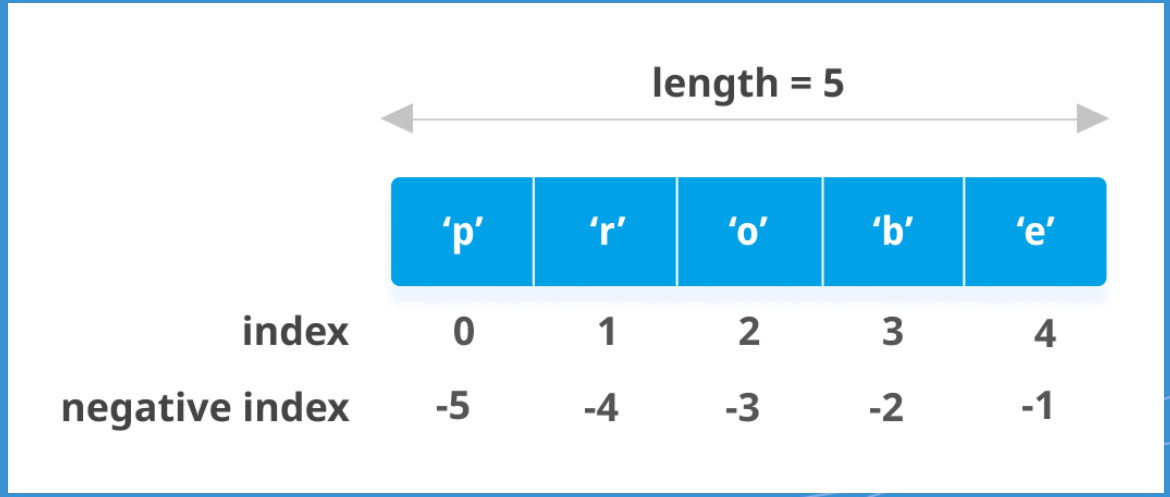
str1[-3:-1] = AC

String Concatenation & Formatting



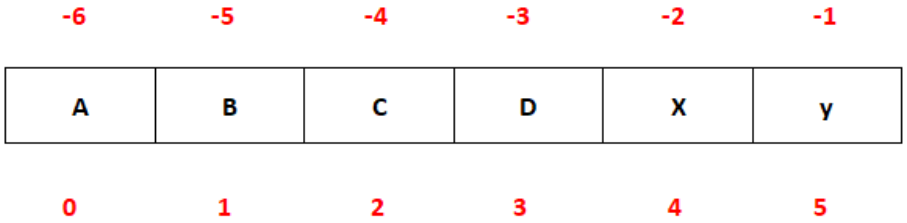
f-strings and format() function

Lists



Lists

- ★ A list is a data type that allows us to store multiple values of any type together and a list can contain duplications.
- ★ We can access individual values using indexing and multiple values using slicing.
- ★ We can iterate over lists using a for loop.



Lists ...

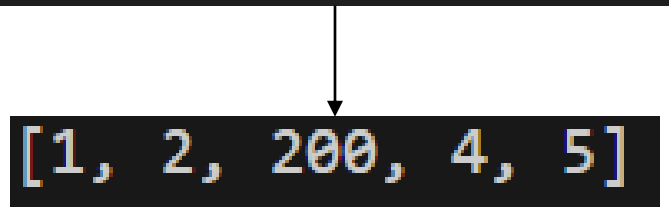
- ★ Lists are mutable.
- ★ This means the values inside a list can be changed and unlike a string won't return a new list when changes have been made.
- ★ We can apply methods to our lists without having to restore them inside our variables.

Lists ...

