CoGrammar

Lecture 5: Handling Strings, Lists and Dictionaries

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**

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**

Lecture Objectives

- **Share a brief recap on Strings and String Methods, the cover some more advanced formatting**

- **Learn about 2 new collection data types: Lists and Dictionaries**

Remembering Strings

```python
# We remember strings and how we can manipulate them?
word = "Hello There!"


idx = word[0 : 8 : 2] # [start : end : step]
print(idx)
```

Remembering Strings

```python
# There's also string methods
word = "do you enjoy Python yet?"


new_upper = word.upper()
new_lower = word.lower()
new_capital = word.capitalize()
```

Remembering Strings

```
# More methods include :
new_strip = word.strip("?")
new_split = word.split(" ")
new_replace = word.replace(" ", "@")
```

Formatting Strings

Instead of simply concatenating strings together, we have the ability to create more complex, formatted strings. With the power of this method we are able to create formatted strings by placing variables into placeholders.

Formatting Strings

```python
# Formatting Strings goes as follows :
name = "John"
surname = "Python"


print("My name is {} {} and I am a thorough enjoyer of {} \
".format(name, surname, surname))

# An alternate way of writing
print("My name is {0} {1} and I am a thorough enjoyer of {1} \
".format(name, surname))
```

Format Examples

```python
# We could even display decimal values!
value = 68.85673423


output = "Your total on your order will be : £{:.2f}"
print(output.format(value))
# Do keep in mind that the decimal function will round up if possible
```

F-Strings

There is another means to format strings that was introduced, which is called Literal String Interpolation, or better known as f-strings.
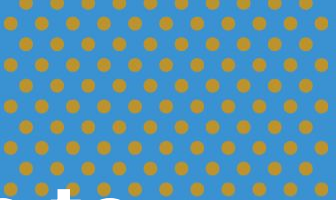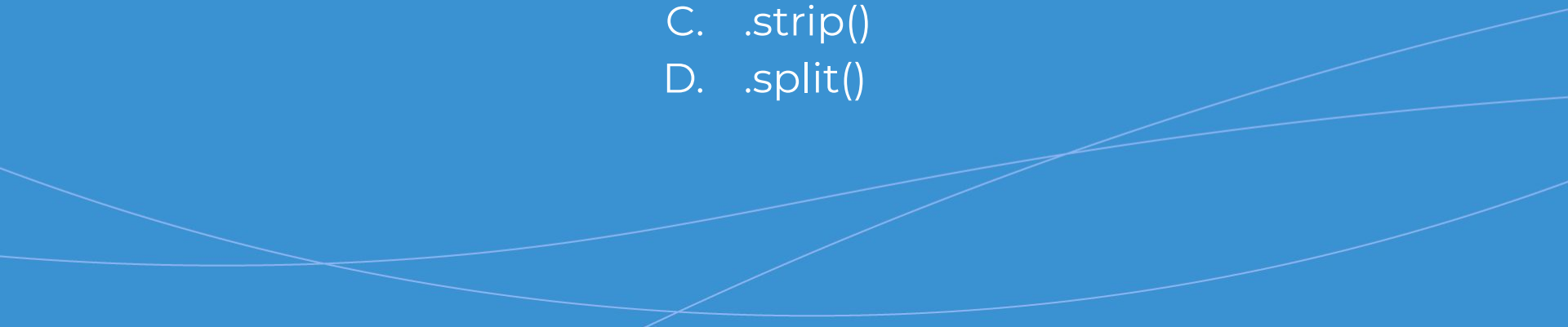
F-String Example

```python
# F-Strings

name = "John"
surname = "Python"

print(f"My name is {name} {surname} and I am a thorough Python enjoyer.")
```

What method could we use to replace 1 character in a string with a new character of our choosing?

A. .replace()
B. .swap()
C. .strip()
D. .split()

Which letter must be added as a string prefix in order to allow for interpolation?

A. s
B. i
C. f
D. d

Which of the following slices will access every second character of a string?

A.   [2:2:1]
B.   [0:2]
C.   [2]
D.   [::2]

Lists

★ **Lists are used when we need to store a lot of data, or the order in which the data is stored is important to us.**

★ **Lists are capable of holding many items in one place as well as keeping the data in order.**

★ **Python will also provide each piece of data an index that will represent its position in the list.**

Lists

★ **A List is a specialised format of storing and organising data.**

★ **A List is basically a group of items / data.**

★ **Lists are known as sequence data types because they behave like an ordered collection of items.**

List Example & Syntax

```python
my_list=[]
#You can even create empty lists

names=["Billy", "Jimmy", "Sally", "Rachel"]
various=["Tom", 32, True, 21.3]

# Lists are excellent for storing multiple pieces of data
# from string to integers, floats, and even booleans

# Lists remove the need to create multiple different
# variables taking up memory in your program, making the
# process of storing data easier and more efficient
```

Indexing Lists

★ Similar to strings, we are able to index and slice lists.

