CoGrammar

Strings

SKILLS FOR LIFE
SKILLS BOOTCAMPS

Department for Education

Foundational Sessions 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: SE Open Class Questions **or** DS Open Class Questions

Foundational Sessions 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

Reminders!

Guided Learning Hours

By now, ideally you should have 7 GLHs per week accrued. Remember to attend any and all sessions for support, and to ensure you reach 112 GLHs by the close of your Skills Bootcamp.

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.

What is ASCII used for in computing?

A. Encrypting data.

B. Mapping characters to integers.

C. Storing multimedia files.

D. Rendering graphics.

Why is Unicode important in modern computing?

A. It supports only Latin characters.

B. It allows for more efficient data storage.

C. It provides a universal character set that includes a wide range of symbols and languages.

D. It speeds up the processing of text data.

Which Python method is considered more efficient for concatenating strings?

A. The '+' operator.

B. The append() method.

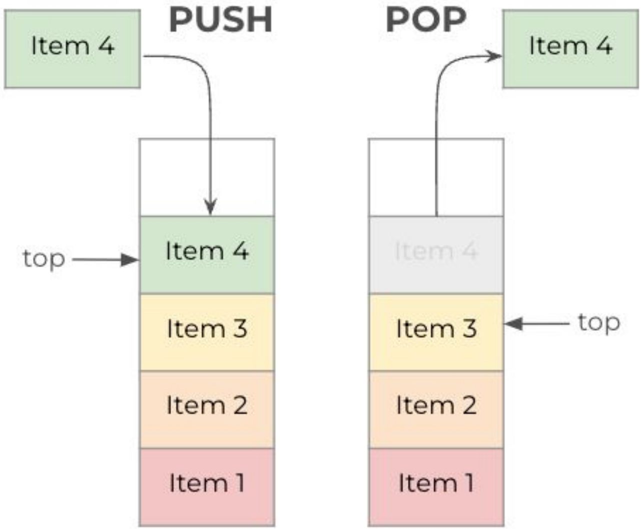
C. The join() method.

D. The concat() function.

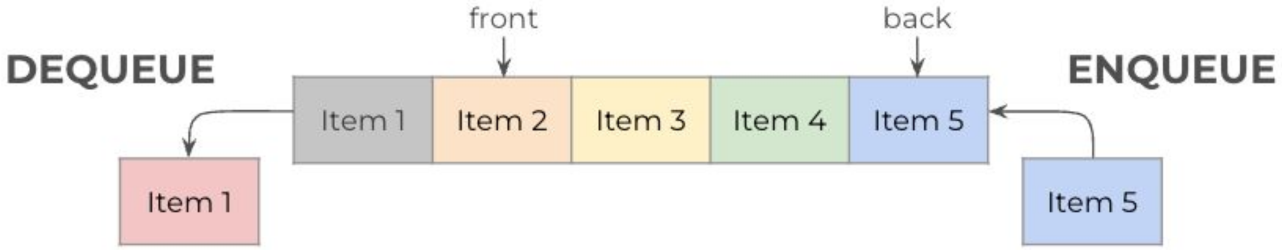
Recap of Stacks and Queues

- **Stacks and Queues:** Types of linear **data structures** that **allow for storage and retrieval of data** based on a specific method of ordering.

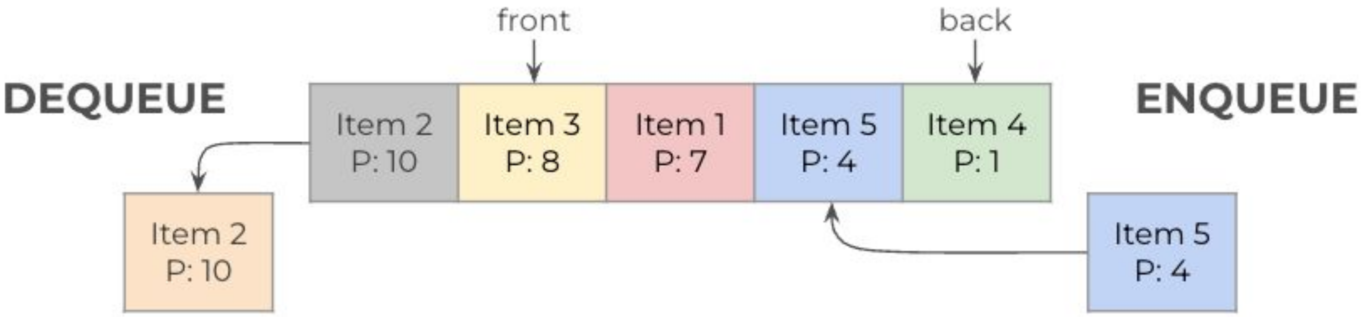
- **Stacks**
  - Uses LIFO ordering (think of stacking waffles)
  - Utilises push() and pop() operations