- ★ To create a list we can surround comma separated values with square brackets. []
- ★ E.g. my_list = [value1, value2, value3]
- ★ Adding Elements: append(), insert()
- ★ Removing Elements: remove(), pop() and ‘del’
- ★ Manipulating elements: sorting, reversing and slicing

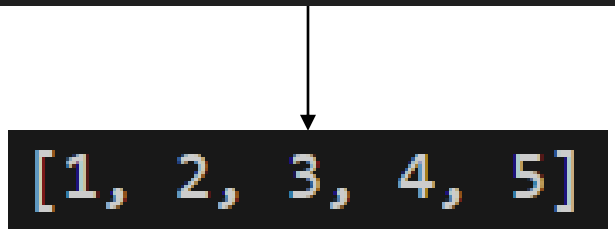
List Example

```
num_list = [1,2,3,4,5]
new_num_list = num_list

new_num_list[2] = 200
print(num_list)
```

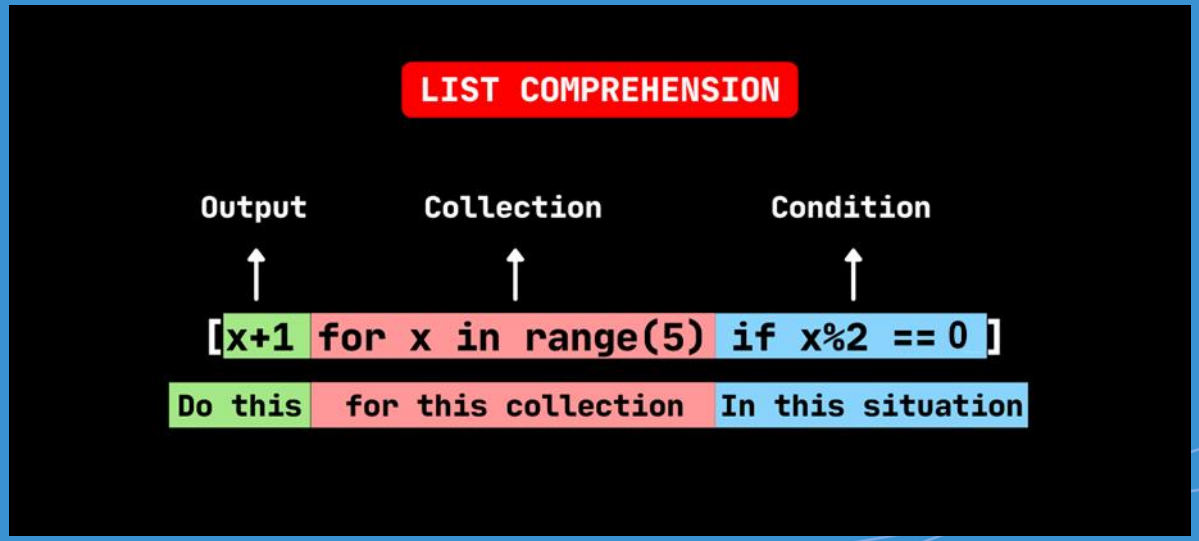


List Example

```
num_list = [1,2,3,4,5]
new_num_list = num_list.copy()

new_num_list[2] = 200
print(num_list)
```



List Comprehension



List Comprehension

★ List comprehension is a condensed method for creating lists in Python. In comparison to conventional for-loops, it offers a more condensed syntax for creating lists.

```
List Comprehension:
# Basic Structure
new_list = [expression for item in iterable]

# Squaring numbers from 0 to 9
squares = [x**2 for x in range(10)]
# Result: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

List Comprehension ...

```
List Comprehension:
# Basic Structure
new_list = [expression for item in iterable]

# Squaring numbers from 0 to 9
squares = [x**2 for x in range(10)]
# Result: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

