print("Hello, World!")

NCSS Challenge - Beginners
Week 3 Part 1
What will we cover?

- Manipulating strings;
- Uppercase and lowercase;
- Parts of strings;
- Characters in a string.
○ Implement simple digital solutions as visual programs with algorithms involving branching (decisions) and user input (ACTDIP011)
○ Recognise different types of data and explore how the same data can be represented in different ways (ACTDIK008)
○ Define problems in terms of data and functional requirements drawing on previously solved problems (ACTDIP017)
Manipulating strings
We’ve seen how to make decisions with strings:

```python
name = input('What is your name? ')
if name == 'Andrew':
    print("That's my brother's name!")
```

But what about what we want to check isn’t exactly equal?
Substrings within a string

- We can check if a smaller string is in a bigger string:
  ```
  msg = 'concatenation is fun'
  print('cat' in msg)  # True
  print('dog' in msg)  # False
  ```

- We can also check:
  ```
  print('cat' not in msg)  # False
  ```
Substrings as conditions of if statements

We can use these checks as the condition in if statements!

```python
name = input('Enter your name? ')  
if 'x' in name:  
    print('Your name contains an x!')
else:  
    print('No "x" in your name.')
```
Now is a great time to ask students to consider any limitations of the previous code snippet. Recall that computers are very precise and exact. Will that code really test if any ‘x’ is in their name? Perhaps get students to shout out names. (Xavier is a good one!) This leads on nicely to modify case, which we will look at next.
Test it out!

Try the first question now!
2. UPPERCASE and lowercase
We can use string *methods* to modify a string:

```python
msg = 'I know my ABC'
print(msg.lower())
   → i know my abc
```

The `msg` string contains a message in mixed case, and when you call the `lower` method it gives back a new message in lowercase only.
String methods don’t change what’s in the variable!

- However, the contents of the variable don’t change!
  ```python
  msg = 'I know my ABC'
  print(msg.lower())
  → i know my abc
  print(msg)
  → I know my ABC
  ```

- We need to save it in a new variable to keep it!
  ```python
  new_msg = msg.lower()
  ```
Changing text to uppercase

- Just as `lower` changes text to lowercase, `upper` changes it to uppercase!

```python
msg = 'I know my ABC'
new_msg = msg.upper()
print(new_msg)
# → I KNOW MY ABC
```

Note that the original contents of `msg` haven’t changed!

```python
print(msg)
# → I know my ABC
```
Teacher Aside! Frequent hiccup concept!

- Students are often confused about what makes a change to the variable, and what they need to save in a new variable!
- You can save a changed version in the same variable!

```python
msg = 'I know my ABC'
msg = msg.upper()
print(msg)
→ I KNOW MY ABC
```
Testing the case of a string

- Instead of changing the case of a string, we might just want to test it!
  
  ```python
  msg = 'a lowercase string!'
  print(msg.islower())
  → True
  print(msg.isupper())
  → False
  ```

- We can also use this as the condition in **if** statements!
Test it out!

Try the second question now!
3 Parts of a string
How long is a (piece of) string?

- We can find out how long a string is using the `len` function:
  
  ```python
  print(len('Hello World!'))
  → 12
  ```

- This counts all the characters, including the 5 letters in "Hello", the space, the 5 letters in "World", and the exclamation mark.
Counting specific characters in a string

- We can find out how long a string is using the `len` function:
  ```python
  msg = 'Hello World!
  print(msg.count('l'))
  → 3
  ```

- You can also count multi-character strings:
  ```python
  msg = 'Hello World!
  print(msg.count('ll'))
  → 1
  ```
Teacher Aside! Method review

- The convention for calling string methods is that the string we are manipulating comes first, and then the method name, with any other information that is required passed in as arguments.

```python
msg = 'I know my ABC'
msg.upper()
```
Replacing parts of a string (a *substring*)

- The replace method `replace` a substring:
  ```python
  msg = 'Hello World!'
  print(msg.replace('l', 'X'))
  → HeXXo WorXd!
  ```

- Similarly, can also replace multi-character strings:
  ```python
  msg = 'Hello World!'
  print(msg.count('Hello', 'Goodbye'))
  → Goodbye World!
  ```
Test it out!

Try the third question now!
4. Characters in a string
We can access a single character using the square bracket subscripting or indexing operation:

```python
msg = 'hello world'
print(msg[0])  # h
print(msg[1])  # e
```

Careful! In computer science, we start counting from 0 rather than from 1!
Accessing parts of a string (from the end!)

- We can also access a single character indexing from the end of the string, using negative numbers:

```
msg = 'hello world'
print(msg[-1])  # Output: d
print(msg[-5])  # Output: w
```
Teacher Aside! Ask the class!

- This is a great opportunity to ask the class to guess at which letter they think would be referenced by a specific index.
- It’s also a good chance, before the next slide, to ask them what they think might happen if you entered an index that doesn’t exist - e.g. character 15 in a 12 character string.
Teacher Aside!

- A common question is why, if we count from 0 for the beginning of a string, don’t we count from -0 at the end of a string.
- Python works by evaluating things from the inside, out. So Python would evaluate -0, which is the same as 0, and then look up the string index 0!
- So, name[0] is the same as name[-0]
Accessing parts of a string that don’t exist

- What happens if we index beyond a string’s length?
  ```python
  msg = 'hello world'
  print(msg[10])
  → d
  print(msg[11])
  → Traceback (most recent call last)...
      IndexError: string index out of range
  ```

- The string has 11 characters, but we are trying to access character 12. *(Remember we count up from 0!)*
Test it out!
Try the fourth question now!
Any Questions?

Find me at:
@groklearning
nicky@groklearning.com