Strings, Part 2

Strings are Immutable

Immutable means unchangeable. Once you have a string, you can't change it - you have to remake it.

This is why you have to set all the string methods equal, like this: `string = string.lower()` instead of just calling `string.lower()`

The string methods don't modify your original string - they return a new string.

Exercise - Immutability

What is the output when you run this code?

```
my_string = "hello"

my_string.upper()
print(my_string)

my_string = my_string.upper()
print(my_string)
```

Indexing

Indexing is how we access an individual thing inside a group.

We use square brackets [] after the variable name to denote indexing.

Indexes start at 0 and go to the length of the object minus 1.

So if our string is ‘hello’ we would go from 0-4

- `mystr = "hello"`
- `mystr[0]` is `"h"`
- `mystr[4]` is `"o"`, the same as `mystr[len(mystr)-1]`

Indexing in Reverse

You can use a negative number to start at the end of the string

`-1` is the last index of a string, the same as `len(string)-1`

`mystr = "hello"`

`mystr[-1]` is `"o"`

`mystr[-4]` is `"e"`

Slicing

We can pull sections of our object using slicing.

Instead of indexing using 1 number we instead use 2 or 3 numbers.

If we use 2 numbers it goes `[start:stop]`

- This **includes** `start`, but **excludes** `stop` (goes to one before `stop`)

If we use 3 numbers it goes `[start:stop:step]`

```
mystr = 'ILikeApples'
```

`mystr[1:5]` is `'Like'`

`mystr[0:len(mystr):3]` would be `'IKpe'`, same as `mystr[::3]`

Slicing in Reverse

You can also use reverse indexes when slicing.

You can use a negative number as your start, stop, or step.

To get a string backwards, you can leave start and stop blank:

`mystr = "hello"`

`mystr[::-1]` is `"olleh"`

Exercise

Write some code to print the second half of a string.

(If the string is odd length, you can choose whether to print the shorter or longer "half")

Example:

```
python
```

```
hon
```

Exercise - Valid Email

Write some code that takes input from the user and prints whether it's a valid email address. Make sure to sanitize the user input with .strip()

An email address is valid if:

- It has a "." at the third-to-last index
- It has exactly one "@" symbol, at the fifth-to-last index or earlier
- There is at least one character before the "@" symbol
- It doesn't have any spaces (doesn't contain " ")

If all these conditions are met, print True. Otherwise, print False.

- To do this, use boolean statements on your string.

Test your code on a few inputs to make sure it works!

Ascii and Escape Characters

By default, python strings use ASCII characters.

We can denote special characters using a backslash.

This is called an escape character.

Different combinations mean different things (see next slide).

There are ways to use different character sets like UTF-8, that support more characters like emojis and other languages.

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

Escape Characters

Most common escape chars:

`\'` Single Quote

`\"` Double Quote

`\\` Backslash

`\n` New Line

`\t` Tab

Other escape chars:

`\r` Carriage Return

`\b` Backspace

`\f` Form Feed

`\ooo` Octal value

`\xhh` Hex value

Each of these only counts as **one character** when you're indexing a string or taking the length of a string, even though it looks like more than one

Print Formatting

By default, the print function takes a string, or multiple items separated by commas, and ends with a new line.

What if you want to have more control and customization?

You can choose to end the print statement with a different character, separate by a different character if you're using commas, or format your print statement in different ways

end=

By default, print statements end in a new line, or `"\n"`

If you want to end with a different character, use the `end=` keyword, and put whatever character you want after the equals.

- The default is `end="\n"`

Exercise: What is the output of this code?

```
print("hello", end=" ")
print("world", end="")
print("!", end="\n")
```

sep=

By default, if you print multiple values separated by commas, they will be separated by spaces

The `sep=` keyword lets you choose a different character to separate them.

- The default is `sep=" "`

Exercise: What is the output of this code?

```
print("today I woke up at ", 8, " am", sep="")
```

Exercise - sep and end

Get input from the user and store it in a variable called `userin.`

Then print the user input. The output should follow exactly this format, including the colon and period at the end:

`You entered: userin.`

Where `userin` is what you got from the user.

You must format the print statement like this:

```
print("You entered",userin)
```

How can you add `sep` and `end` keywords to get the exact formatting shown above?

Formatting with .format()

Print formatting is a more streamlined way to print variables within strings than using commas or concatenation.

Print a string, and anywhere you want to place a variable, put curly brackets {}. Then call .format() on the string and pass in the variable names to substitute for the brackets.

You can also specify more within the brackets such as naming each placement, data type, number of decimals in a float, etc.

Formatting with f-strings

A newer version of print formatting added in Python 3.6, and a better alternative to using .format()

Put `f` before your string, and in curly brackets you can directly put the names of any variables you want to print.

Optionally, you can put a colon after the variable name and then add a data type specifier like `.2f` (print a float to two decimal places).

```
long_number = 12.456789012345
print(f"This is our number: {long_number:.2f}")
```

```
This is our number: 12.46
```

Exercise - Print Formatting

You need to write a script that will generate an email to a customer who has just made a purchase. You have 3 variables:

- `name`, which stores the customer's name as a string
- `product`, which stores the product name as a string
- `price`, which stores the price of the product as a float

Use an f-string to generate an email message with the following text, and print it. Make sure to round the price to 2 decimal places. The email should be one multi-line string.

```
Dear {name},

Thank you for your purchase of a {product}. Your credit card has been charged ${price}.

We appreciate your business and look forward to serving you again.

Sincerely,

The ABC Company
```

Home Practice

Write some code that takes a string and tells if it is a palindrome (same forwards and backwards).

Hint: Use indexing/slicing and boolean expressions

Examples:

`racecar: True`

`cat: False`

Resources

Strings:

https://docs.python.org/3/library/string.html

https://www.w3schools.com/python/python_strings.asp

https://realpython.com/python-strings/

More info about print formatting:

https://realpython.com/python-f-strings/