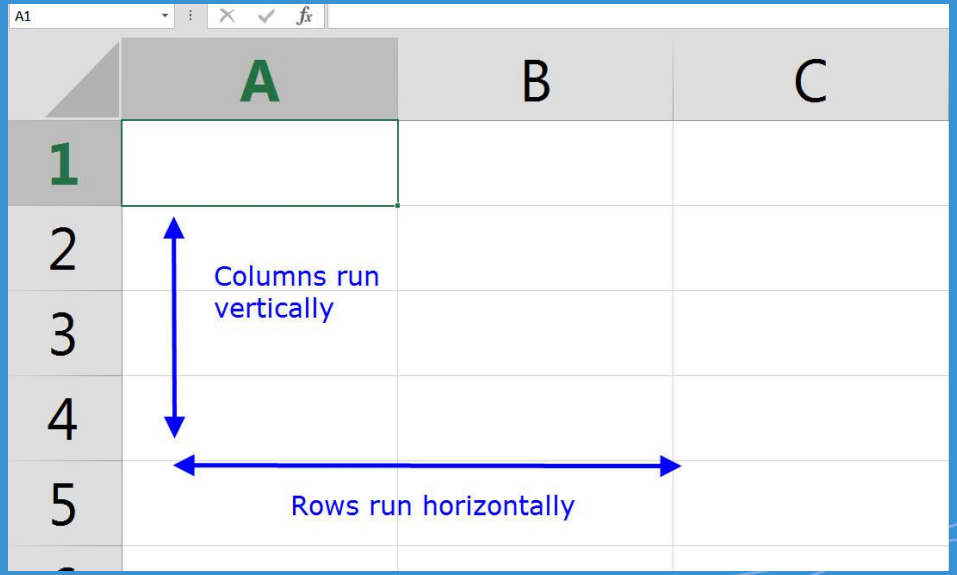
- ★ **Expression:** The expression to be evaluated and included in the new list.
- ★ **Item:** The variable representing an element in the iterable (e.g., a range, list, string).
- ★ **Iterable:** The source of data to iterate over.

Benefits & Precautions

★ **Benefits:**
- ***Conciseness:*** Achieve the same result with less code.
- ***Readability:*** Express your intent more clearly and compactly.
- ***Efficiency:*** List comprehensions are often faster than equivalent for-loops.

★ **Considerations:**
- ***Avoid Complexity:*** While list comprehensions are powerful, avoid making them overly complex for the sake of readability.
- ***Conditional Expressions:*** Can be used for conditional inclusion. ie. *value_if_true* ***if*** *condition* ***else*** *value_if_false*

2D Lists



2D List

- ★ A list within a list.
- ★ Outer List (1 Dimension) + Inner List (1 Dimension) = 2D

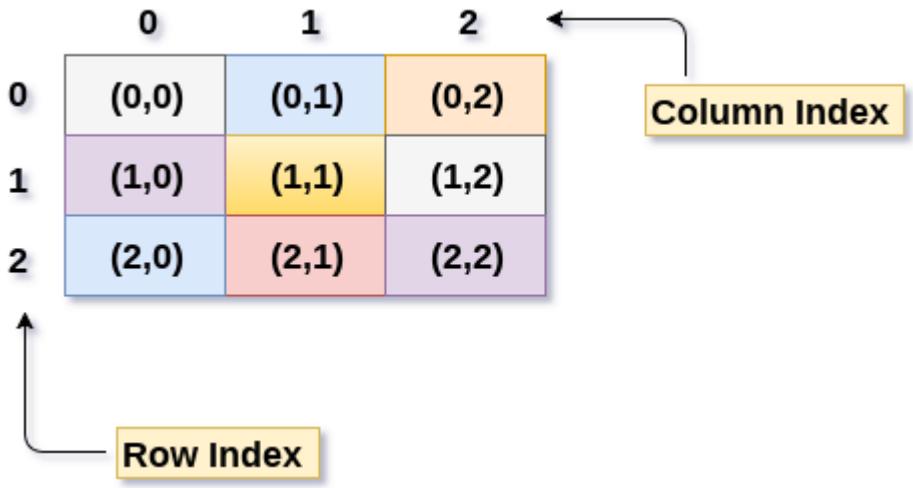
Rows and Columns

- ★ Elements are essentially accessed using rows and column indices.

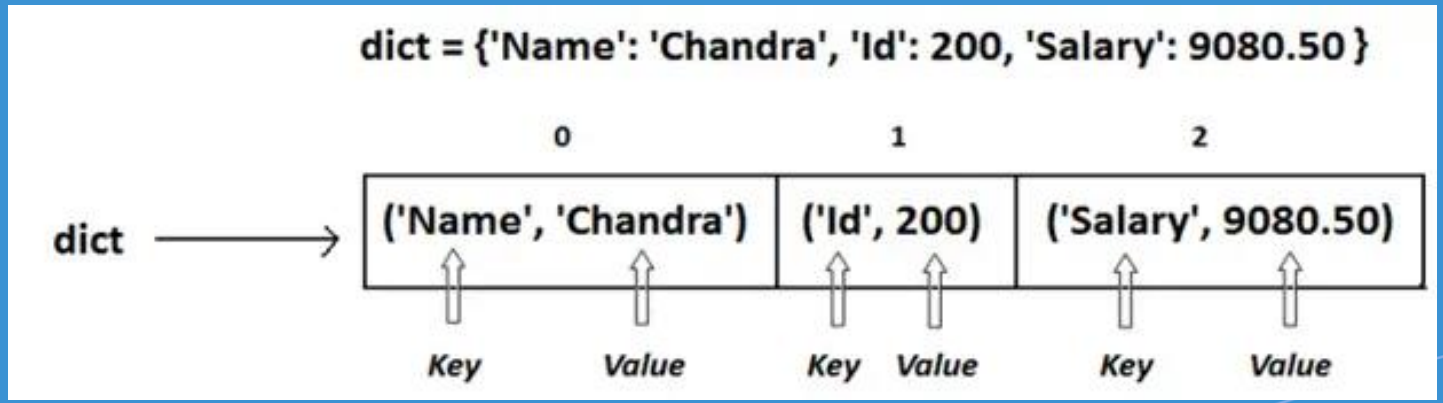
```
two_d = [
        [1, 2, 3],
        [4, 5, 6],
        [7, 8, 9]
        ]
```

Traversing the 2D

- ★ Nested Loops (iterate through rows and columns)
- ★ List comprehension



Dictionaries



Dictionaries

- ★ In Python, dictionaries function like the dictionaries we commonly use in English class, such as those from Oxford.
- ★ Python dictionaries are similar to a list, however each item has two parts, a key and a value.
- ★ To draw a parallel, consider an English dictionary where the key represents a word, and the associated value is its definition.

Dictionary Example

```
# Dictionary Example

my_dictionary = {
    "name": "Terry",
    "age": 24,
    "is_funny": False
}
```

- Dictionaries are enclosed in curly brackets; key value pairs are separated by colon and each pair is separated by a comma.
- On the left is the key, on the right is the value.

Dictionary Function

- The dict() function in Python is a versatile way to create dictionaries or cast to dictionary data type.
- Create dictionaries through assigning values to keys by passing in keys and values separated by an = sign.

```
# Creating a dictionary with direct key-value pairs
my_dict = dict(name="Kitty", age=25, city="Belarus")
print(my_dict)
# Output: {'name': 'Kitty', 'age': 25, 'city': 'Belarus'}
```

Dictionary Update

- To append or add elements to a dictionary in Python, you can use the update() method or simply use the square bracket notation.

```
my_dict = dict(name="Kitty", age=25, city="Belarus")
# Adding or updating a key-value pair
my_dict.update({'breed': 'Shorthair'})
print(my_dict)
# Output: {'name': 'Kitty', 'age': 25, 'city': 'Belarus', 'breed': 'Shorthair'}
```

Dictionary Access

- To access a value in a dictionary, we simply call the key and Python will return the value paired with the provided key.
- Similar to indexing, however we provide a key name instead of an index number.

```
my_dict = dict(name="Kitty", age=25, city="Belarus")
name = my_dict["name"]
# Output: 'Kitty'
```

Dictionary Operations

- **Key-Value Pairs**
  - items() - *for key, value in my_dict.items()*
- **Fetching**
  - get() - *name = my_dict.get('name')*
- **Updating**
  - update() - *update_dict = {'age': 29, 'salary': 60000}*
    - *my_dict.update(update_dict)*
- **Deleting**
  - pop() - *age = my_dict.pop('age', 'Not Specified')*

CoGrammar

Questions around Sequences

CoGrammar

Thank you for joining