Today's Topics:

- Number and string types
- String operators and methods

LEE WILKINS

OFFICE : 6C.9

CONTACT: MIO

INTRODUCTION TO COMPUTER PROGRAMMING IN ENGINEERING AND SCIENCE

How do we make a variable best_var and give a value of 6?

How do we multiply best_var by 2?

How do we make a variable other_var and give a value of "I like brownies"?

How do we tell Python to print other_var?

QUICK REVIEW

Reserved words (Can’t be used to name variables)

False def if raise None del import return True elif in try and else is while as except lambda with assert finally nonlocal yield break for not class form or continue global pass

snake_case or camelCase

Can’t contain special characters !@#$%^&*()+

Can’t start with a capital letter

Must be clear and make sense, no single letters (be descriptive)

VARIABLE NAME REVIEW

Each variable has a data type.

String: str A series of characters: “A string” or ‘a string’

Integer: int A whole number: 3 or 59

Float: A floating number, a decimal point: 3.4 or 1.0 or 5.224

Boolean: bool A true or false value

Complex: A set of numbers (real and imaginary): complex(3,5)

List / Array: a list of items [10, 30, 5, 30]

PYTHON DATA TYPES

You can check the type using type(varName). Here, I am making a variable and printing the result of type.

```
myIntExample = 10
print(type(myIntExample))
>> <class 'int'>

myFloatExample = 10.0
print(type(myFloatExample))
>><class 'float'>

myStringExample = "10"
print(type(myStringExample))
>><class 'string'>

myListExample = [10, 10, 10]
print(type(myListExample))
<class ‘list’>

myBoolExample = True
print(type(myBoolExample))
<class 'bool'>
```

CHECK TYPE

Concatenation

Glue two strings together end to end

'Hello' + 'World' ⟹ 'HelloWorld'

You can concatenate more than two strings at once

'Hello' + ' ' + 'World' ⟹ 'Hello World'

Repetition

Repeat a string multiple times

'Hello' * 3 ⟹ 'HelloHelloHello'

String operators also follow BEDMAS/PEDMAS

STRING OPERATORS

String operators also follow BEDMAS/PEDMAS

```
greeting = "hello"
tone = "!"
print(greeting + tone*3)
# hello!!!
print((greeting+tone)*3)
# hello!hello!hello!
```

STRING OPERATORS

STRING INDEXING

Indexing is counting through the characters (letters, digits, etc) in a string

Python uses zero-based indexing

string[0] gives us the first character of the string

```
greeting = "hello"
print(greeting[0])
```

Negative index numbers count backwards from the end of the string

string[-1] gives us the last character of the string

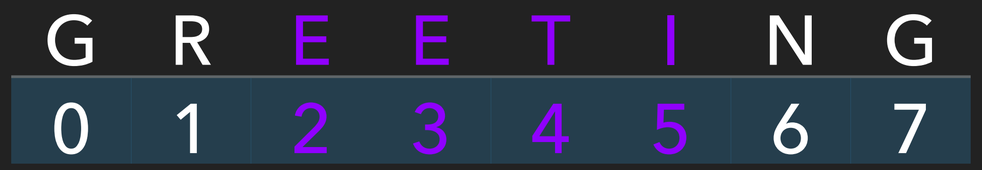
```
greeting = "hello"
print(greeting[-1])
```

We can use slicing to fetch a **substring** – a part of a string

some_string[2:6]

This will give us the substring from the 3rd character to the 6th

```
greeting = "greeting"
print(greeting[2:6])
>> eeti
```



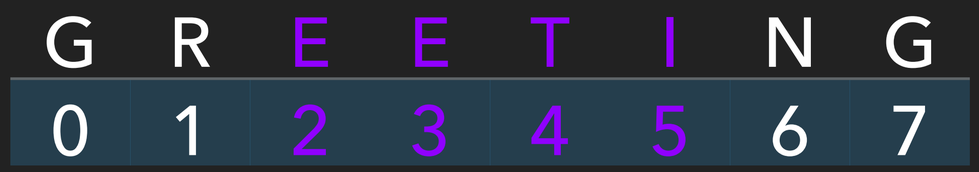
Remember, Python uses zero-indexing

The 7th character (index 6) is not included

You may find it helpful to imagine the indexes as counting *between* the characters