★ However, instead of indexing by character, we index lists by the entire value in that specific position.

Indexing Example

```
names = ["Billy", "Sally", "Cammy"]
print(names[0])

# Result >> "Billy"

print(names[-1])

# Result >> "Cammy"
```

Finding the Length of a List

★ **Similar to what we can do with strings, we can use the len() function to find the length of a list.**

★ **Example :**

```
my_list = ["The", "Joy", "of", "Learning"]

print(len(my_list))

# Result >> 4
```

Accessing all values in a List

★ **Sometimes we need to access all values / items in a list at the same time; to achieve this we can simply iterate through the list with a for loop.**

★ **This would be especially useful when we cannot exactly see all the data within a list and we must evaluate the data / make adjustments / add to the list / remove from the list.**

Looping through Lists Example

```
names = ["Jimmy", "Billy", "Terry", "Kerry", "Joe"]

# Remember that the temp variable in a for loop
#    can be named anything and that it only
#        exists within the loop.
for i in names:
    print(names)

# Result >>  Jimmy
#            Billy
#            Terry
#            Kerry
#            Joe
```

In Operator and Lists

```python
names = ["Jimmy", "Billy", "Terry", "Kerry", "Joe"]

name_one = "Lucy"
name_two = "Terry"

if name_one in names:
    print('''
    This if statement will NOT execute, because Lucy
    does not exist in the list
    ''')

if name_two in names:
    print('''
    This if statement will execute, because Terry
    does in fact exist within the list
    ''')
```

Appending data to Lists

★ **We can add new items to a list by using the .append() method, keep in mind that append will only add to the end of a list, and nowhere else.**

★ **Example :**

```
names = ["Jimmy", "Billy", "Terry", "Kerry", "Joe"]

names.append("Sally")
# The list is now updated with the new item

print(names)

# Result >> ['Jimmy', 'Billy', 'Terry', 'Kerry', 'Joe', 'Sally']
```

Extending Lists

★ We can extend lists with multiple values which will be attached at the end.

★ It is similar to append, but is capable of adding multiple values.

```
numbers = [1,2,3,4]

numbers.extend([5,6,7,8])

print(numbers)

# Result >> [1,2,3,4,5,6,7,8]
```

Inserting into List

★ **We can insert values at a specific position in the list using indexing.**

★ **Takes two arguments, first is the index, followed by the element to add.**

```python
numbers = [1, 2, 3, 4, 5]

numbers.insert(2, 'Hi')

print(numbers)

# Result >> [1, 2, 'Hi', 3, 4, 5]
```

Popping from a List

★ **The pop method will remove an element at an index, then return it.**

★ **Return meaning that the popped element can be stored and used in a variable.**
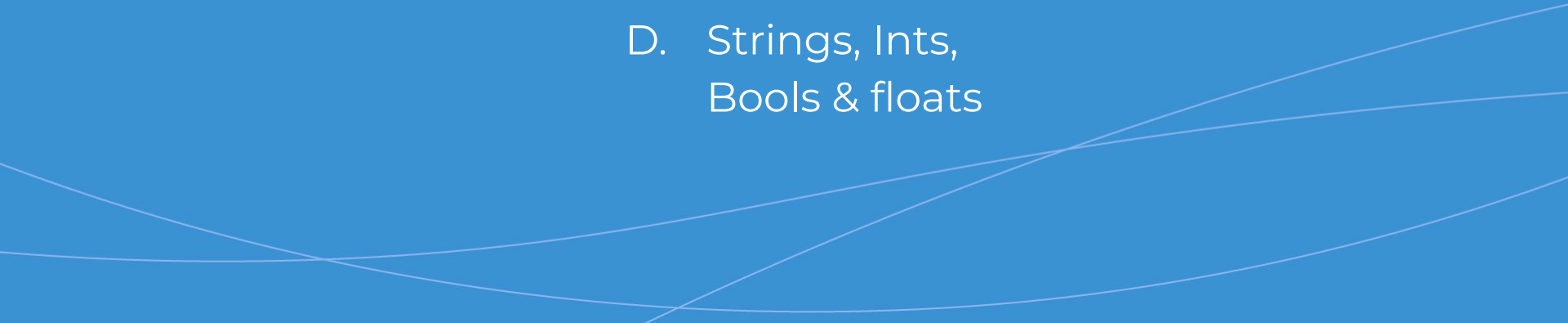
```
numbers = [1, 2, 3, 4, 5]

popped_number = numbers.pop()
# If no index is specified, then pop will remove
#    the last element in the list

print(popped_number)

# Result >> 5
```
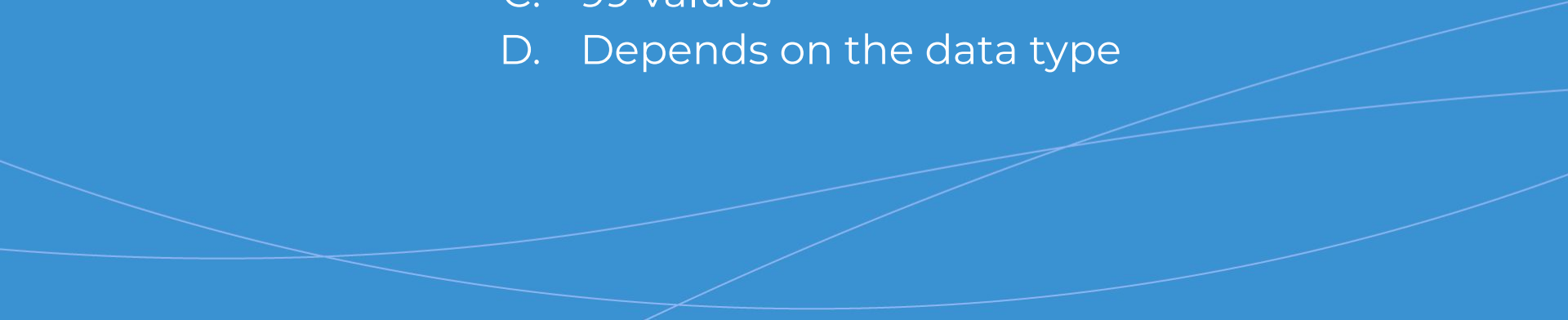
Which data types can we store in lists?

