Why Python Programming Language?

The python language is one of the most accessible programming languages available because it has simplified syntax and is not complicated, which gives more emphasis on natural language. Due to its ease of learning and usage, python codes can be easily written and executed much faster than other programming languages.

Python was created more than 30 years ago, which is a lot of time for any community of programming language to grow and mature adequately to support developers ranging from beginner to expert levels. There are plenty of documentation, guides, and Video Tutorials for Python language that learners and developers of any skill level or ages can use and receive the support required to enhance their knowledge in python programming language.

Due to its corporate sponsorship and big supportive community of python, python has excellent libraries that you can use to select and save your time and effort on the initial cycle of development. There are also lots of cloud media services that offer cross-platform support through library-like tools, which can be extremely beneficial.

Libraries with specific focus are also available like nltk for natural language processing or scikit-learn for machine learning applications.

Variables

Variables are containers for storing data values. Unlike other programming languages, Python has no command for declaring a variable. A variable is created the moment you first assign a value to it.

Variables do not need to be declared with any particular type, and can even change type after they have been set.

We declare a name on the left side of the equals operator ("="), and on the right side, we

assign the value that we want to save to use later, eg;

School = "Hamoye one school"

When you create a variable, the line where you assign the value is a step called

declaration. We've just declared a variable with a name of "school" and assigned it

the value of the string data type "Hamoye one school". This string is now stored in memory, and we're able to access it by calling the variable name "school".

Variable Naming Convention

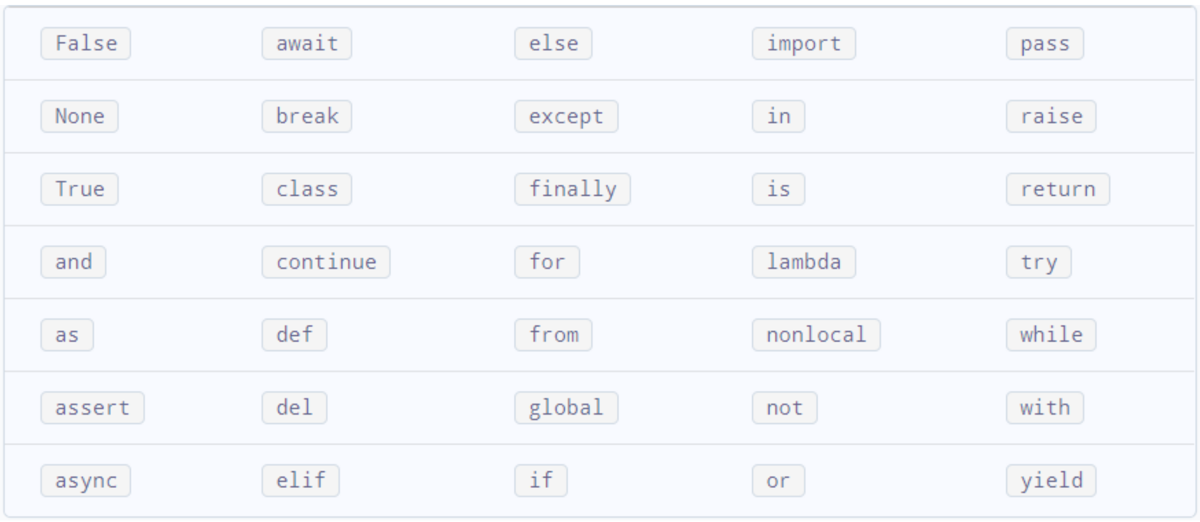
A variable can have a short name (like x and y) or a more descriptive name (age, carname, total_volume). Rules for Python variables:

1. A variable name must start with a letter or the underscore character

2. A variable name cannot start with a number
3. A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _).
4. A variable name must not be any reserved word or keyword, e.g. int, name.

Keywords in Python Programming Language

All the keywords except **True**, **False** and **None** are in lowercase and they must be written as they are. The list of all the keywords is given in the image below:



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

Fig 2.0 Python keyword

5. Variable names are case-sensitive (age, Age and AGE are three different variables)

Examples

First_name = "Zalihat" # string variable

Age = 23 # integer variable

Cgpa = 4.56 # float variable

Present = True # boolean variable

Data Types in python

Almost all languages use data types, they are essential to every program. Data types are how we define values, likes words or numbers. If I were to ask you what a sentence is made up of, you would probably reply with “words or characters.” Well, in programming, we call them strings. Just the same as we refer to numbers as their own data type as well.

Data types define what we can do and how these values are stored in memory on the computer. We will at some common data types;

1. **Integers**

These data types are often called integers or ***ints***. They are positive or negative WHOLE numbers with no decimal point. Integers are used for a variety of reasons, between math calculations and indexing, eg 30, 2,etc.

2. **Floats**

Anytime a number has a decimal point on it, they're known as floating point data types. It doesn't matter if it has 1 digit, or 20, it's still a float. The primary use of floats is in math calculations, although they have other uses as well eg 3.5, 4.0, etc.

3. **Booleans**

The boolean data type is either a True or False value. Think of it like a switch, where it's either off or on. It can't be assigned any other value except for **True** or **False**. Booleans are a key data type, as they provide several uses. One of the most common is for tracking whether something occurred. For instance, if you took a video game and wanted to know if a player was alive, when the player spawned initially, you would set a boolean to "True". When the player lost all their lives, you would set the boolean to "False". This way you can simply check the boolean to see if the player is alive or not. This makes fora quicker program rather than calculating lives each time.

4. **Strings**

Also known as "String Literals," these data types are the most complex of the four. The actual definition of a string is:

Strings in Python are arrays of bytes representing unicode characters.

To most beginners, that's just going to sound like a bunch of nonsense, so let's break it down into something simple that we can understand. Strings are nothing more than a set of characters, symbols, numbers, whitespace, and even empty space between two sets of quotation marks. In Python we can use either single or double quotes to create a string. Most of the time it's personal preference, unless you want to include quotes within a string. Whatever is wrapped inside of the quotation marks will be considered a string, even if it's a number, eg "Hello World!", "3432", etc.

Comments

Comments are like notes that you leave behind, either for yourself or someone else to read. They are not read in by the interpreter, meaning that you can write whatever you want, and the computer will ignore it. A good comment will be short, easy to read, and to the point. Putting a comment on every line is tedious, but not putting any comments at all is bad practice.

Comment starts with a # and python ignores anything after the #. Examples of comments

print(90) #prints two

Python Operators

Operators are used to perform operations on variables and values. Python divides the operators in the following groups:

1. Arithmetic operators
2. Comparison operators
3. Logical operators

Python Arithmetic Operators

Arithmetic operators are used with numeric values to perform common mathematical operations:

Name	Operator	Example
Addition	+	Z = x + y
Subtraction	-	Z = x-y
Multiplication	*	Z = x*y
Division	/	Z = x/y
Modulus	%	Z = x%y
exponential	**	Z = x**2

Fig 2.1 Python arithmetic operators

Python Comparison Operators

Comparison operators are used to compare two values:

Name	Operator	Example
Equal	= =	x == y
Not equal	!=	y != a
Greater than	>	x < y
Less than	<	x < y
Greater than or equal to	>=	x >= y
Less than or equal to	<=	x <=y

Fig 2.2 Python comparison operators

Python Logical Operators

Logical operators are used to combine conditional statements:

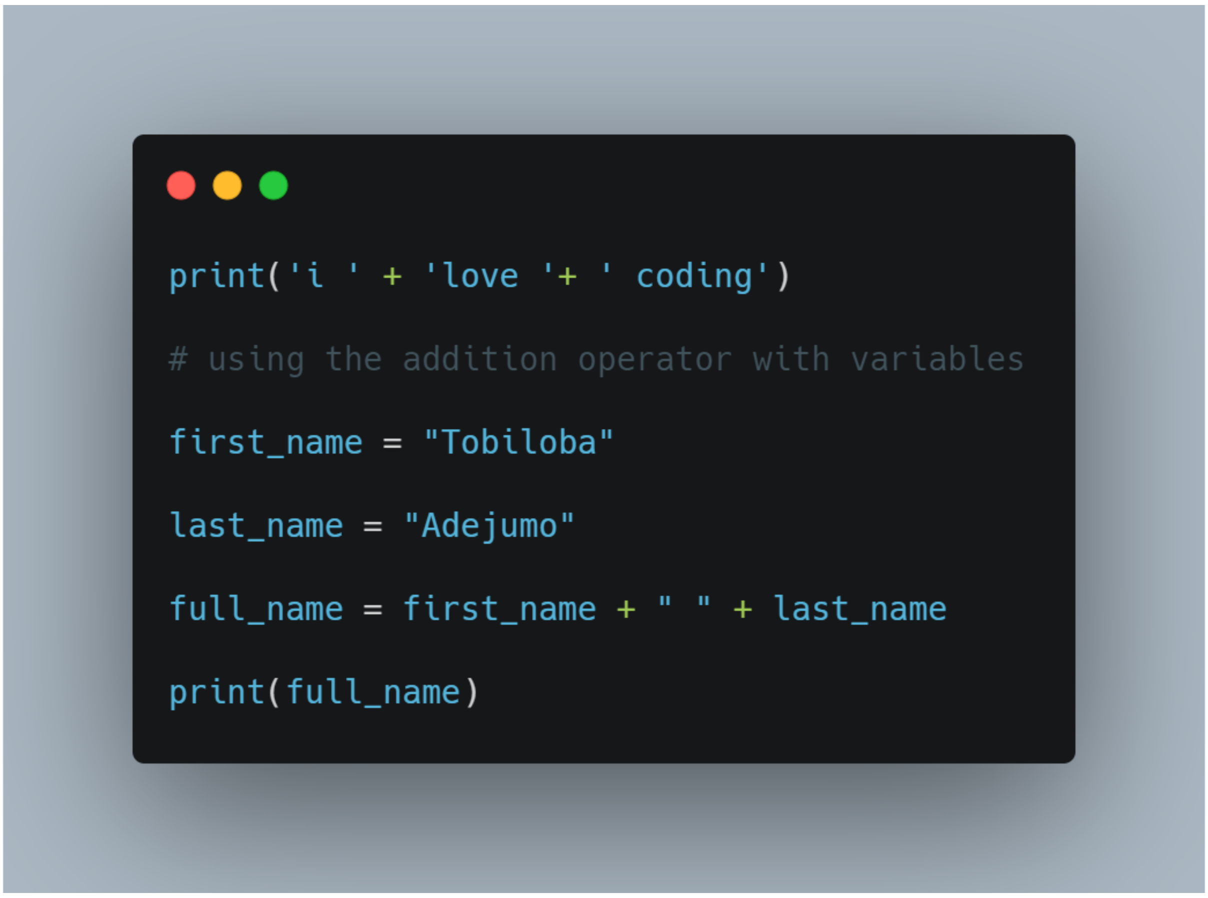
Operator	Description	example
and	Returns True if both statements are true	x < 5 and x < 10
or	Returns True if one of the statements is true	x < 5 or x < 4
not	Reverse the result, returns False if the result is true	not(x < 5 and x < 10)

Fig 2.3 Python logical operators

Working with strings

String Concatenation

When we talk about concatenating strings, I mean that we want to add one string to the end of another. This concept is just one of many ways to add string variables together to complete a larger string. For the first example, let's add three separate strings together:



```python
print('i ' + 'love '+ ' coding')

# using the addition operator with variables

first_name = "Tobiloba"

last_name = "Adejumo"

full_name = first_name + " " + last_name

print(full_name)
```

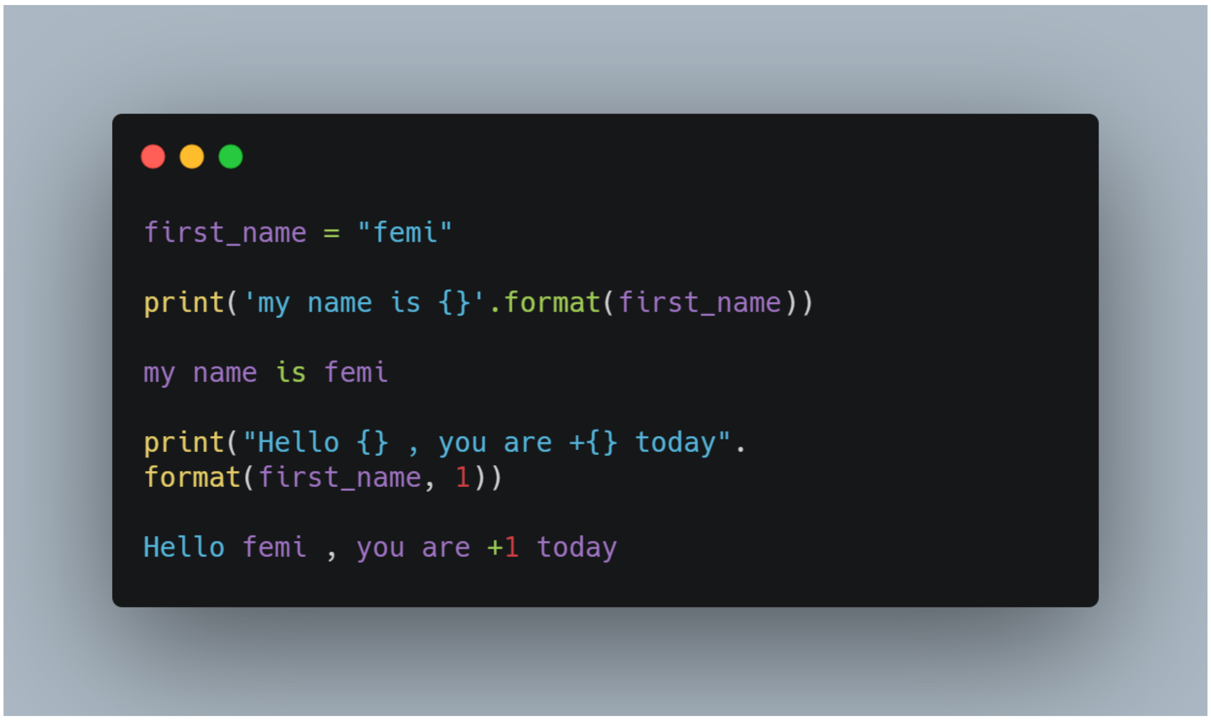
Note : you cannot concatenate a string and a number using the addition operator.

Formatting Strings

Earlier we created a full name by adding multiple strings together to create a larger string. While this is perfectly fine to use, for larger strings it becomes tough to read. Imagine that you had to create a sentence that used 10 variables. Appending all ten variables into a sentence is tough to keep track of, not to mention read. We'll need to use a concept called string formatting. This will allow us to write an entire string and inject the variables we want to use in the proper locations. Using this we can also concatenate a string and a number.

