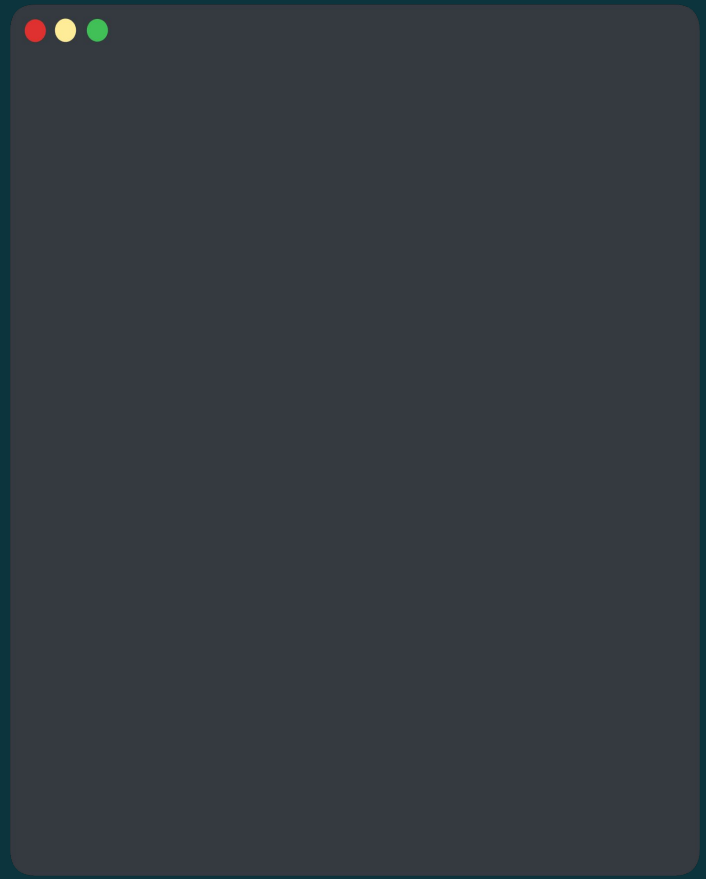
07 - Strings

Strings Recap



Strings Recap

- Strings is a data structure that contains a sequence of characters enclosed in quotes.
- Strings in Python can be created using **single quotes, double quotes**, or **triple quotes**.
- Strings are used to represent and manipulate textual data in your program.

```
single_quote = 'This is a string'

double_quotes = "This is also a string"

triple_quotes = '''This is a string that spans multiple lines. Often useful for writing docstrings.'''
```

Using Strings

- In Python, strings are represented as a `**str**` data type.
- Strings are an immutable data-structure,
  - i.e. you cannot modify the value of an initialised string.

Using Strings

- In Python, strings are represented as a `**str**` data type.
- Strings are an immutable data-structure,
    - i.e. you cannot modify the value of an initialised string.

In this chapter, you'll find a cheat sheet that covers some of the common string operations.

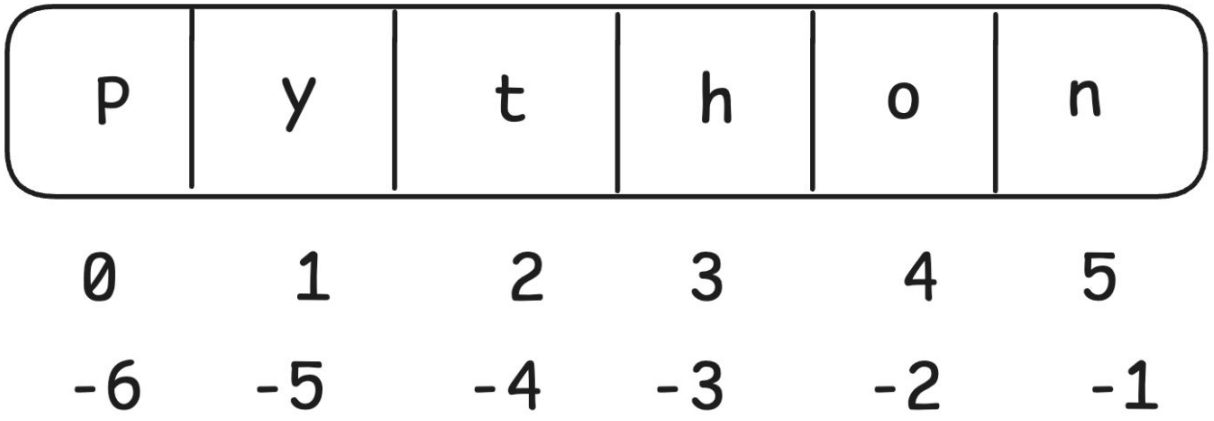
String Indexing

String Indexing

```
name = "Python"
```

```
name[0] = "P"
name[1] = "y"

name[-1] = "n"
name[-2] = "o"
```



String Indexing

Indexing allows you to access individual characters in a string using zero-based indexing.