- **Queues**
  - Uses **FIFO** ordering (think of a shop queue)
  - Utilises **enqueue()** and **dequeue()** operations



- **Priority Queues**
  - Arranged according to the assigned priority of elements
  - Utilises enqueue() and dequeue() based on priority of elements



Strings Topics

1. ASCII
2. Unicode
3. String Concatenation in Python
4. String Join Method in Python
5. Performance Comparison of “+” and join() methods
6. String Builder Pattern

Multilingual Social Media Platform

Imagine you are developing a global social media platform that needs to support multiple languages and efficiently process large volumes of user-generated content daily. How would you handle diverse character encoding and ensure efficient string manipulation to manage this data effectively?

SPOILER!

By mastering **character encoding standards like Unicode** and employing **efficient string manipulation techniques** in Python, you can build a robust platform capable of handling text data from any language and at any scale!

Relevance of Character Encoding

- Essential in software development for representing text in computers and across diverse communication technologies.

- **ASCII** (American Standard Code for Information Interchange) and **Unicode** are two primary systems of character encoding.

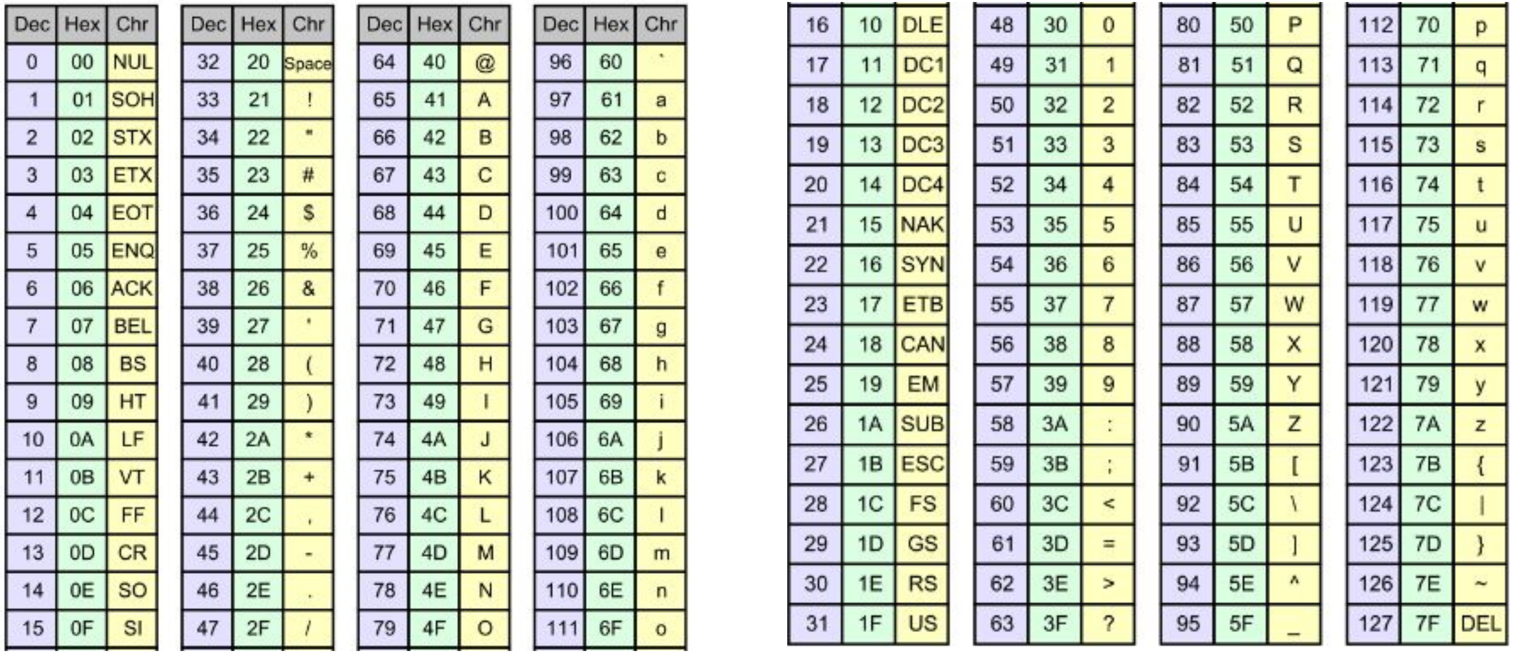
ASCII

ASCII is a character encoding standard used for electronic communication, representing text in computers and other devices with limited set of 128 code points