.format()

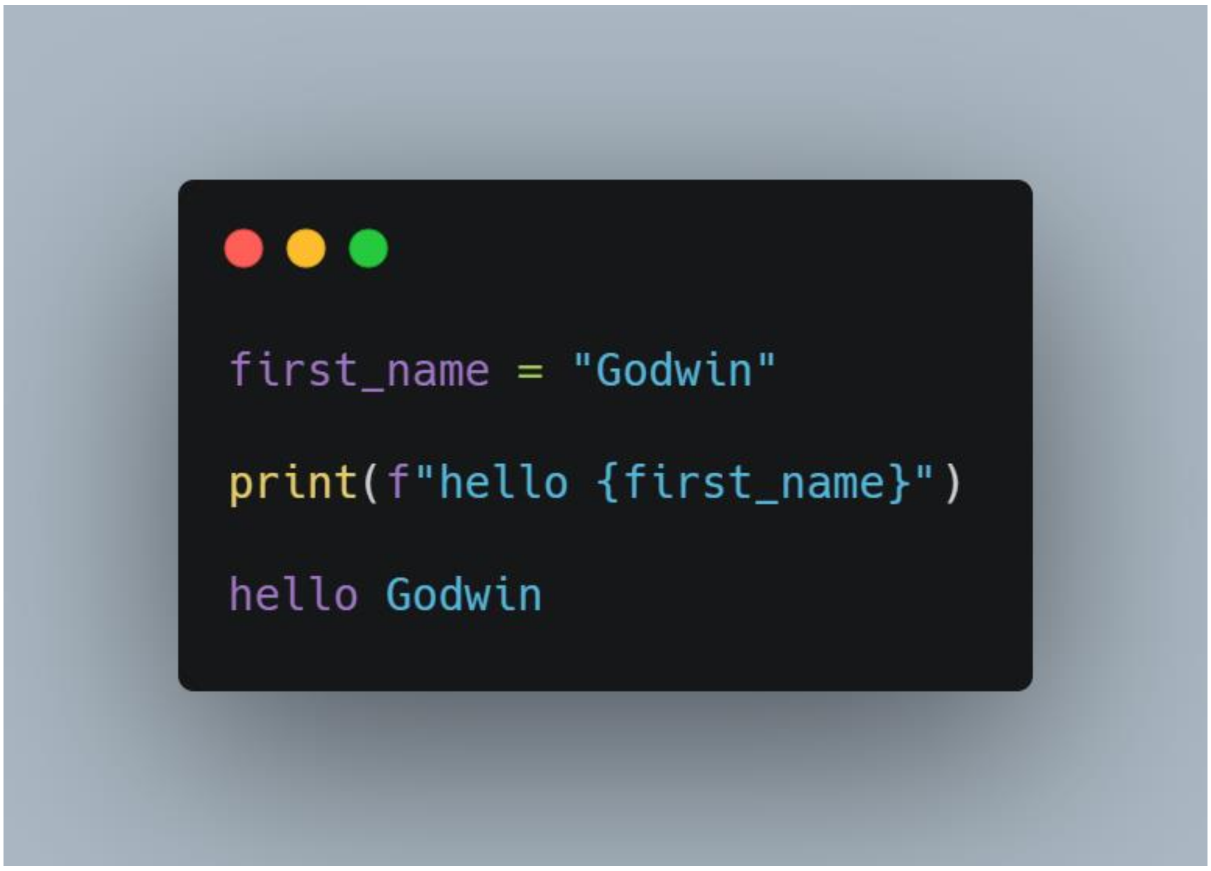
The format() method takes the passed arguments, formats them, and places them in the string where the placeholders { } are:



```
first_name = "femi"

print('my name is {}'.format(first_name))

my name is femi

print("Hello {} , you are +{} today".
format(first_name, 1))

Hello femi , you are +1 today
```

f Strings (New in Python 3.6)

The new way to inject variables into a string in Python is by using what we call f strings. By putting the letter "f" in front of a string, you're able to inject a variable into a string directly in line. This is important, as it makes the string easier to read when it gets longer, making this the preferred method to format a string. Just keep in mind you need Python 3.6 to use this; otherwise you'll receive an error. To inject a variable in a string, simply wrap curly brackets around the name of the variable. Let's look at an example:



```
first_name = "Godwin"

print(f"hello {first_name}")

hello Godwin
```

String Index

One other key concept that we need to understand about strings is how they are stored. When a computer saves a string into memory, each character within the string is assigned what we call an “index.” An index is essentially a location in memory. Think of an index as a position in a line that you’re waiting in at the mall. If you were at the front of the line, you would be given an index number of zero. The person behind you would be given index position one. The person behind them would be given index position two and so on.

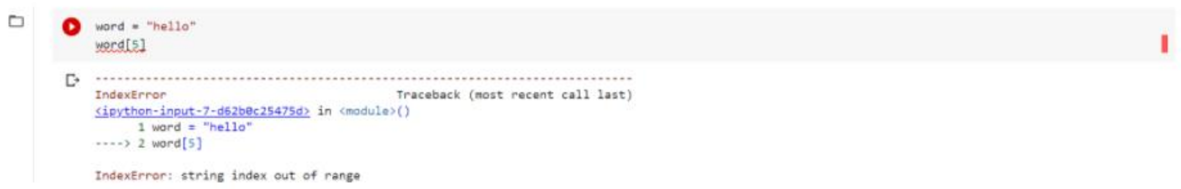
Note that in python and most programming languages index starts from 0 not 1.

Here are examples of string indexing:

```
word = "animal"

print(word[0])

print(word[1])

print(word[2])
```

As you can see from the example above, you index a string using a square bracket and the index.

Be very careful when working with indexes. An index is a specific location in memory. If you try to access a location that is out of range, you will crash your program because it’s trying to access a place in memory that does not exist. For example, if we tried to access index 5 on the “Hello”



```
word = "hello"
word[5]
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-7-d62b0c25475d> in <module>()
      1 word = "hello"
----> 2 word[5]

IndexError: string index out of range
```

String Slicing

I want to just quickly introduce the topic of slicing. Slicing is used mostly with Python lists; however, you can use it on strings as well. Slicing is essentially when you only want a piece of the variable, such that if I only wanted “He” from the word “Hello”, we would write the following:

print(word[0:2])

The first number in the bracket is the starting index; the second is the stopping index. We will touch on this concept later when we talk about lists.

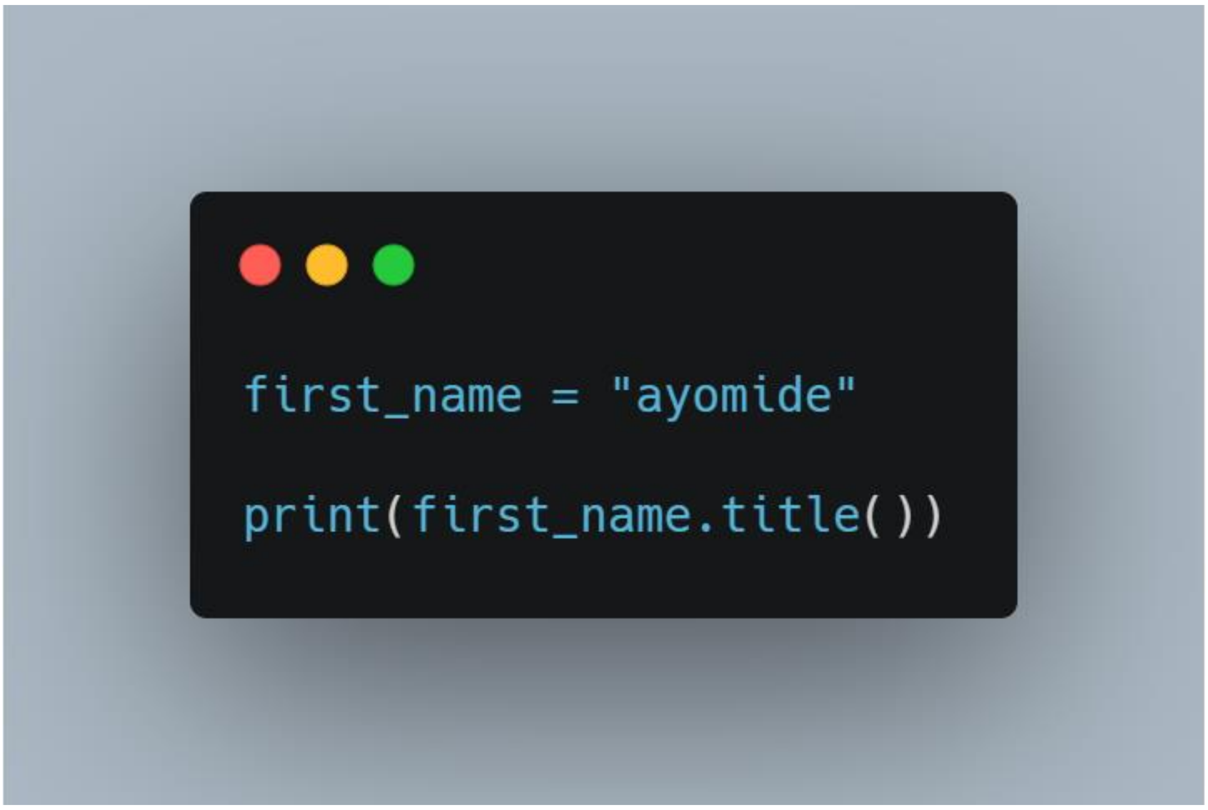
String Manipulation

In many programs that you’ll build, you’re going to want to alter strings in one way or another. String manipulation just means that we want to alter what the current string is.