STRING SLICING

```
greeting = "greeting"
print(greeting[6])
# >> n just 6
print(greeting[:6])
# >> greeti before 6
print(greeting[6:])
# >> ng 6 and after
```



STRING SLICING

Interpolation lets us use variables inside of a string

In Python, this is called **st**ring formatting or an f-string

greeting = f'Hello {name}, it is {day_of_week} today!'

f before the starting quote mark

Variable names in curly braces { }

This is new syntax as of Python 3.6

```
name = "lee"
day_of_week = 5
greeting = f'Hello {name}, it is {day_of_week} today!'
print(greeting)
```

STRING INTERPOLATION

Sometimes in a string, we want to use a special character that Python might misinterpret, like ' or " or { }

These characters need to be **escaped** so Python understands they're part of the string

In Python, we can escape a character by putting a \ in front of it

e.g.: print('It\'s raining outside.')

This means the \ character also needs to be escaped if it's part of the string (\\)

https://www.w3schools.com/python/gloss_python_escape_characters.asp Here is a list of them

\n is for a new line

A NOTE ON SPECIAL CHARACTERS...

A **method** is a function that that acts on a specific object

Python uses dot notation for methods

Some methods **return** (give back to us) a property of the object

```
print(greeting.count("e")) # counts the number of es, returns 2
print (len(greeting)) # tells us the length of the string (8),
#length worts differently (sorry)
```

Some methods return a modified version of the object

```
print(greeting.upper())
# prints GREETING
print(greeting.lower())
# prints greeting
https://www.w3schools.com/python/python_ref_string.asp find more here
```

STRING METHODS

```
#here I am printing my string, note the case.
my_test_string = "hEllo thIS is a tESt"
print(my_test_string)
# here I just print the variables, but I am not storing them anywhere
# the methods return the value inside the print function
print(my_test_string.upper())
print(my_test_string.lower())
# but my variable still has its mixed case
print(my_test_string)
#here I am storing the upper and lower case strings in new variables,
# but not  printing them
my_test_string_uppercase = my_test_string.upper()
my_test_string_lowercase = my_test_string.upper()
# now i am printing my new variables, with their upper and lower case contents
print(my_test_string_uppercase)
print(my_test_string_lowercase)
```

STRING METHODS, STORING AND PRINTING

This can be useful for cleaning up strings that the user inputs OR for clearing up data. For example, data might have extra spaces, capital letters or characters that mean your strings do not match.

“mystring” will not equal “mystring “ or “myString”

.lower() can be used to make all letters lower case

.strip() will remove white space (there is .lstrip() and .rstrip() for right and left)

```
my_answer = input("what is the question?")
if "yes" in my_answer.lower():
    print("its there")
else:
    print("its not there")
```

STRING METHODS

```
my_Var = "this is a test   " # this one contains white space
if my_Var == "this is a test": #therefor it does not equal this string
    print("it matches") # this condition will never be met
    print("not yet")


my_Var = my_Var.strip() # I can use strip to remove white space
if my_Var == "this is a test": # now it equals the string
    print("it matches")
else:
    print("not yet")

my_Var = my_Var.replace(" ", "") # i can remove all white space by using replace
print (my_Var)
```

STRING METHODS

Lists are a way of storing many variables (in this case, strings). You can convert a string to a list.

We will talk more about lists later, but you can look ahead at the list methods https://www.w3schools.com/python/python_ref_list.asp

You can access list items like this

my_list[0] first item in list

my_list[1] second item in list

STRINGS TO LISTS

```
my_list = "a dog, a cat, a donkey"
print(my_list[0])
# prints a
print(type(my_list))
# prints <class 'str'>
my_list = my_list.split(",")
print(my_list[0])
# prints a dog
print(type(my_list))
# prints <class 'list'>
```

Split can be used to break a string up into a list. The split method takes one parameter, it is whatever delineator you want to split your string by, surrounded by quotes

In this example, we split the string by ","

But we could split a string by spaces, | , or any other word or series of words

```
my_list = "dog cat donkey”
my_list = my_list.split(“ “)
>> my_list[0] > “dog”
my_list = "dog|cat|donkey”
my_list = my_list.split(“|”)
>> my_list[0] > “dog”
```

STRING METHODS

CODE EXERCISES

USING MATH AND FUNCTIONS

NEXT CLASS: