MY JOURNAL TO PYTHON

NAME: 喬萬斯

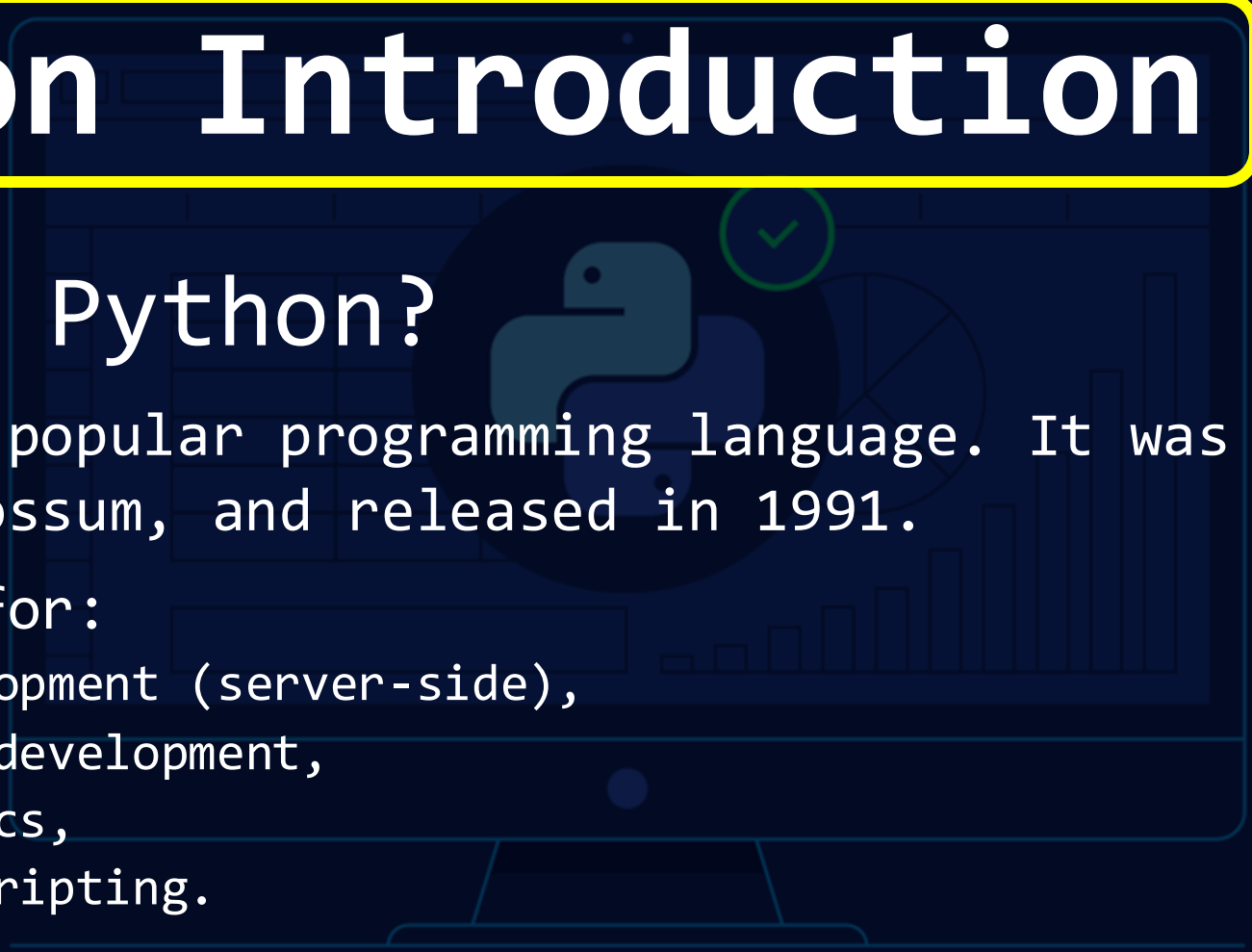
Teacher: My dear great teacher

4110E234

AGENDA

- Python Introduction
- Python Syntax
- Python Comments
- Python Variables
- Python Data Types
- Python Numbers
- Python Casting
- Python Strings
- Python Booleans
- Python Operators
- Python Lists
- Python Tuples
- Python Sets
- Python Dictionaries
- Python If ... Else
- Python While Loops
- Python For Loops
- Python Functions

Python Introduction



What is Python?

Python is a popular programming language. It was created by Guido van Rossum, and released in 1991.

It is used for:

- web development (server-side),
- software development,
- mathematics,
- system scripting.

Python Introduction

What can Python do?