Luckily, Python has plenty of methods that we can use to alter string data types.

1. **.title()**

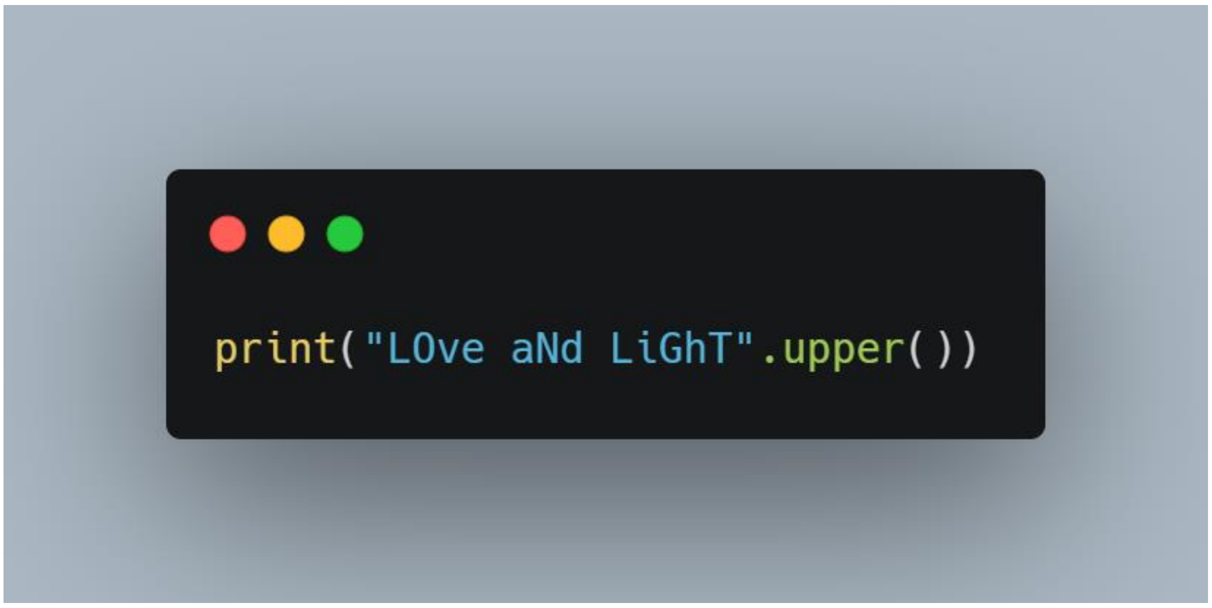
Often, you’ll run into words that aren’t capitalized that should be usually names. The title method capitalizes all first letters in each word of a string.



```python
first_name = "ayomide"

print(first_name.title())
```

2. **.upper()**

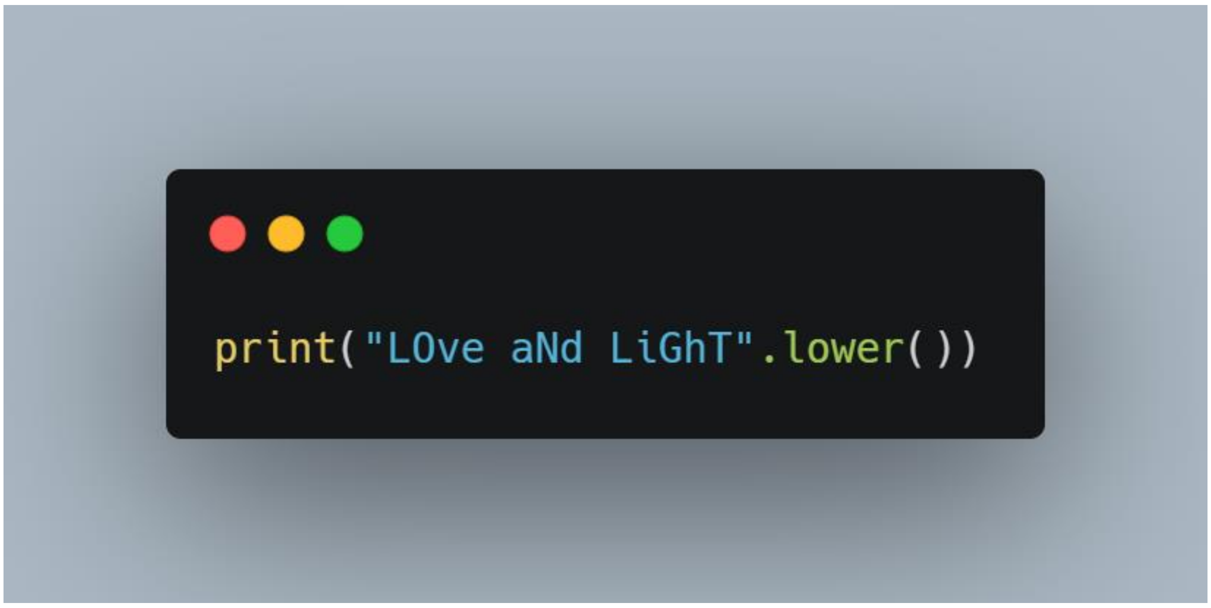
This method converts a string to uppercase.



```python
print("LOve aNd LiGhT".upper())
```