- ASCII assigns numeric values to letters, numerals, and symbols, but only includes 128 code points.

- This limitation renders ASCII **inadequate for languages outside the basic English alphabet**.

ASCII Table



Dec	Hex	Chr	Dec	Hex	Chr	Dec	Hex	Chr	Dec	Hex	Chr
0	00	NUL	32	20	Space	64	40	@	96	60	`
1	01	SOH	33	21	!	65	41	A	97	61	a
2	02	STX	34	22	"	66	42	B	98	62	b
3	03	ETX	35	23	#	67	43	C	99	63	c
4	04	EOT	36	24	$	68	44	D	100	64	d
5	05	ENQ	37	25	%	69	45	E	101	65	e
6	06	ACK	38	26	&	70	46	F	102	66	f
7	07	BEL	39	27	'	71	47	G	103	67	g
8	08	BS	40	28	(	72	48	H	104	68	h
9	09	HT	41	29	)	73	49	I	105	69	i
10	0A	LF	42	2A	*	74	4A	J	106	6A	j
11	0B	VT	43	2B	+	75	4B	K	107	6B	k
12	0C	FF	44	2C	,	76	4C	L	108	6C	l
13	0D	CR	45	2D	-	77	4D	M	109	6D	m
14	0E	SO	46	2E	.	78	4E	N	110	6E	n
15	0F	SI	47	2F	/	79	4F	O	111	6F	o

16	10	DLE	48	30	0	80	50	P	112	70	p
17	11	DC1	49	31	1	81	51	Q	113	71	q
18	12	DC2	50	32	2	82	52	R	114	72	r
19	13	DC3	51	33	3	83	53	S	115	73	s
20	14	DC4	52	34	4	84	54	T	116	74	t
21	15	NAK	53	35	5	85	55	U	117	75	u
22	16	SYN	54	36	6	86	56	V	118	76	v
23	17	ETB	55	37	7	87	57	W	119	77	w
24	18	CAN	56	38	8	88	58	X	120	78	x
25	19	EM	57	39	9	89	59	Y	121	79	y
26	1A	SUB	58	3A	:	90	5A	Z	122	7A	z
27	1B	ESC	59	3B	;	91	5B	[	123	7B	{
28	1C	FS	60	3C	<	92	5C	\	124	7C	|
29	1D	GS	61	3D	=	93	5D	]	125	7D	}
30	1E	RS	62	3E	>	94	5E	^	126	7E	~
31	1F	US	63	3F	?	95	5F	_	127	7F	DEL

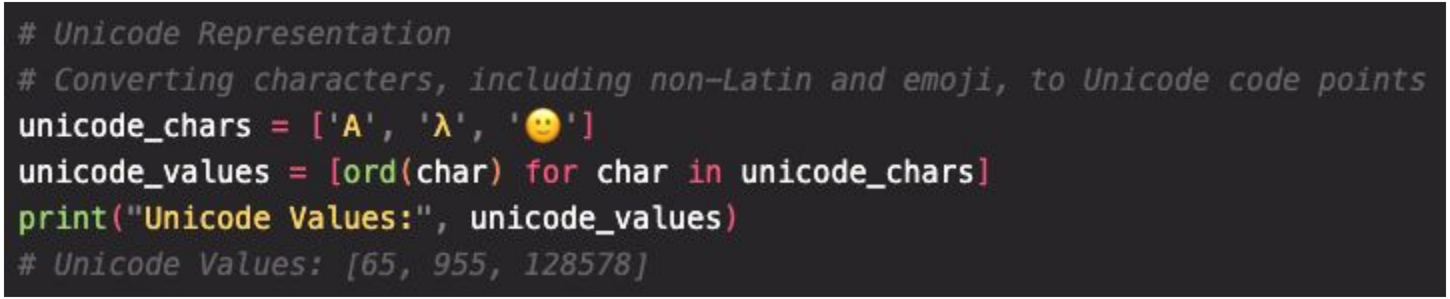
Unicode

Unicode is a universal encoding system that allows the representation of a comprehensive character set, including those required for global multilingual text processing

- Unicode was developed to overcome ASCII's limitations, **supporting a wider array of characters**.