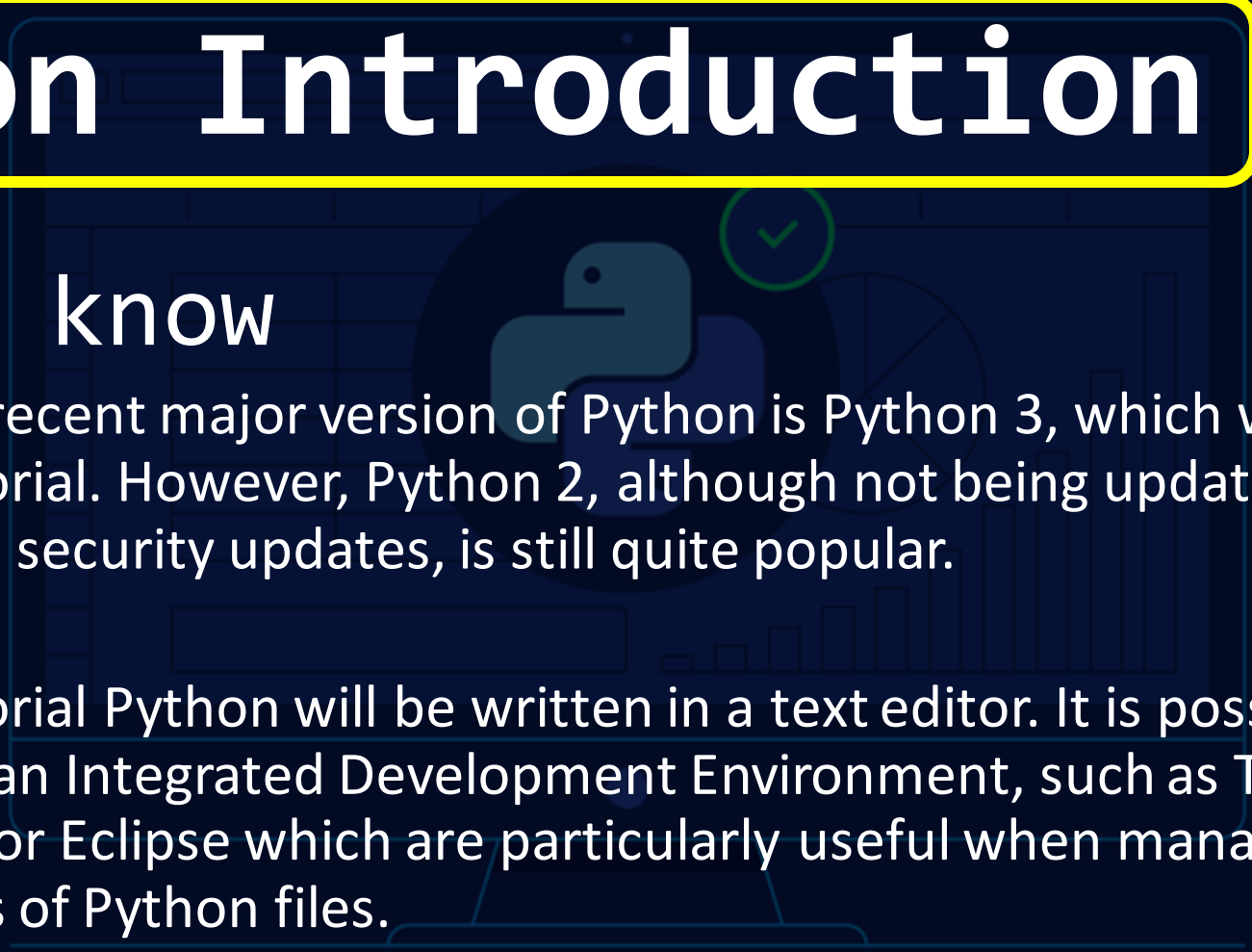
- Python can be used on a server to create web applications.
- Python can be used alongside software to create workflows.
- Python can connect to database systems. It can also read and modify files.
- Python can be used to handle big data and perform complex mathematics.
- Python can be used for rapid prototyping, or for production-ready software development.

Python Introduction

Why Python?

- Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc).
- Python has a simple syntax similar to the English language.
- Python has syntax that allows developers to write programs with fewer lines than some other programming languages.
- Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.
- Python can be treated in a procedural way, an object-oriented way or a functional way.

Python Introduction



Good to know

- The most recent major version of Python is Python 3, which we shall be using in this tutorial. However, Python 2, although not being updated with anything other than security updates, is still quite popular.

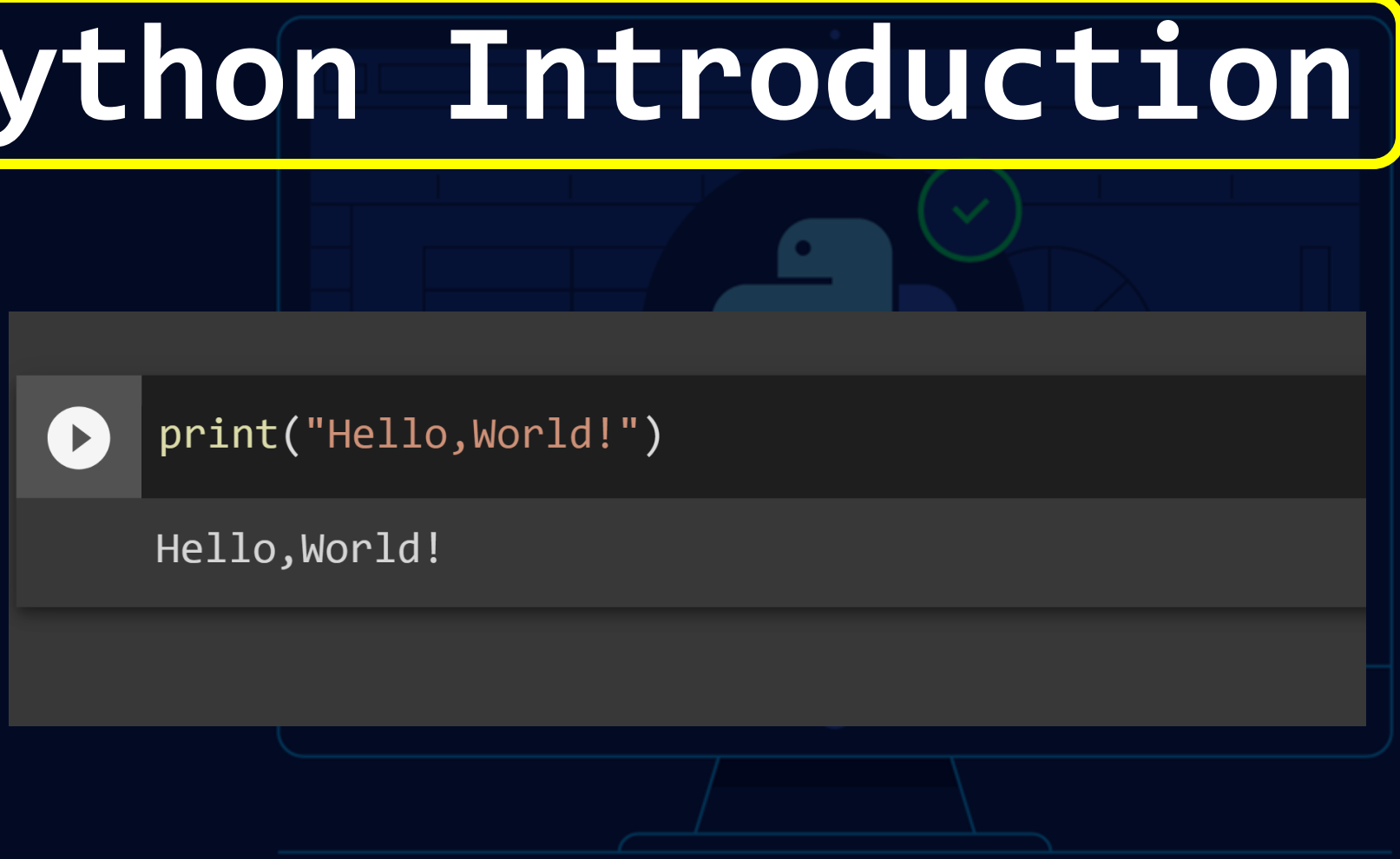
- In this tutorial Python will be written in a text editor. It is possible to write Python in an Integrated Development Environment, such as Thonny, Pycharm, Netbeans or Eclipse which are particularly useful when managing larger collections of Python files.