A. Other lists
B. Options A, C & D
C. Dictionaries
D. Strings, Ints, Bools & floats

What is the length limit of a default list?

A. There is no limit
B. 100 values
C. 99 values
D. Depends on the data type

Which of the will always access only the last value in a list?

A. [last]

B. [0:-1]

C. [-1]

D. [0]

Let's Breathe

Let's take a small break before moving on to the next topic.

CoGrammar

Dictionaries

★ **Python dictionaries are similar to a list, however each item has two parts, a key and a value.**

★ **For example, an English dictionary has the word (key) and then it's definition (value).**

Dictionaries

★ **Dictionaries are enclosed in curly brackets; key value pairs are separated by a colon and each pair is separated by a comma.**

★ **On the left is the key, on the right is the value.**

```
my_dictionary = {

    "name" : "Terry",
    "age" : 23,
    "is_funny" : False

}
```

Dict Function

★ **Used to create dictionaries.**

★ **Assign values to keys by passing in keys and values separated by and = sign.**

```
new_dictionary = dict(name="kitty", age=0.5, kitten=True)

print(new_dictionary)

# Result >> {'name': 'kitty', 'age': 0.5, 'kitten': True}
```

Accessing values in dictionaries

★ To access a value in a dictionary, we simply call the key and Python will return the value paired with said key.

★ Similar to indexing, however we provide a key name instead of an index number.

Example

```python
new_dictionary = dict(name="kitty", age=0.5, kitten=True)

print(new_dictionary["name"])

# Result >> kitty

print(new_dictionary["age"])

# Result >> 0.5
```

Accessing all values

★ **We are able to use the .values() method on a dictionary in a for loop to access every value in a dictionary.**

```python
new_dictionary = dict(name="kitty", age=0.5, kitten=True)

for value in new_dictionary.values():
    print(value)


# Result >> kitty
#           0.5
#           True
```

Accessing all keys

★ **We are able to use the .keys() method on a dictionary in a for loop to access every key in a dictionary.**

```python
new_dictionary = dict(name="kitty", age=0.5, kitten=True)

for keys in new_dictionary.keys():
    print(keys)


# Result >> name
#           age
#           kitten
```

Accessing both keys & values

★ **We are able to use the .items() method on a dictionary in a for loop to access both keys and values in a dictionary.**

```python
new_dictionary = dict(name="kitty", age=0.5, kitten=True)

for keys, values in new_dictionary.items():
    print(keys, " : ", values)

# Result >> name   :   kitty
#           age    :   0.5
#           kitten :   True
```

Popping from a dictionary

★ **Similar to the list, we can use .pop() to remove a pair out of a dictionary and return the value of the pair in a variable.**
★ **Please note that using .pop() must have the key as an argument for pop to work.**

```python
new_dictionary = dict(name="kitty", age=0.5, kitten=True)

value = new_dictionary.pop("name")

print(value)

# Result >> kitty
```

What data types can be used as keys for dictionaries?

A. Lists
B. Dictionaries
C. Strings, Ints, Bools & floats
D. Options A, B & C

Which of the below is a limitation of dictionaries?

A. They can store an infinite number of values as long as each value has a unique key

B. Their index starts on 1 instead of 0

C. We cannot iterate over them

D. They can only store 100 values

CoGrammar

Q & A SECTION

Please use this time to ask any questions relating to the topic, should you have any.

CoGrammar

Thank you for joining!