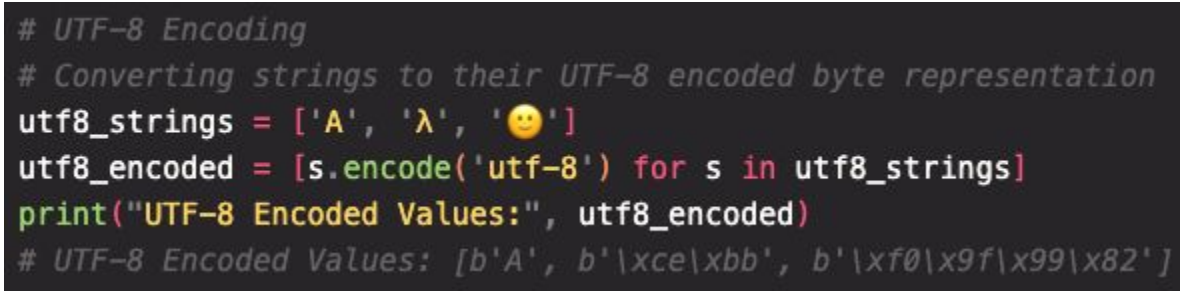
- It supports over **149 813 characters** as of version 15.1, including non-Latin scripts and emojis. Unicode unifies different encoding schemes to **facilitate global communication**.

- The unicode encoding for 🙂 is 128 578. Note that ord() in Python uses unicode instead of ASCII, given ASCII’s limitations.



```python
# Unicode Representation
# Converting characters, including non-Latin and emoji, to Unicode code points
unicode_chars = ['A', 'λ', '🙂']
unicode_values = [ord(char) for char in unicode_chars]
print("Unicode Values:", unicode_values)
# Unicode Values: [65, 955, 128578]
```

- UTF-8, a popular Unicode format, encodes characters using 1 to 4 bytes, providing efficient data representation.



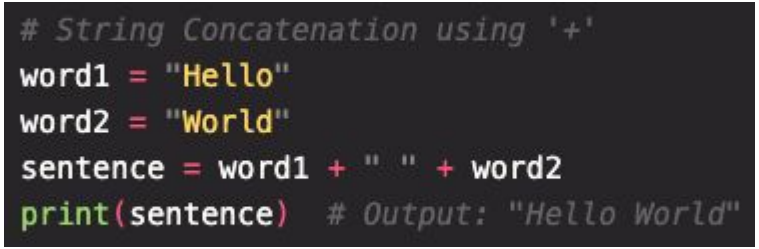
```
# UTF-8 Encoding
# Converting strings to their UTF-8 encoded byte representation
utf8_strings = ['A', 'λ', '🙂']
utf8_encoded = [s.encode('utf-8') for s in utf8_strings]
print("UTF-8 Encoded Values:", utf8_encoded)
# UTF-8 Encoded Values: [b'A', b'\xce\xbb', b'\xf0\x9f\x99\x82']
```

- When we say UTF-8 uses 1-4 bytes, it means that depending on the character, it might use just 1 byte (like for 'A'), 2 bytes (like for 'λ'), or even up to 4 bytes (like for '🙂'). This allows UTF-8 to represent any character in the Unicode standard while **still being efficient** for characters in the ASCII range.

String Concatenations in Python

- **Strings in Python are immutable**, which means once a string is created, it cannot be changed; therefore, concatenating with '+' creates new strings each time.

- This method is less efficient for concatenating many strings due to **increased memory and processing requirements**.