Python Introduction

Python Syntax compared to other programming languages

- Python was designed for readability, and has some similarities to the English language with influence from mathematics.
- Python uses new lines to complete a command, as opposed to other programming languages which often use semicolons or parentheses.
- Python relies on indentation, using whitespace, to define scope; such as the scope of loops, functions and classes. Other programming languages often use curly-brackets for this purpose.
-

Python Introduction



Python Syntax

Execute Python Syntax

As we learned in the previous page, Python syntax can be executed by writing directly in the Command Line:

```
>>> print("Hello, World!")
Hello, World!
```

Or by creating a python file on the server, using the .py file extension, and running it in the Command Line:

```
C:\Users\Your Name>python myfile.py
```

Python Indentation

Python Indentation

Indentation refers to the spaces at the beginning of a code line.

Where in other programming languages the indentation in code is for readability only, the indentation in Python is very important.

Python uses indentation to indicate a block of code.

Example

```
if 9 > 3:
  print("Nine is greater than three!")
```

```
Nine is greater than three!
```

Python Indentation

- Python will give you an error if you skip the indentation:

Example

Syntax Error:

```
if 9 > 3:
print("Nine is greater than three!")
```

```
  File "<ipython-input-5-07668de806fb>", line 2
    print("Nine is greater than three!")
        ^
IndentationError: expected an indented block
```

SEARCH STACK OVERFLOW

Python Indentation

- The number of spaces is up to you as a programmer, the most common use is four, but it has to be at least one.

Example

```
if 9 > 3:
 print("Nine is greater than three!")
if 9 > 3:
        print("Nine is greater than three!")
```

```
Nine is greater than three!
Nine is greater than three!
```

Python Indentation

- You have to use the same number of spaces in the same block of code, otherwise Python will give you an error:

Example

Syntax Error:

```
[7]  if 9 > 3:
      print("Nine is greater than three!")
            print("Nine is greater than three!")
```

```
  File "<ipython-input-7-8188e35bdaa6>", line 3
    print("Nine is greater than three!")
    ^
IndentationError: unexpected indent
```

SEARCH STACK OVERFLOW

Python Comments

Python Comments

Creating a Comment

Comments starts with a `#`, and Python will ignore them:

Example

```
#This is a comment.
print("Hello, World!")
```

```
Hello, World!
```

Python Comments

- Comments can be placed at the end of a line, and Python will ignore the rest of the line:

Example

```
print("Hello, World!") #This is a comment.
```

```
Hello, World!
```

Python Comments

- A comment does not have to be text that explains the code, it can also be used to prevent Python from executing code:

Example

```
#print("Hello, World!")
print("Go, Tigers!")
```

```
Go, Tigers!
```

Python Variables

Variables

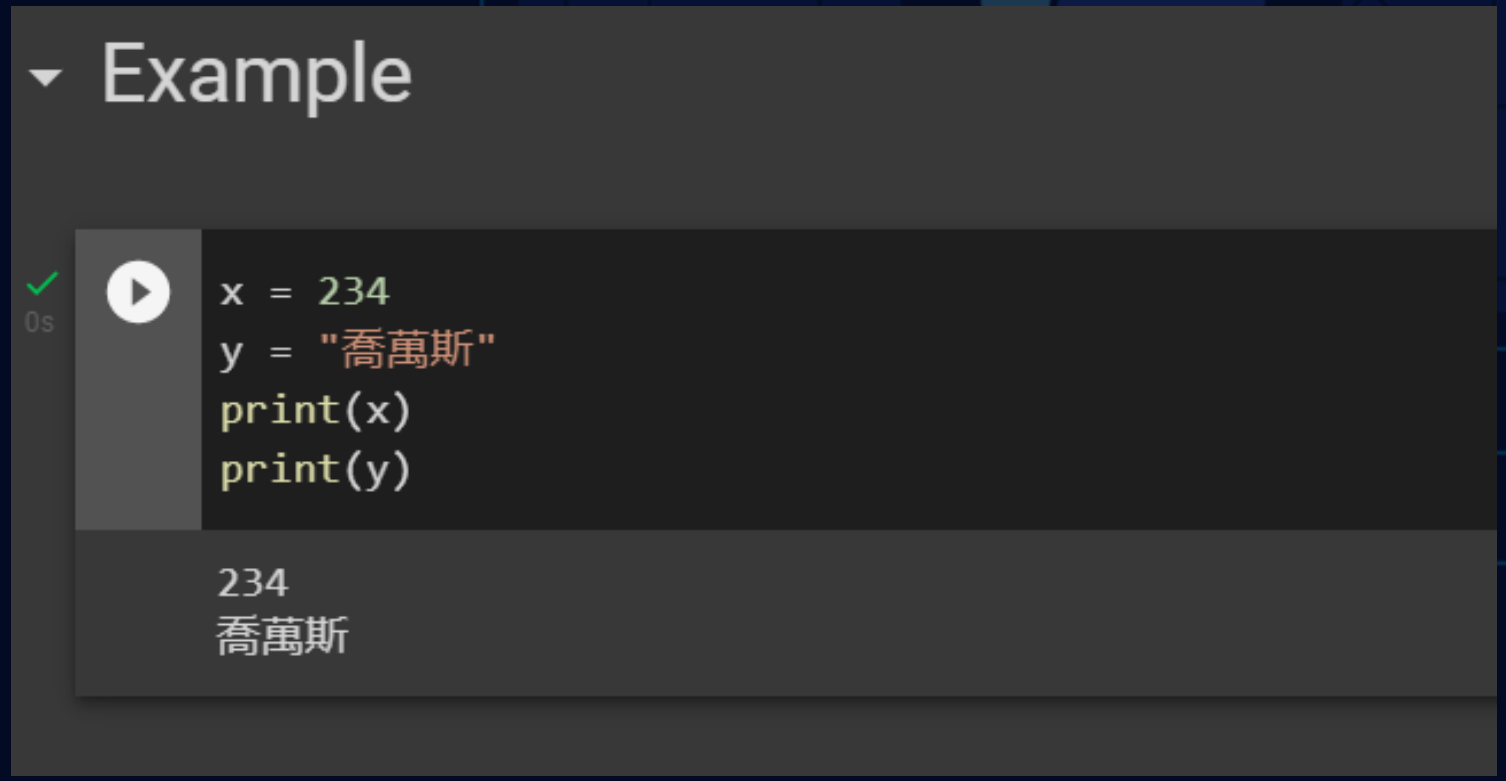
- Variables are containers for storing data values.