3. **.lower()**

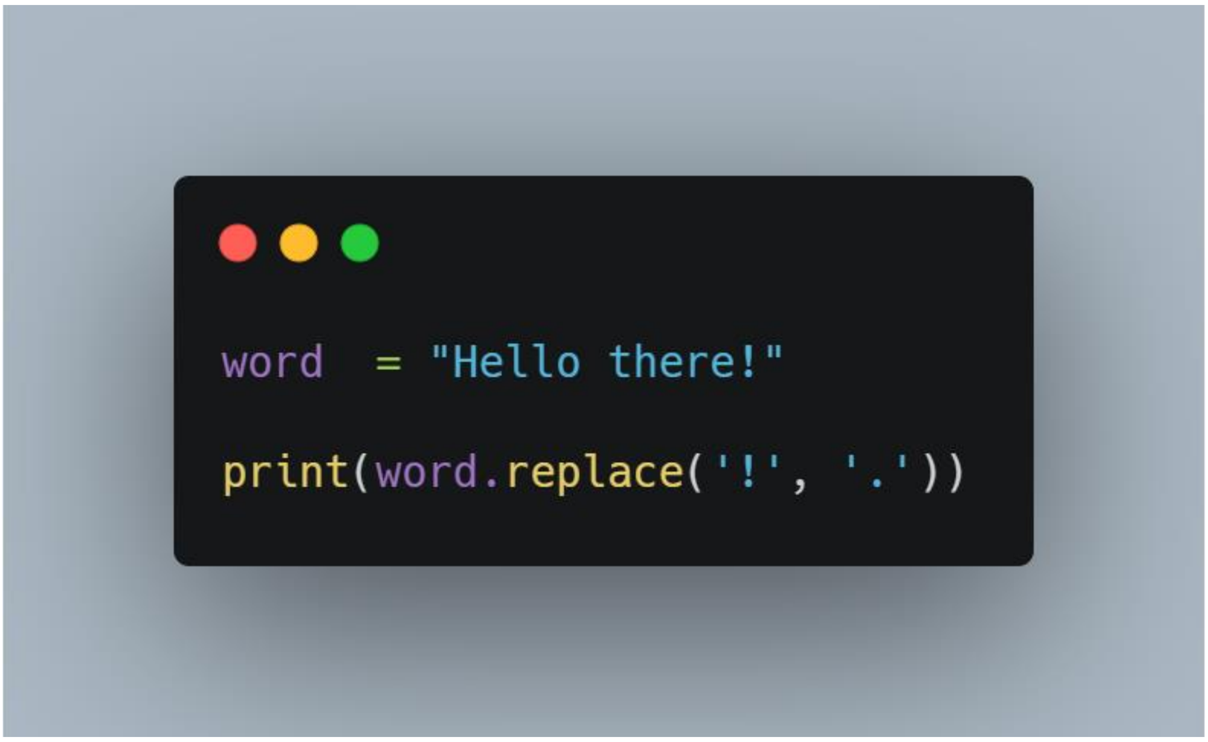
This is the opposite of the .upper(), it converts a given string to lowercase.



```python
print("LOve aNd LiGhT".lower())
```

4. **.replace()**

The replace method works like a find and replace tool. It takes in two values within its parenthesis, one that it searches for and the other that it replaces the searched value with:

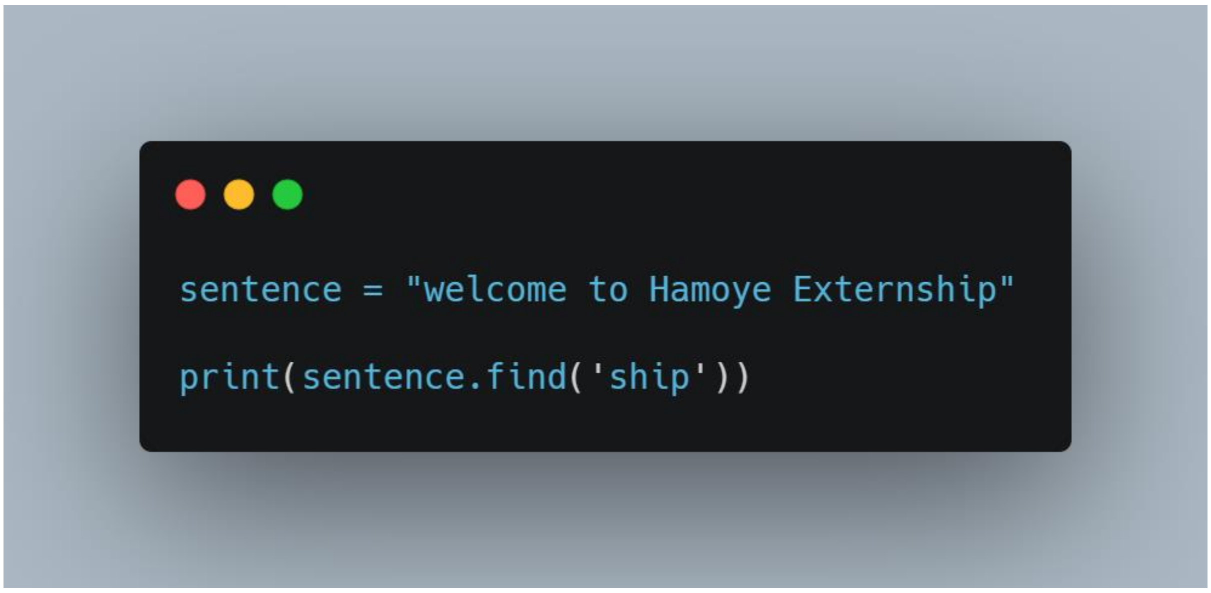


```python
word  = "Hello there!"

print(word.replace('!', '.'))
```

5. **.find( )**

The find method will search for any string we ask it to. In this example, we try to search for an entire word, but we could search for anything including a character or a full

Sentence:



```python
sentence = "welcome to Hamoye Externship"

print(sentence.find('ship'))
```

Note that the method returns the index of the word in the sentence.

In this course we only cover a few string methods. As we move forward we are going to use more methods, to learn more about string methods check the documentation here 3.6.1 String Methods.

Conditional Statements

In Python, the code executes in a sequential manner i.e. the first line will be executed first followed by the second line and so on. What will we do in a case where we have to decide that a certain part of code should run only if the condition is True? In such cases, Decision making is required, which is achieved through conditional statements.

IF Statement

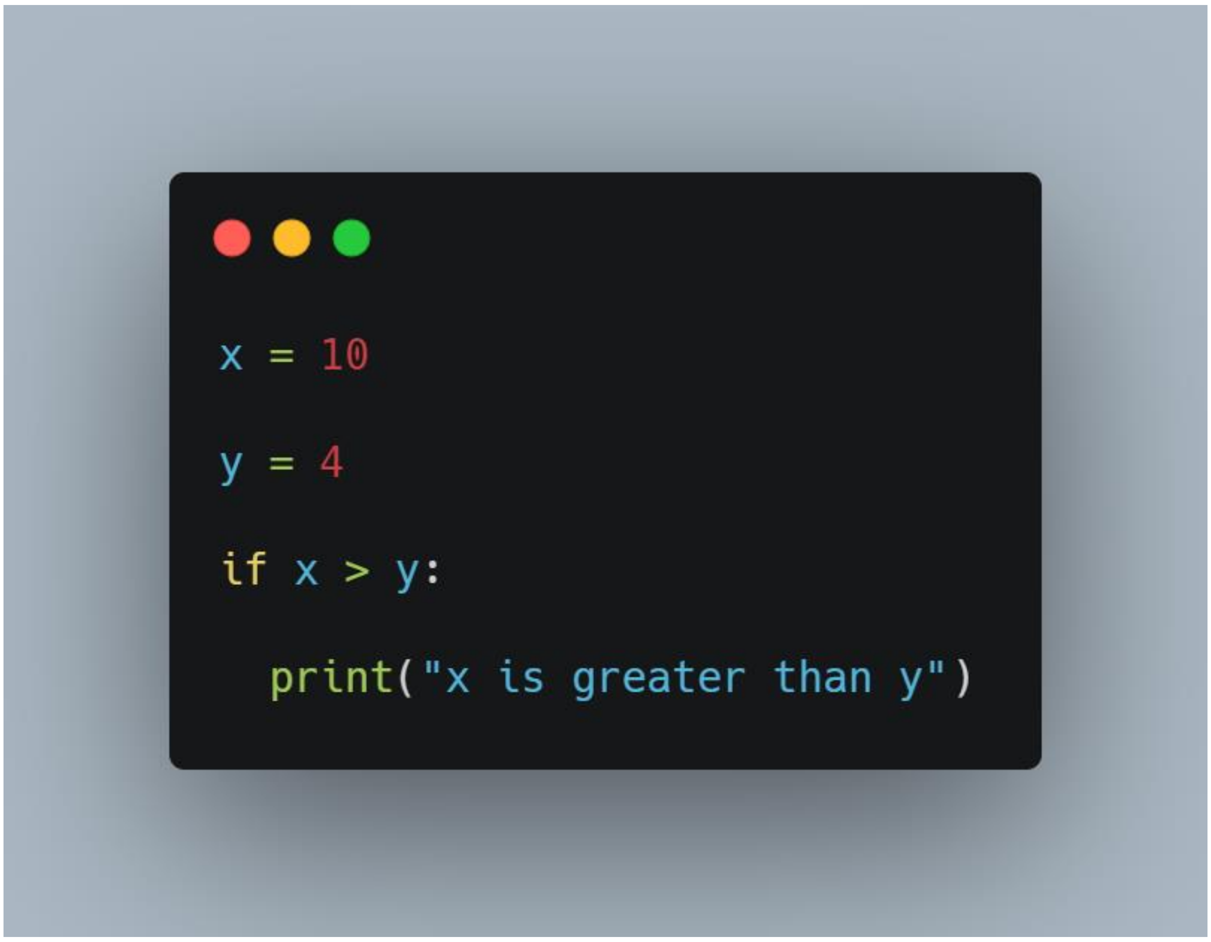
In Python, if statement is used when we have to execute a code block, only if a test condition is True. In this case, the program will evaluate the test expression and will execute statement(s) only if the text expression is True. An if statement is written by using the if keyword.