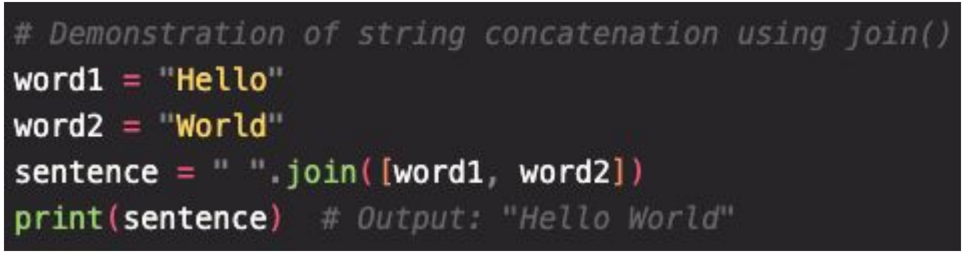


```python
# String Concatenation using '+'
word1 = "Hello"
word2 = "World"
sentence = word1 + " " + word2
print(sentence)  # Output: "Hello World"
```

String Join Method in Python

- The join() method in Python is a **more efficient way to concatenate strings**, especially for large numbers of strings.

- It creates a new string in a single pass, improving performance.



```python
# Demonstration of string concatenation using join()
word1 = "Hello"
word2 = "World"
sentence = " ".join([word1, word2])
print(sentence)  # Output: "Hello World"
```

Performance Comparison

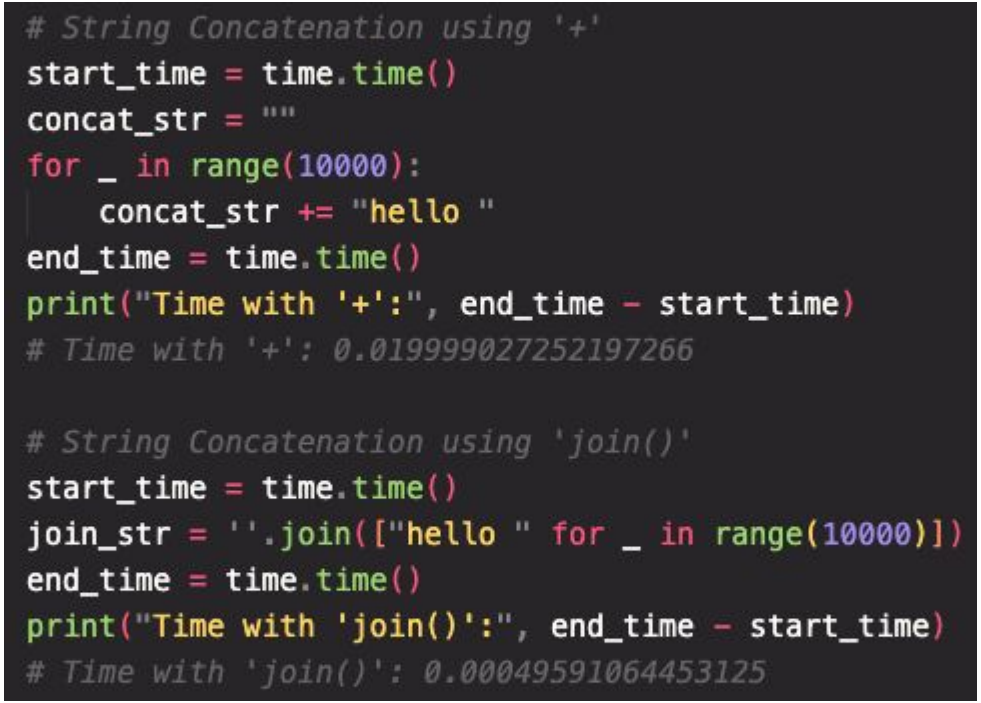
“+” concatenation

- **Performance:** O(n^2)
- This is because each time you concatenate two strings, a new string is created and the characters from both strings are copied into it.

join() concatenation

- **Performance:** O(n)
- Only creates one new string at the end, and it only needs to copy the characters once.