Creating Variables

- Python has no command for declaring a variable.
- A variable is created the moment you first assign a value to it.

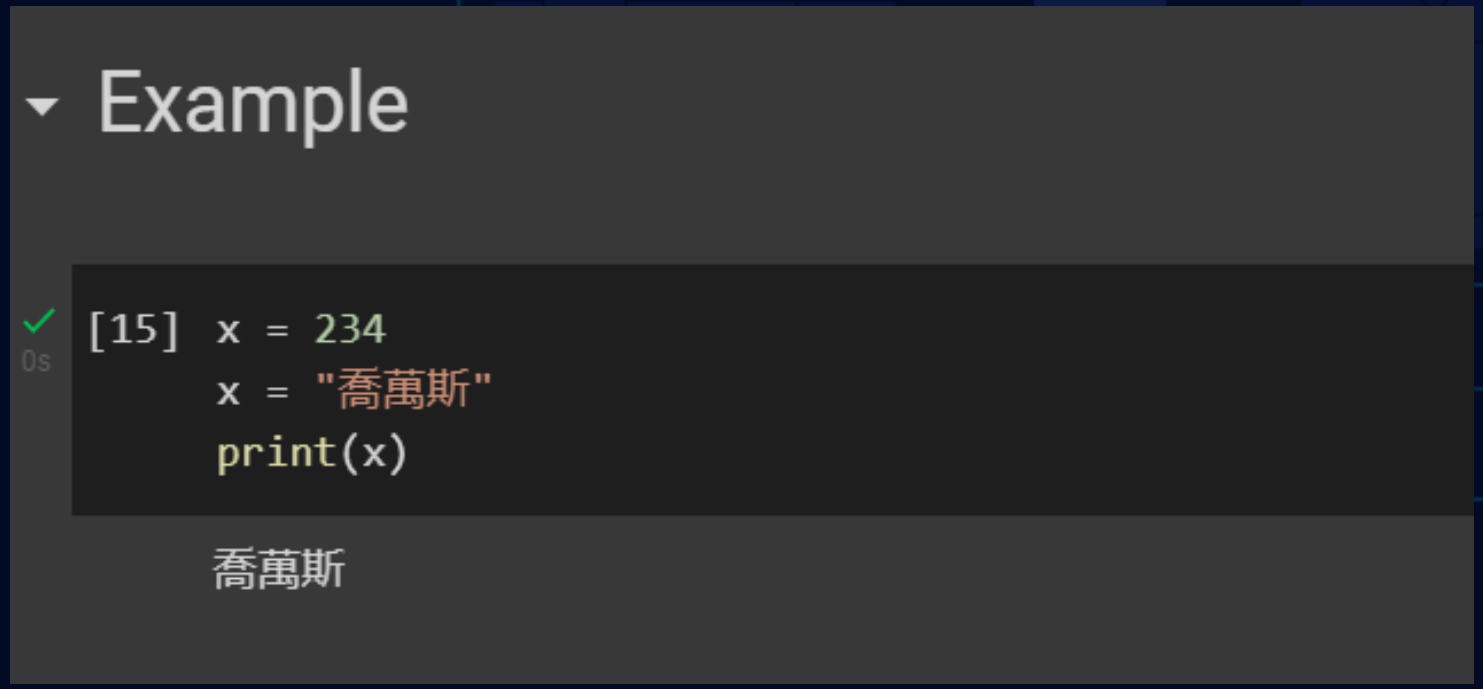
Python Variables

Creating Variables



Python Variables

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



Python Variables

Casting

- If you want to specify the data type of a variable, this can be done with casting.

Example

```
x = str(3)
y = int(6)
z = float(9)

print(x)
print(y)
print(z)
```

```
3
6
9.0
```

Python - Variable Names

Variable Names

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

- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- Variable names are case-sensitive (age, Age and AGE are three different variables)

Python - Variable Names

Example

Legal variable names:

```
myvar = "喬萬斯"
my_var = "喬萬斯"
_my_var = "喬萬斯"
myVar = "喬萬斯"
MYVAR = "喬萬斯"
myvar2 = "喬萬斯"


print(myvar)
print(my_var)
print(_my_var)
print(myVar)
print(MYVAR)
print(myvar2)
```

```
喬萬斯
喬萬斯
喬萬斯
喬萬斯
喬萬斯
喬萬斯
```

Python - Variable Names

Example

Illegal variable names:

```
2myvar = "喬萬斯"
my-var = "喬萬斯"
my var = "喬萬斯"
```

```
  File "<ipython-input-20-0714fdf66782>", line 1
    2myvar = "喬萬斯"
         ^
SyntaxError: invalid syntax
```

SEARCH STACK OVERFLOW

Remember that variable names are case-sensitive

Python - Variable Names

Multi Words Variable Names

Variable names with more than one word can be difficult to read.

There are several techniques you can use to make them more readable:

Camel Case

Each word, except the first, starts with a capital letter:

```
myVariableName = "John"
```

Pascal Case

Each word starts with a capital letter:

```
MyVariableName = "John"
```

Snake Case

Each word is separated by an underscore character:

```
my_variable_name = "John"
```

Python Variables - Assign Multiple Values

Many Values to Multiple Variables

Python allows you to assign values to multiple variables in one line:

Example

```
x, y, z = "Anger", "Chaos", "Rage"

print(x)
print(y)
print(z)
```

```
Anger
Chaos
Rage
```

Note: Make sure the number of variables matches the number of values, or else you will get an error.