Syntax:

If (condition) :

Statement

Example:



```
x = 10

y = 4

if x > y:

  print("x is greater than y")
```

If else Statement

In Python, when we combine the else code block with the if code block, the if code block is executed only if the test condition is True, and the else code block is executed in the cases when the test condition is False. An if else statement is written by using the if else keyword.

Syntax:

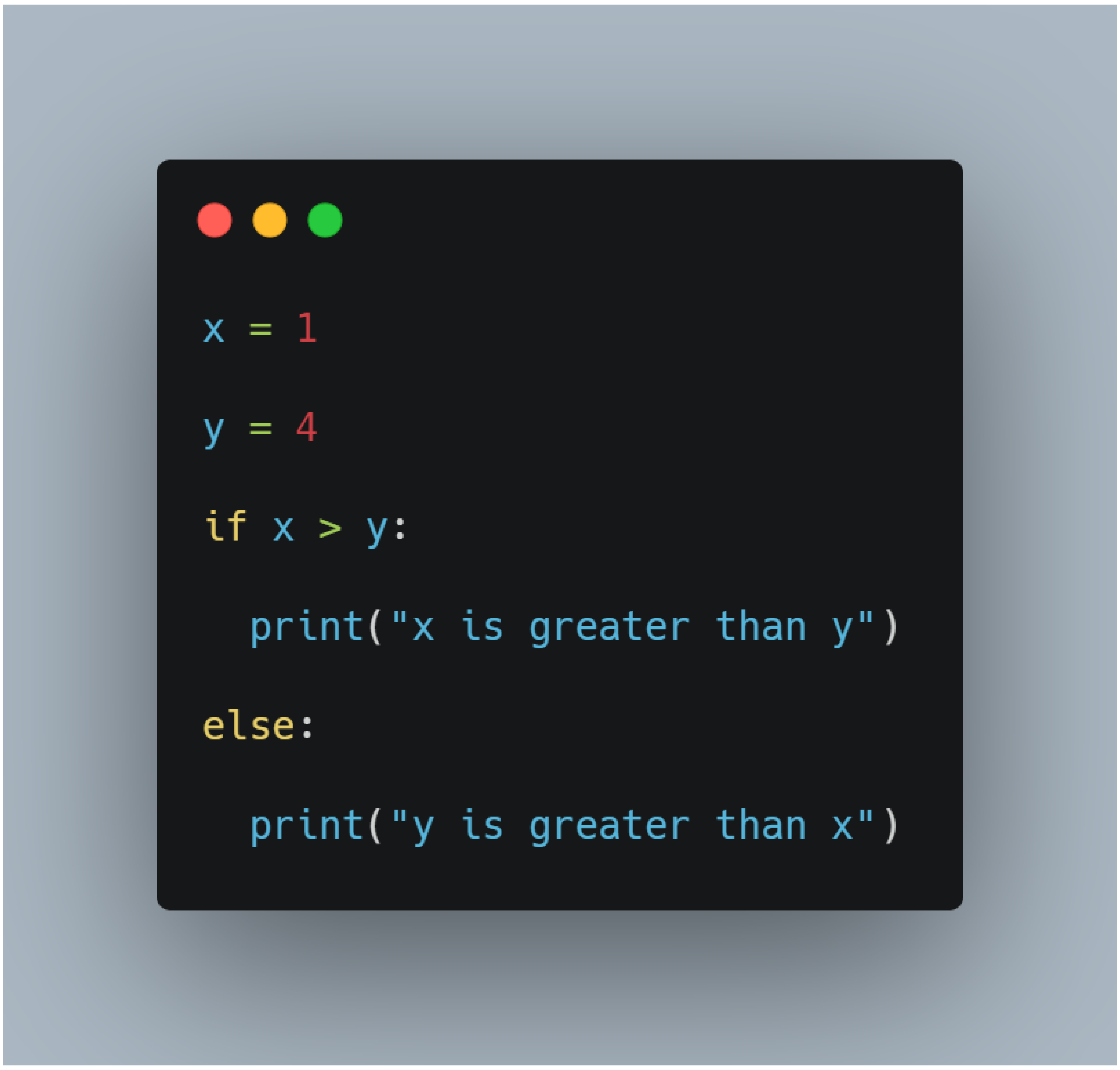
If (condition):

statement(s)

Else:

statement(s)

Example:



```
x = 1

y = 4

if x > y:

  print("x is greater than y")

else:

  print("y is greater than x")
```

Elif Statement

In Python, an elif statement is used when we have to check multiple conditions. elif is short form for else if. In such cases, firstly, the if test condition is checked, If it is true, then the if code block is executed and if it is false, then the next elif test condition is checked and so on. If all the conditions are false, then the else code block is executed. An if else statement is written by using the if elif else keyword.

Syntax:

if (condition):

statement(s)

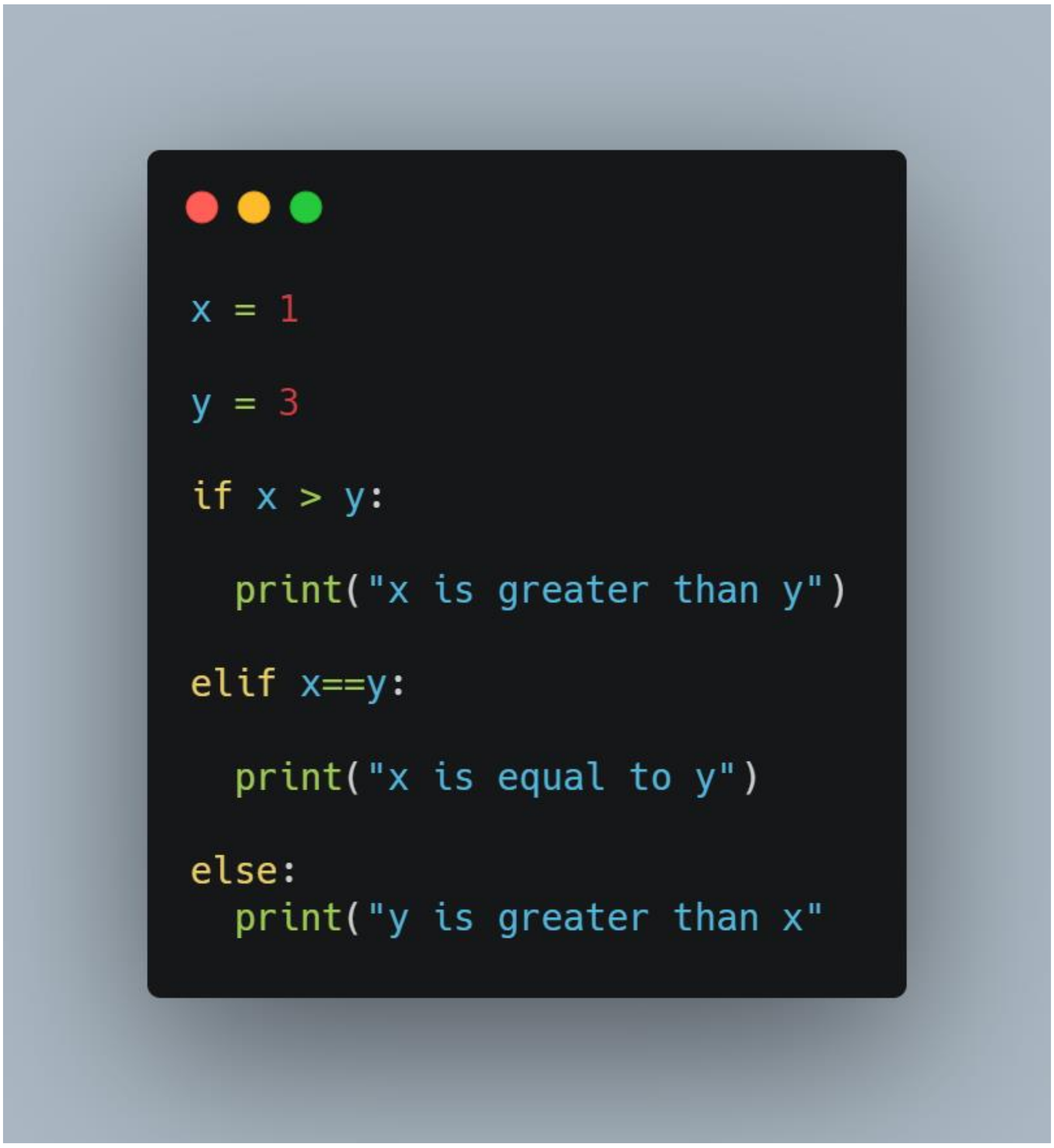
elif (condition):

statement(s)

else:

statement(s)

Example:



```
x = 1

y = 3

if x > y:

  print("x is greater than y")

elif x==y:

  print("x is equal to y")

else:
  print("y is greater than x"
```

Lists

A list is a data structure in Python that is a mutable, ordered sequence of elements.

Mutable means that you can change the items inside, while ordered sequence is in reference to index location. The first element in a list will always be located at index 0.

Each element or value that is inside of a list is called an item. Just as strings are defined as characters between quotes, lists are defined by having different data types between

square brackets [ ]. Also, like strings, each item within a list is assigned an index,

or location, for where that item is saved in memory. Lists are also known as a data

collection. Data collections are simply data types that can store multiple items. We'll see

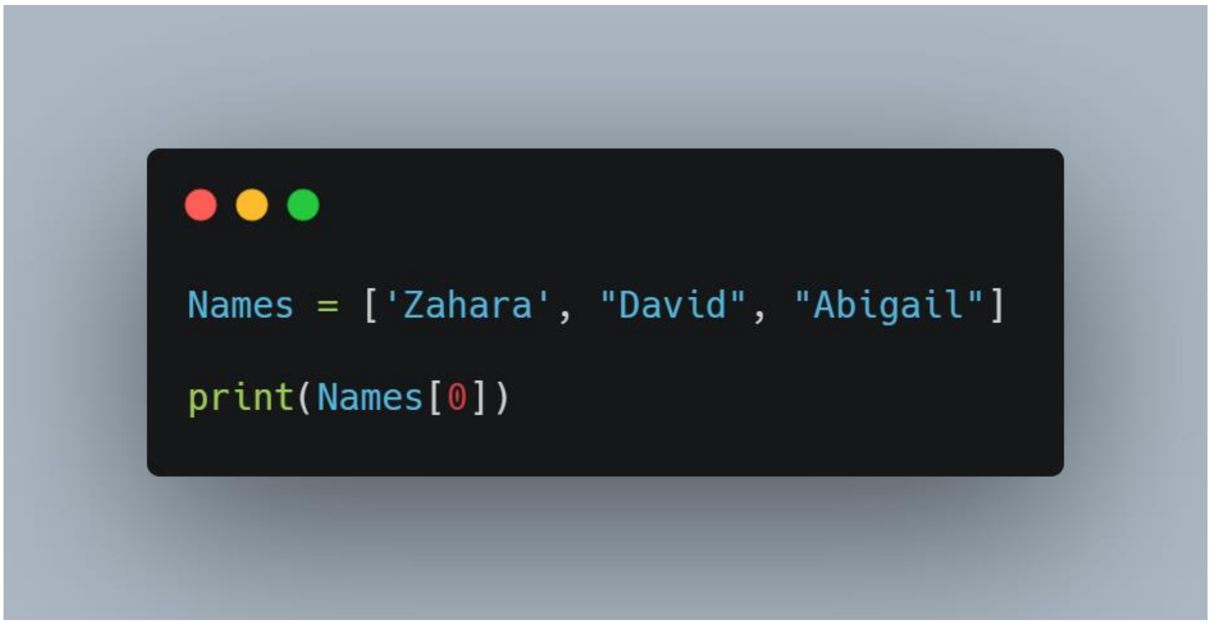
other data collections, like dictionaries and tuples, later.

Creating a list is just like creating a variable, here is an example,

Names = ['Zahara', "David", "Abigail"]

Accessing an item within a list

Just like we did with strings, we use a square bracket and an index. Here is an example,



```python
Names = ['Zahara', "David", "Abigail"]

print(Names[0])
```

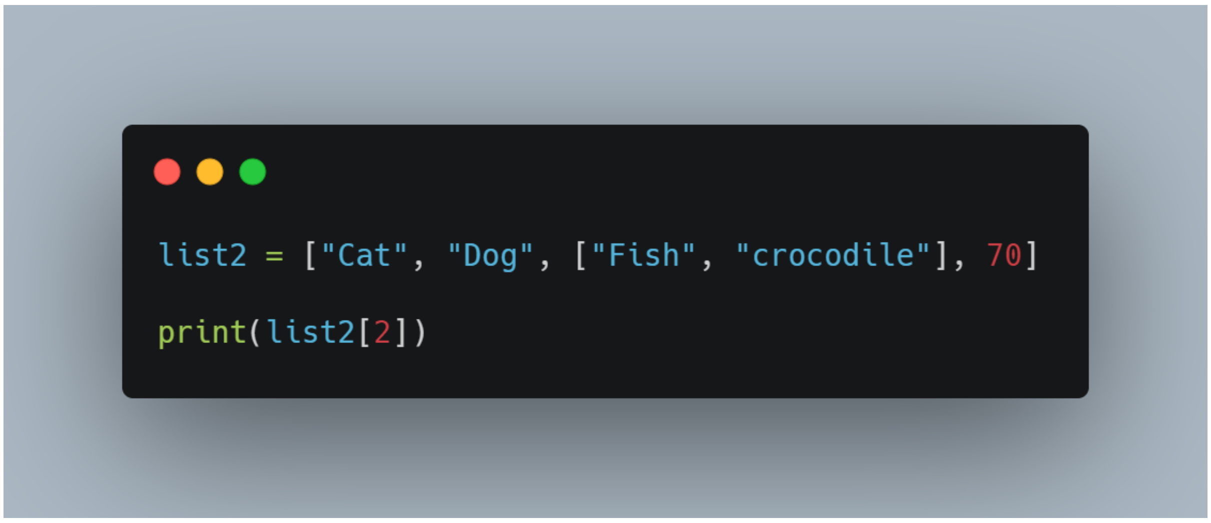
The above code returns the first item in the list.