Performance Comparison



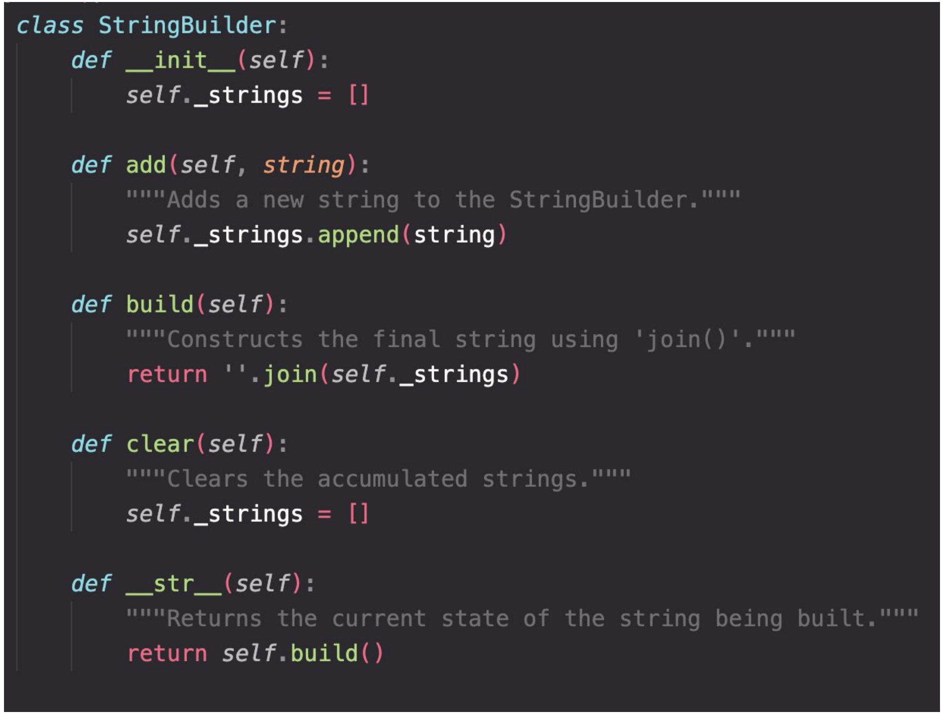
```
# String Concatenation using '+'
start_time = time.time()
concat_str = ""
for _ in range(10000):
    concat_str += "hello "
end_time = time.time()
print("Time with '+':", end_time - start_time)
# Time with '+': 0.019999027252197266

# String Concatenation using 'join()'
start_time = time.time()
join_str = ''.join(["hello " for _ in range(10000)])
end_time = time.time()
print("Time with 'join()':", end_time - start_time)
# Time with 'join()': 0.00049591064453125
```

The join() method is clearly more efficient since it completes the task much faster than standard Python concatenation.

String Builder Pattern in Python

- The string builder pattern is a **common design pattern for efficient string manipulation** and is implemented using the join() method.

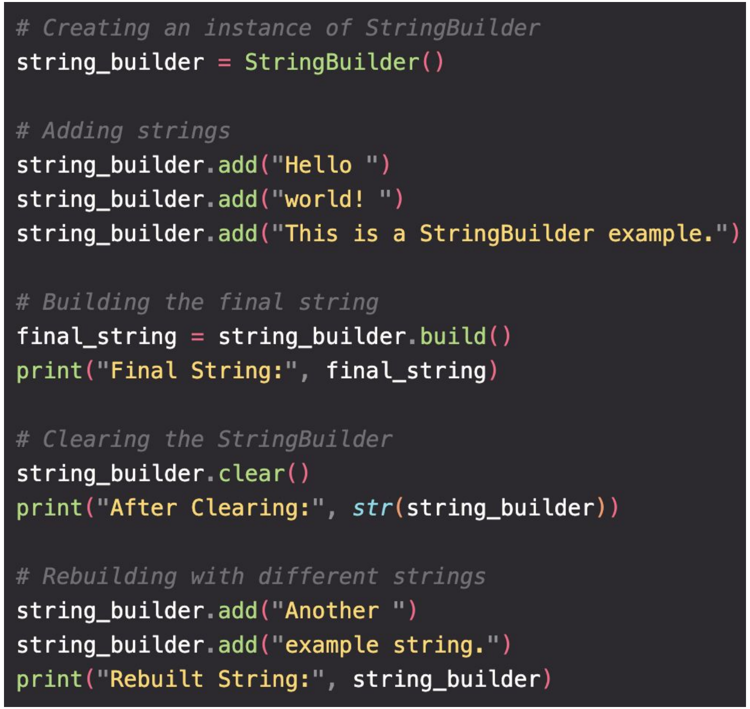
- A design pattern is a **general repeatable solution to a commonly occurring problem in software design**.



```python
class StringBuilder:
    def __init__(self):
        self._strings = []

    def add(self, string):
        """Adds a new string to the StringBuilder."""
        self._strings.append(string)

    def build(self):
        """Constructs the final string using 'join()'."""
        return ''.join(self._strings)

    def clear(self):
        """Clears the accumulated strings."""
        self._strings = []

    def __str__(self):
        """Returns the current state of the string being built."""
        return self.build()
```

The StringBuilder class is simple and only has a few important functions, namely **add()**, **build()**, and **clear()**.