Python Variables - Assign Multiple Values

One Value to Multiple Variables

And you can assign the *same* value to multiple variables in one line:

Example

```
x = y = z = "Disaster"

print(x)
print(y)
print(z)
```

```
Disaster
Disaster
Disaster
```

Python Variables - Assign Multiple Values

Unpack a Collection

If you have a collection of values in a list, tuple etc. Python allows you to extract the values into variables. This is called *unpacking*.

Example

Unpack a list:

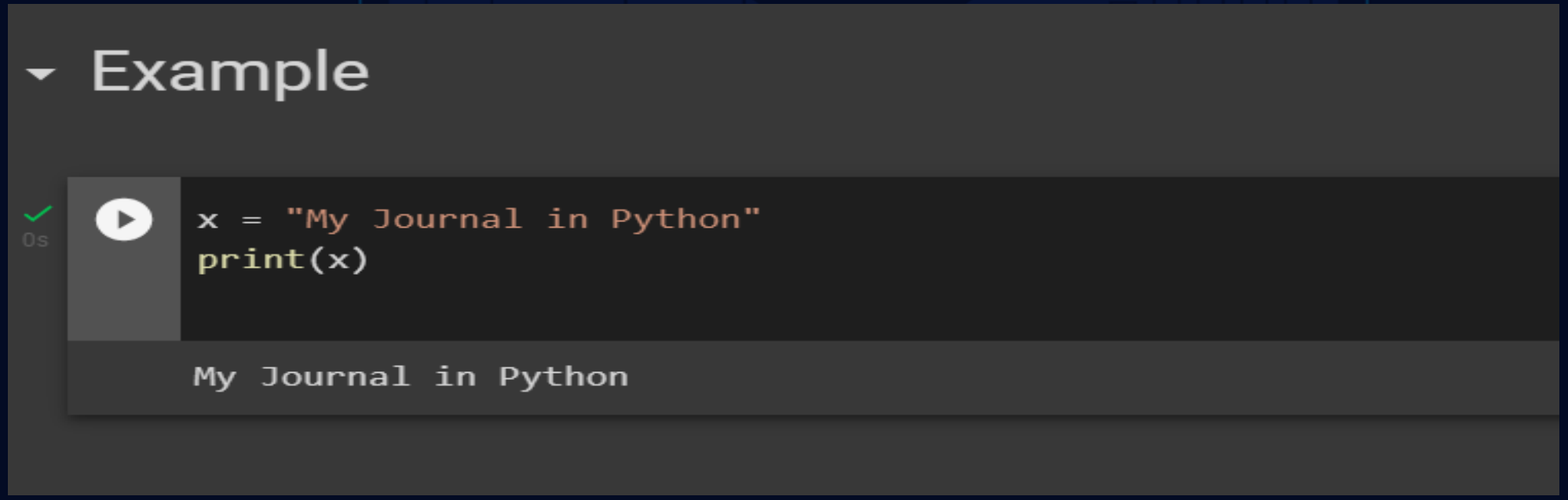
```
emotion = ["ANGER", "FEAR", "SADNESS"]
x, y, z = emotion

print(x)
print(y)
print(z)
```

```
ANGER
FEAR
SADNESS
```

Python - Output Variables

Output Variables

The Python print() function is often used to output variables.



Python - Output Variables

In the `print()` function, you output multiple variables, separated by a comma:

Example

```
x = "My Journal"
y = "in"
z = "Python"
print(x, y, z)
```

```
My Journal in Python
```

Python - Output Variables

You can also use the + operator to output multiple variables:

Example

```
x = "My Journal "
y = "in "
z = "Python"
print(x + y + z)
```

```
My Journal in Python
```

Notice the space character after `"My Journal "` and `"in "`, without them the result would be "MyJournalinPython".

Python - Output Variables

For numbers, the + character works as a mathematical operator:

.

Example

```
x = 3
y = 69
print(x + y)
```

```
72
```

Python - Output Variables

In the print() function, when you try to combine a string and a number with the + operator, Python will give you an error:

Example

```
x = 234
y = "喬萬斯"
print(x + y)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-2-57ad1ea9315b> in <module>
      1 x = 234
      2 y = "喬萬斯"
----> 3 print(x + y)

TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

SEARCH STACK OVERFLOW

Python - Output Variables

The best way to output multiple variables in the print() function is to separate them with commas, which even support different data types:

Example

```
x = 234
y = "喬萬斯"
print(x, y)
```

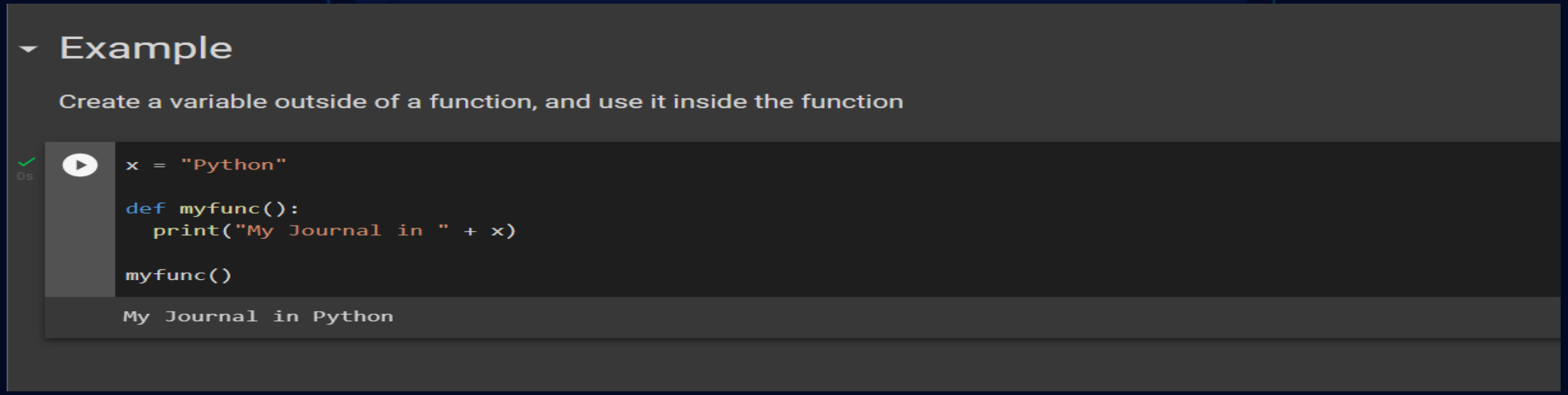
```
234 喬萬斯
```

Python - Global Variables

Global Variables

Variables that are created outside of a function (as in all of the examples above) are known as global variables.

Global variables can be used by everyone, both inside of functions and outside.



Python - Global Variables

If you create a variable with the same name inside a function, this variable will be local, and can only be used inside the function. The global variable with the same name will remain as it was, global and with the original value.

Example

Create a variable inside a function, with the same name as the global variable