You can also create a list with items of different data types. Lets see an example,

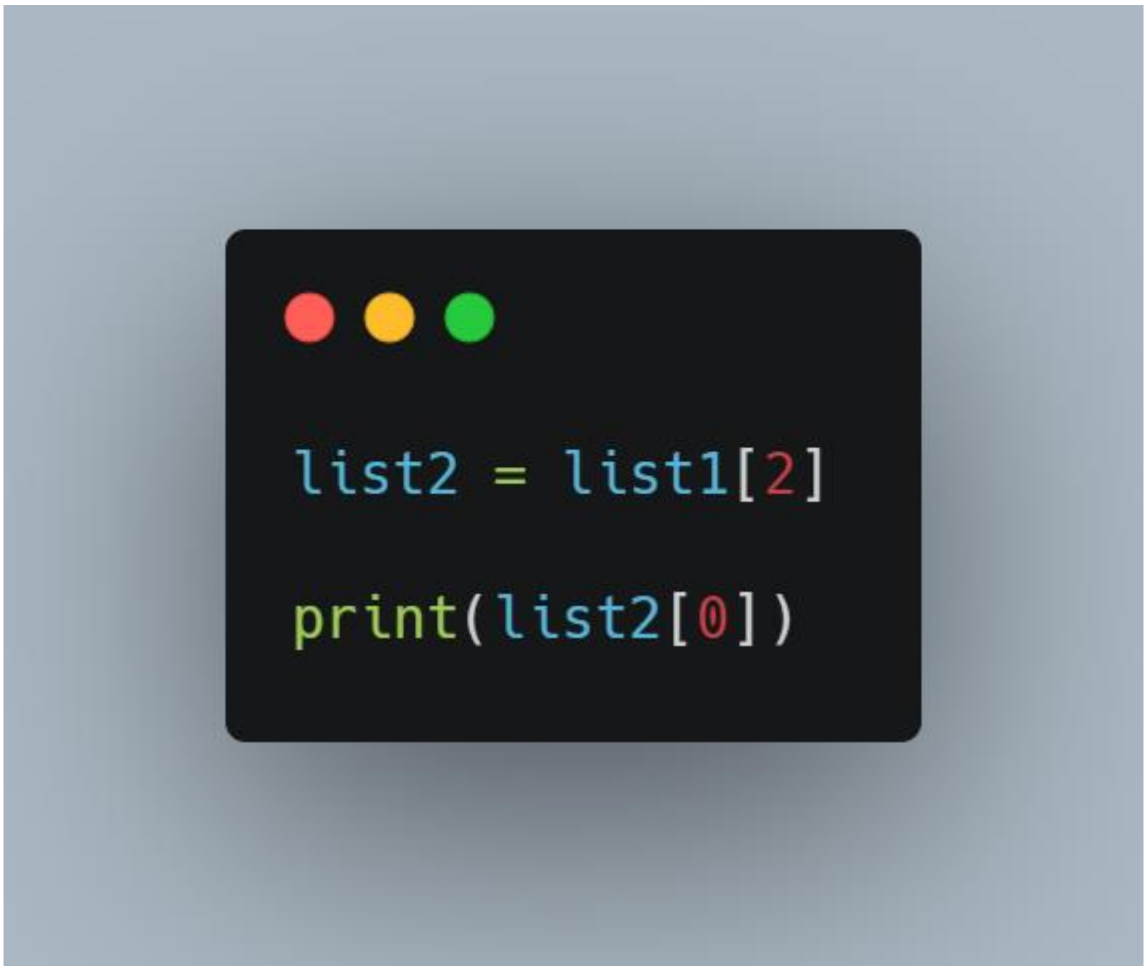
list1 = [2, 9.7, "word", True]

Lists within lists

Let's get a little more complex and see how lists can be stored within another list:


```
list2 = ["Cat", "Dog", ["Fish", "crocodile"], 70]

print(list2[2])
```

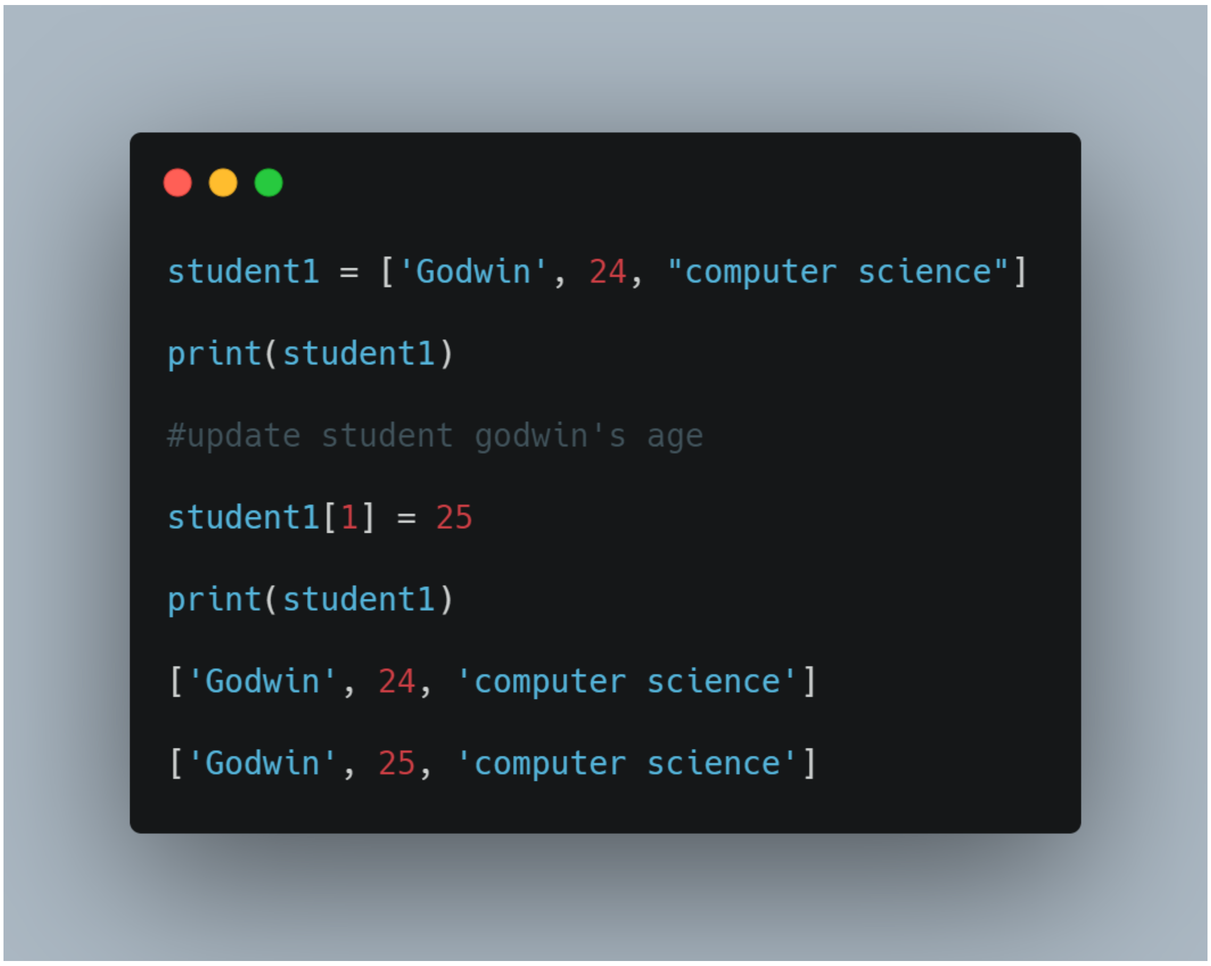
Note that you can access the element in the inner list using index, lets print “fish”,


```
list2 = list1[2]

print(list2[0])
```

Altering/ changing the values of items in a list.

At some point we might need to change or update a list. For example’ if we have a student record we might need to change the ages every year.

Here is an example;



```
student1 = ['Godwin', 24, "computer science"]

print(student1)

#update student godwin's age

student1[1] = 25

print(student1)

['Godwin', 24, 'computer science']

['Godwin', 25, 'computer science']
```