Positive indices start from 0, while negative indices count from the end (-1 for the last character)

String Indexing

Indexing allows you to access individual characters in a string using zero-based indexing.

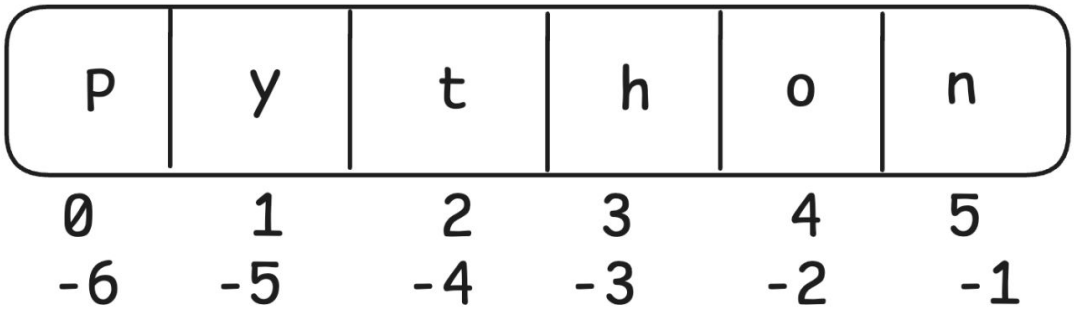
Positive indices start from 0, while negative indices count from the end (-1 for the last character)

- You can think of Strings as an array or list of characters. You can access a specific character by its index (similar to how you'd do it in a list)

String Slicing

String Slicing

```
name = "Python"
```




```
name[0:2] = "Py"  - start from index 0 to index 2 (exclusive)
name[1:4] = "yth" - start from index 1 to index 4 (exclusive)

name[-6:-4] = "Py"  - start from index -6 to index -4 (exclusive)
name[-5:-2] = "yth" - start from index -5 to index -3 (exclusive)
```

String Slicing

- Slicing allows you to extract a portion of a string using a start and end index.
- Extract substring using slicing syntax **text[start:stop:step]**
  - `**start**` is inclusive
  - `**stop**` is exclusive
  - `**step**` is the stride
- Slicing is a way to acquire a subset from the given list by slicing it respectively from start to end.
- You can slice lists and strings by defining their start and end.

String Slicing

Remember the syntax:

- text[start:stop:step]
  - `**start**` is inclusive
  - `**stop**` is exclusive
  - `**step**` is the stride

```
text = "Python Programming"

print(text[7:18])    # Output: Programming

print(text[:6])      # Output: Python

print(text[::2])     # Output: Pto rgamn
```

String formatting

String formatting

It is the process of inserting a custom string in predefined text.

String formatting

It is the process of inserting a custom string in predefined text.

Python provides different methods to format and interpolate variables into strings.

String formatting

It is the process of inserting a custom string in predefined text.

Python provides different methods to format and interpolate variables into strings.

The most common techniques are:

1. `str.format()`
2. `f-string` (Python 3.6+)
3. `%` formatting

String formatting

- Formatted string literals (i.e., f-strings) provide a way to embed expressions inside string literals, using a minimal syntax.
- It works by adding a `f` char at the beginning of a string, and then a placeholder `{}` which can contain variables, functions, etc. to format the value.
- Cleaner and easier to write.

String formatting

Best practice:

Use f-strings for readability and performance (Python 3.6+).

```
name = "Alice"
age = 30

# 1. Using str.format()
formatted_str = "Name: {}, Age: {}".format(name, age)
print(formatted_str)  # Output: Name: Alice, Age: 30

# 2. Using f-strings (Python 3.6+)
formatted_str = f"Name: {name}, Age: {age}"
print(formatted_str)  # Output: Name: Alice, Age: 30

# 3. Using % formatting
formatted_str = "Name: %s, Age: %d" % (name, age)
print(formatted_str)  # Output: Name: Alice, Age: 30
```

Encoding vs Decoding

Encoding vs Decoding

Encoding: converting a string to bytes for storage or transmission.

- Often used for writing to files or sending data over the network.

Decoding: converting bytes back to a string

- Often used for reading data from files or receiving data over the network.

Encoding vs Decoding

encode() - converts a string value into a collection of bytes, using the specified scheme.

```
text = "Hello, World!"


# Encoding
encoded_text = text.encode('utf-8')
print(encoded_text)  # Output: b'Hello, World!'
```

Encoding vs Decoding

decode() - converts a collection of bytes into a string value using the specified scheme.

```
text = "Hello, World!"


# Encoding
encoded_text = text.encode('utf-8')
print(encoded_text)  # Output: b'Hello, World!'


# Decoding
decoded_text = encoded_text.decode('utf-8')
print(decoded_text)  # Output: Hello, World!
```

Conclusion

- Common string operations
- String Indexing
  - Accessing items from start to end
  - Accessing items from end to start
- String Slicing
  - Slicing strings from start to end
  - Slicing strings from end to start
- String formatting
  - Use f-strings as default approach