```
x = "Python"

def myfunc():
  x = "Teacher"
  print("My Great " + x)

myfunc()

print("My Journal in " + x)
```

```
My Great Teacher
My Journal in Python
```

Python - Global Variables

The global Keyword

Normally, when you create a variable inside a function, that variable is local, and can only be used inside that function.

To create a global variable inside a function, you can use the `global` keyword.

Example

If you use the global keyword, the variable belongs to the global scope:

```
def myfunc():
  global x
  x = "Python"

myfunc()

print("My Journal in " + x)
```

```
My Journal in Python
```

Python - Global Variables

Also, use the global keyword if you want to change a global variable inside a function.

Example

To change the value of a global variable inside a function, refer to the variable by using the global keyword:

```
x = "Teacher"

def myfunc():
  global x
  x = "Python"

myfunc()

print("My Journal in " + x)
```

```
My Journal in Python
```

Python Data Types

Built-in Data Types

In programming, data type is an important concept.
Variables can store data of different types, and different types can do different things.
Python has the following data types built-in by default, in these categories:

Text Type:	`str`	Set Types:	`set, frozenset`
Numeric Types:	`int, float, complex`	Boolean Type:	`bool`
Sequence Types:	`list, tuple, range`	Binary Types:	`bytes, bytearray, memoryview`
Mapping Type:	`dict`	None Type:	`NoneType`

Python Data Types

Getting the Data Type

You can get the data type of any object by using the `type()` function:

Example

Print the data type of the variable x:

```
x = 5
print(type(x))
```

```
<class 'int'>
```

Python Data Types

Setting the Data Type

In Python, the data type is set when you assign a value to a variable:

Example	Data Type	Try it
x = "Hello World"	str	Try it »
x = 20	int	Try it »
x = 20.5	float	Try it »
x = 1j	complex	Try it »
x = ["apple", "banana", "cherry"]	list	Try it »
x = ("apple", "banana", "cherry")	tuple	

Python Data Types

Setting the Data Type

In Python, the data type is set when you assign a value to a variable:

x = range(6)	range	Try it »
x = {"name" : "John", "age" : 36}	dict	Try it »
x = {"apple", "banana", "cherry"}	set	Try it »
x = frozenset({"apple", "banana", "cherry"})	frozenset	Try it »
x = True	bool	

Python Data Types

Setting the Data Type

In Python, the data type is set when you assign a value to a variable:

x = b"Hello"	bytes	Try it »
x = bytearray(5)	bytearray	Try it »
x = memoryview(bytes(5))	memoryview	Try it »
x = None	NoneType	Try it »

Python Data Types

Setting the Specific Data Type

If you want to specify the data type, you can use the following constructor functions:

Example	Data Type	Try it
x = str("Hello World")	str	Try it »
x = int(20)	int	Try it »
x = float(20.5)	float	Try it »
x = complex(1j)	complex	Try it »
x = list(("apple", "banana", "cherry"))	list	

Python Data Types

Setting the Specific Data Type

If you want to specify the data type, you can use the following constructor functions:

x = tuple(("apple", "banana", "cherry"))	**tuple**	**Try it »**
x = range(6)	range	Try it »
x = dict(name="John", age=36)	dict	Try it »
x = set(("apple", "banana", "cherry"))	set	Try it »
x = frozenset(("apple", "banana", "cherry"))	frozenset	Try it »

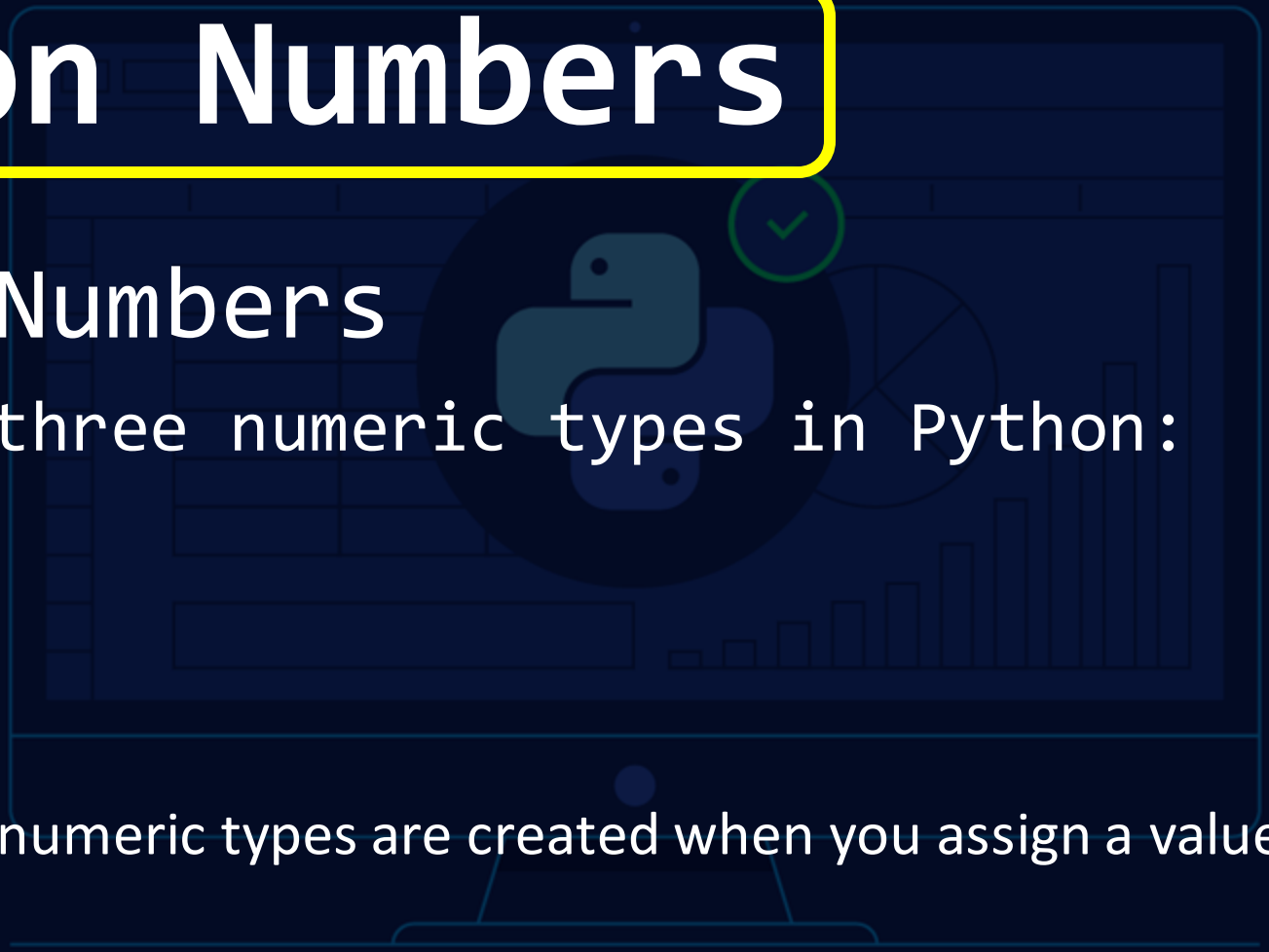
Python Data Types

Setting the Specific Data Type

If you want to specify the data type, you can use the following constructor functions:

x = bool(5)	bool	Try it »
x = bytes(5)	bytes	Try it »
x = bytearray(5)	bytearray	Try it »
x = memoryview(bytes(5))	memoryview	

Python Numbers



Python Numbers

There are three numeric types in Python:

- int
- float
- Complex

Variables of numeric types are created when you assign a value to them:

Python Numbers

Python Numbers

Example

```
x = 1    # int
y = 2.8  # float
z = 1j   # complex
```

Python Numbers

To verify the type of any object in Python, use the type() function:

Example

```
x = 1
y = 2.8
z = 1j

print(type(x))
print(type(y))
print(type(z))
```

```
<class 'int'>
<class 'float'>
<class 'complex'>
```

Python Numbers

Int

Int, or integer, is a whole number, positive or negative, without decimals, of unlimited length.

Example

Integers:

```
x = 6
y = 36912151821
z = -135792467

print(type(x))
print(type(y))
print(type(z))
```

```
<class 'int'>
<class 'int'>
<class 'int'>
```

Python Numbers

Float

Float, or "floating point number" is a number, positive or negative, containing one or more decimals.

Example

Floats:

```
x = 3.19
y = 6.13
z = -23.69

print(type(x))
print(type(y))
print(type(z))
```

```
<class 'float'>
<class 'float'>
<class 'float'>
```

Python Numbers

Float can also be scientific numbers with an "e" to indicate the power of 10.

Example

Floats:

```
x = 23e4
y = 46E7
z = -98.3e290

print(type(x))
print(type(y))
print(type(z))
```

```
<class 'float'>
<class 'float'>
<class 'float'>
```

Python Numbers

Complex

Complex numbers are written with a "j" as the imaginary part:

Example

Complex:

```
x = 2+7j
y = 9j
z = -3j

print(type(x))
print(type(y))
print(type(z))
```

```
<class 'complex'>
<class 'complex'>
<class 'complex'>
```

Python Numbers

Type Conversion

You can convert from one type to another with the int(), float(), and complex() methods:

Example

Convert from one type to another:

```
#convert from int to float:
x = float(1)

#convert from float to int:
y = int(2.8)

#convert from int to complex:
z = complex(1)

print(x)
print(y)
print(z)

print(type(x))
print(type(y))
print(type(z))
```

```
1.0
2
(1+0j)
<class 'float'>
<class 'int'>
<class 'complex'>
```

Note: You cannot convert complex numbers into another number type.

Python Numbers

Random Number

Python does not have a random() function to make a random number, but Python has a built-in module called random that can be used to make random numbers:

Example

Import the random module, and display a random number between 1 and 9:

```
import random

print(random.randrange(1, 10))
```

```
1
```

In our Random Module Reference you will learn more about the Random module.

Python Casting

Specify a Variable Type

There may be times when you want to specify a type on to a variable. This can be done with casting. Python is an object-orientated language, and as such it uses classes to define data types, including its primitive types.

Casting in python is therefore done using constructor functions:

- `int()` - constructs an integer number from an integer literal, a float literal (by removing all decimals), or a string literal (providing the string represents a whole number)
- `float()` - constructs a float number from an integer literal, a float literal or a string literal (providing the string represents a float or an integer)
- `str()` - constructs a string from a wide variety of data types, including strings, integer literals and float literals

Python Casting

Example

Integers:

```
x = int(3)
y = int(6.9)
z = int("9")
print(x)
print(y)
print(z)
```

```
3
6
9
```

Python Casting

Example

Floats:

```
x = float(3)
y = float(6.8)
z = float("9")
w = float("12.7")
print(x)
print(y)
print(z)
print(w)
```