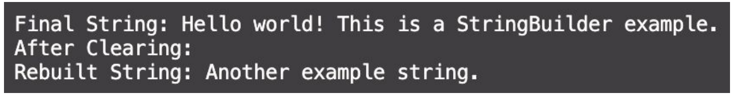
When it is instantiated, the initial string is empty, but then **adds strings as add() is called** and **efficiently builds the entire string with build()**



```python
# Creating an instance of StringBuilder
string_builder = StringBuilder()

# Adding strings
string_builder.add("Hello ")
string_builder.add("world! ")
string_builder.add("This is a StringBuilder example.")

# Building the final string
final_string = string_builder.build()
print("Final String:", final_string)

# Clearing the StringBuilder
string_builder.clear()
print("After Clearing:", str(string_builder))

# Rebuilding with different strings
string_builder.add("Another ")
string_builder.add("example string.")
print("Rebuilt String:", string_builder)
```

In this example we build two strings by incrementally adding to our initially empty string. As you can see, StringBuilder makes building large strings easy and efficient, which is very important the larger the input gets.

The output of this example is as follows:



```
Final String: Hello world! This is a StringBuilder example.
After Clearing:
Rebuilt String: Another example string.
```

Worked Example

You are tasked with generating a report that compiles data from multiple sources. The report should include textual data in various languages, **including some non-Latin characters and emojis,** to cater to a global audience. **The data is in the form of a list of strings**, each string representing a piece of the report. Your goal is to **efficiently construct this report while ensuring proper character encoding.**

CoGrammar

1. How would you ensure that the non-Latin characters and emojis in the report are correctly encoded?

2. What method would you use to concatenate the strings for the report to ensure maximum efficiency?

3. How would you implement the string builder pattern in Python to construct the report?

Worked Example

You are tasked with generating a report that compiles data from multiple sources. The report should include textual data in various languages, **including some non-Latin characters and emojis,** to cater to a global audience. **The data is in the form of a list of strings**, each string representing a piece of the report. Your goal is to **efficiently construct this report while ensuring proper character encoding.**

1. How would you ensure that the non-Latin characters and emojis in the report are correctly encoded?

- **Encoding the strings in UTF-8 ensures that all characters, including non-Latin and emojis, are correctly represented.**

2. What method would you use to concatenate the strings for the report to ensure maximum efficiency?

- **The join() method is used for concatenating strings as it is more efficient than using the '+' operator, especially for a large number of strings.**

3. How would you implement the string builder pattern in Python to construct the report?

- **Simply put, the string builder pattern is implemented using a function string_builder that utilises the join() method.**

Summary

Character Encoding Concepts

★ **ASCII:** A character encoding standard with 128 code points, primarily for English text.

★ **Unicode:** A comprehensive system supporting over 149,000 characters, accommodating global languages and symbols.

Differences Between ASCII and Unicode

★ ASCII's limitation is its inability to represent characters beyond basic English.

★ Unicode overcomes this by encoding a vast array of characters, including non-Latin scripts and emojis.

Summary

Python String Manipulation

★ Strings in Python are immutable; concatenating with '+' creates new strings each time.

★ The join() method is more efficient than '+' for concatenating multiple strings, as it creates a new string in a single pass.

String Builder Pattern in Python

★ A method set for efficient string construction without needing to create new strings for each concatenation.

★ Utilises the join() method for dynamic and memory-efficient string assembly.

Further Learning

- ASCII basics and history
- Unicode system and significance
- Comparing string concatenation methods
- StringBuilder pattern implementation
- Builder pattern usage and benefits

What is a primary advantage of using Unicode over ASCII?

A. Unicode supports a wider range of characters.

B. Unicode is faster to process.

C. Unicode uses less memory.

D. Unicode is older and more tested.

Which statement best describes efficient string handling and character encoding?

A. ASCII for non-Latin characters, ‘+’ for concatenations.

B. Unicode for all characters, ‘+’ for long strings.

C. Unicode for emotion icons, ‘join()’ for concatenation.

D. ASCII for icons, string builder for long texts.

Questions and Answers

Questions around Sets, Functions and Variables