Strings

SHARP SIGHT

What you'll learn

- What strings are
- How to create strings
- Retrieving characters and substrings
- String concatenation
- String manipulation with functions and string methods

String basics

"Strings" are just character data

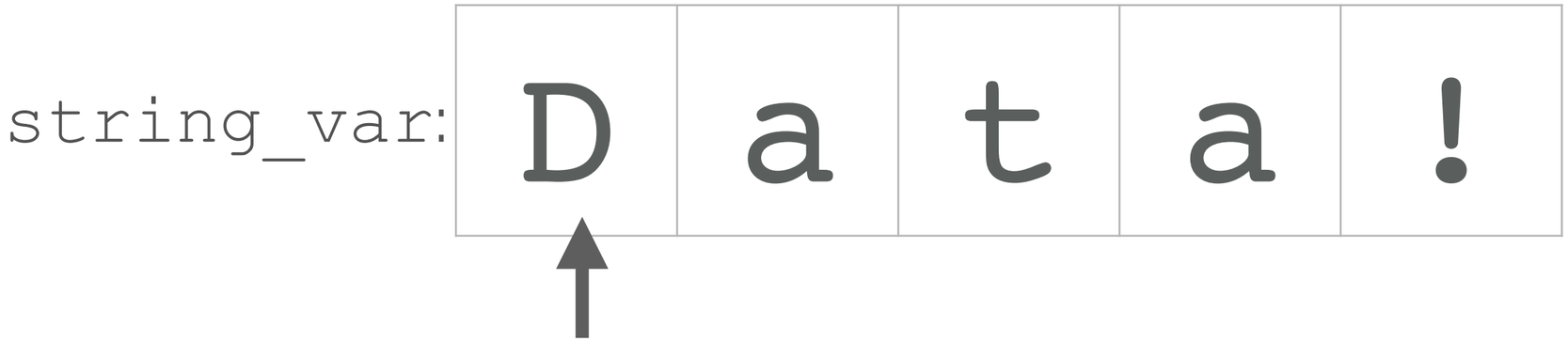
```
string_var = 'Data!'
```

`string_var` is a variable that contains the string `'Data!'`

The characters `'Data!'` are a string

Technically, strings are sequences of characters

```
string_var = 'Data!'
```



The string 'Data!' is actually a sequence of 5 characters that are stored together in `string_var`.

Strings in Python 3 can use characters from almost any language

```
foreign_string = 'データ'
print(foreign_string)
'データ'
```

The Unicode character system has "over 137,000 characters covering 146 modern and historic scripts."

– Wikipedia, https://en.wikipedia.org/wiki/List_of_Unicode_characters

Creating Strings

We create strings with quotes

- Both single quotes and double quotes will work

```
string1 = 'This is a string'
print(string1)
This is a string

string2 = "Also a string"
print(string2)
Also a string
```

Single quotes are okay

Double quotes are also okay

How are single and double both legal?

- We can use quotes within quotes
  - If you want to use a single quote, inside a string, enclose the string inside double quotes, and visa versa

Single quote inside string

```
string3 = "Python's a good language."

string4 = 'He said "I like Python." '
```

Double quote inside string

Simple operations on strings

Important functions for strings

- Python has several built-in functions that operate on strings
  - Note: not all of these are specific to strings!

Basic string functions

Function	What it does
`print()`	prints the string
`len()`	return string's length
`str()`	convert other data type to string

`print()` PRINTS STRINGS

- We can print strings using the **`print()`** function

- **`print()`** strips out the quotes on that enclose the string

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

`len()` GETS THE LENGTH OF A STRING

- **`len()`** counts the number of characters in a string

```
len("This is a string")
16
```

`str()` CONVERTS DATA INTO A STRING

- **`str()`** converts other data types to strings

The initial data is a number (a `float`)

The output after using `str()` is actually a string.

```
numVar = 4.4

str(numVar)
'4.4'
```

You can combine two strings using the + operator

- This is called "string concatenation"

Note: you need to manually add in a space between strings.

```
stringPart1 = 'Winter is '
stringPart2 = 'coming'

newString = stringPart1 + stringPart2

print(newString)
Winter is coming
```

Here, we're combining two strings with the + operator.

String Indexing

(working with parts of strings)

Indexing overview

- An index is a way to give a numeric ‘address’ to an element in a sequence
- The characters of a string can be accessed by numeric index
- Python uses 0-based index
  - The index is the offset from the first character
- We can also “slice” strings using indexes
  - Get a substring, instead of just one character

Remember: strings are sequences of characters

```
string_var = 'Data!'
```

	0	1	2	3	4
`string_var`:	D	a	t	a	!

The characters in `'Data!'` are actually elements of a a sequence (the string)

Each character has an index (a position in the sequence)

```
string_var = 'Data!'
```

The numeric position of each character is called the "index"

0	1	2	3	4
D	a	t	a	!

Python indexes start at zero

```
string_var = 'Data!'
```

The first character in a string has an index of 0

0	1	2	3	4
D	a	t	a	!

We retrieve characters from strings by using brackets, [ ]

```
string_var = 'Data!'
string_var[1]
```

index:	0	1	2	3	4
string_var:	D	a	t	a	!

The code **`string_var[1]`** will retrieve the character **`a`**

Why start at zero?

- An index represents an *offset* from the first character

```
string_var = 'Data!'
string_var[1]
```

index:	0	1	2	3	4
`string_var`:	D	a	t	a	!

So, **`string_var[1]`** returns the character one position to the right of the first character

Get the last character of a string with the index –1

```
string_var = 'Data!'
string_var[-1]
```

index:	0	1	2	3	4
string_var:	D	a	t	a	!

So, **`string_var[-1]`** retrieves the last character

String Slicing

(creating substrings)

Substrings in Python

- Python does not have a substring function

- Instead, we “slice” strings to create substrings

- Use the bracket notation to slice a string
  - Example: **`string_var[1:5]`**
  - indicate start and end of substring, inside of brackets

Get substrings using "bracket" notation

- Use a start and stop index, separated by a colon

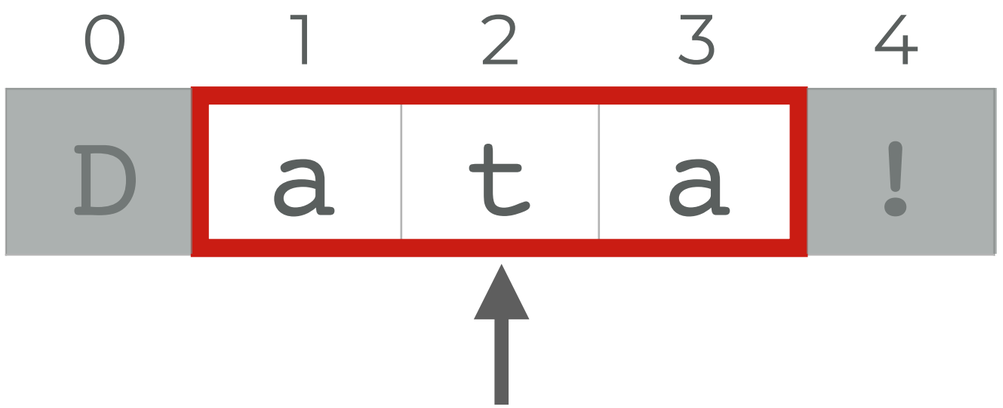
```
your_string[ start-index : stop-index ]
```

The numeric index of the first character of the substring

The numeric index of the character where the substring stops (this is not included!)

An example of a substring

```
string_var = 'Data!'
string_var[1:4]
```

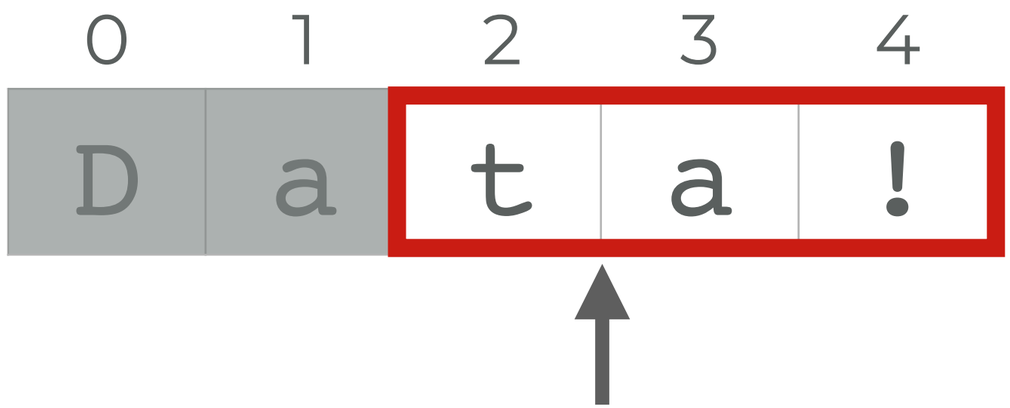


The substring includes the start index but *excludes* the stop index

If you remove the stop index, the substring will go to the end of the string

```
string_var = 'Data!'
string_var[2:]
```

We did not provide a stop index!



The substring includes the characters from the start index all the way to the end of the string

A few notes on slicing

- Slicing is important
  - Learn this well!

- Slicing can also get more complicated
  - We're keeping things simple here

- Slicing, used in more complex data structures
  - Lists
  - Arrays
  - DataFrames
  - NumPy arrays

String Methods

What are methods?

- Function vs method?
  - Methods are associated with classes, but functions are not
  - To put it simply, methods are specific to classes

Use string methods by using "dot notation"

`string_var` is a string

```
string_var = 'Data!'
string_var.count('a')
2
```

We use string methods by typing a dot ("`.`"), followed by the method name

`count()` is a string method that counts the occurrences of a given character

Most commonly used string methods

- Python has many useful string methods
  - This is an abridged list of string methods

Method	What it does
`lower()`	convert characters to lower case
`upper()`	convert characters to upper case
`count()`	count number of occurrences of a sequence of characters
`find()`	find the offset of first occurrence of a sequence of characters
`replace()`	replace a sequence of characters with a new sequence of characters

Recap

Recap of what we learned

- How to create strings
- Retrieving characters and substrings
- String concatenation
- String manipulation with functions and string methods