```
3.0
6.8
9.0
12.7
```

Python Casting

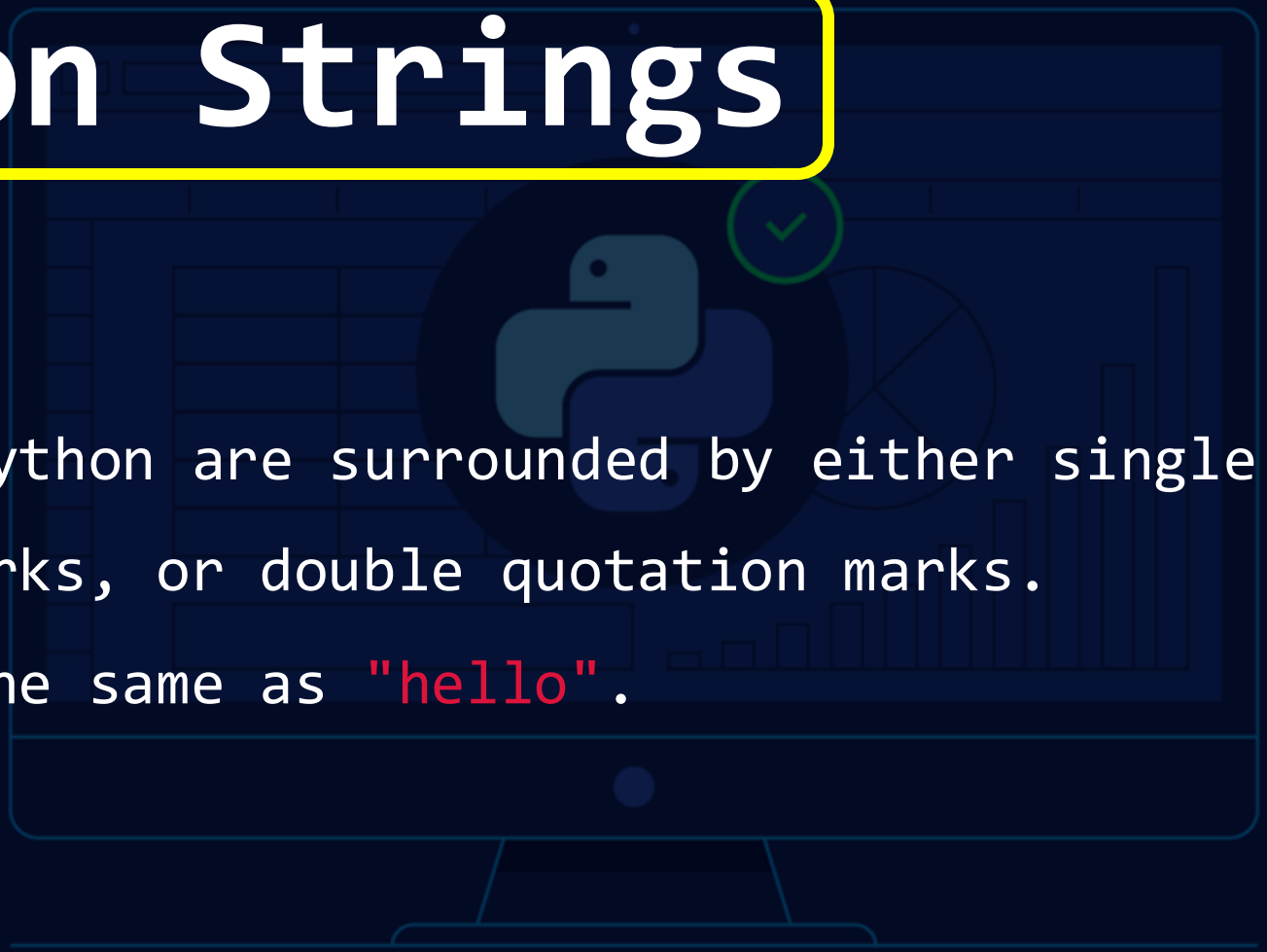
Example

Strings:

```
x = str("k3")
y = str(6)
z = str(9.0)
print(x)
print(y)
print(z)
```

```
k3
6
9.0
```

Python Strings



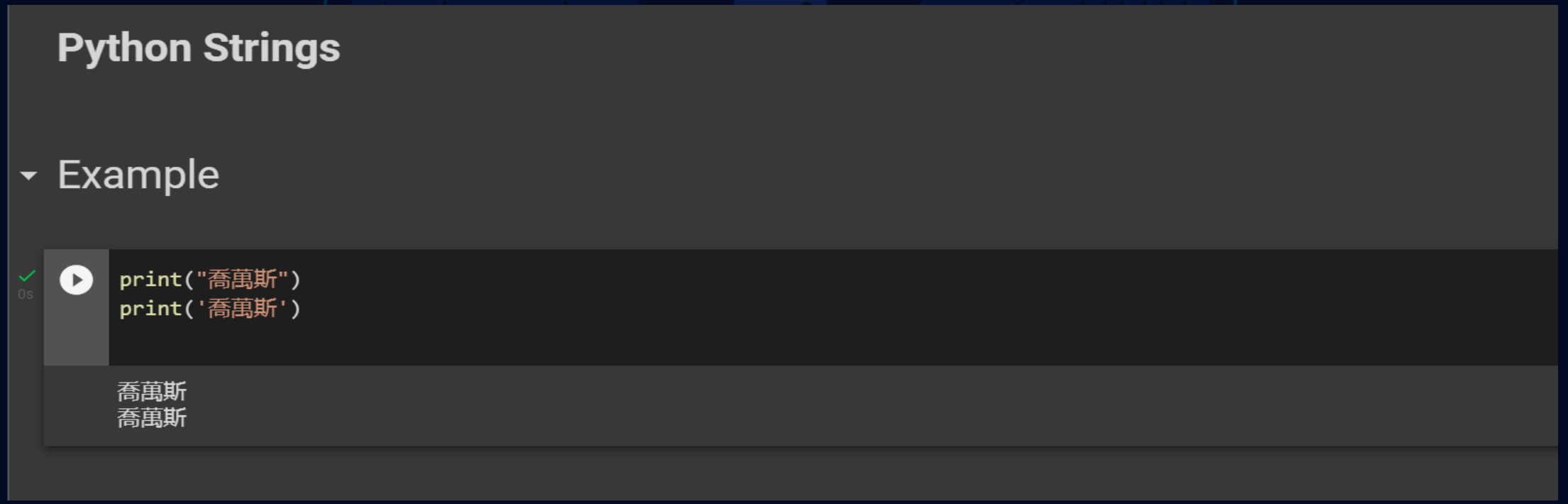
Strings

Strings in python are surrounded by either single quotation marks, or double quotation marks.

'hello' is the same as "hello".

Python Strings

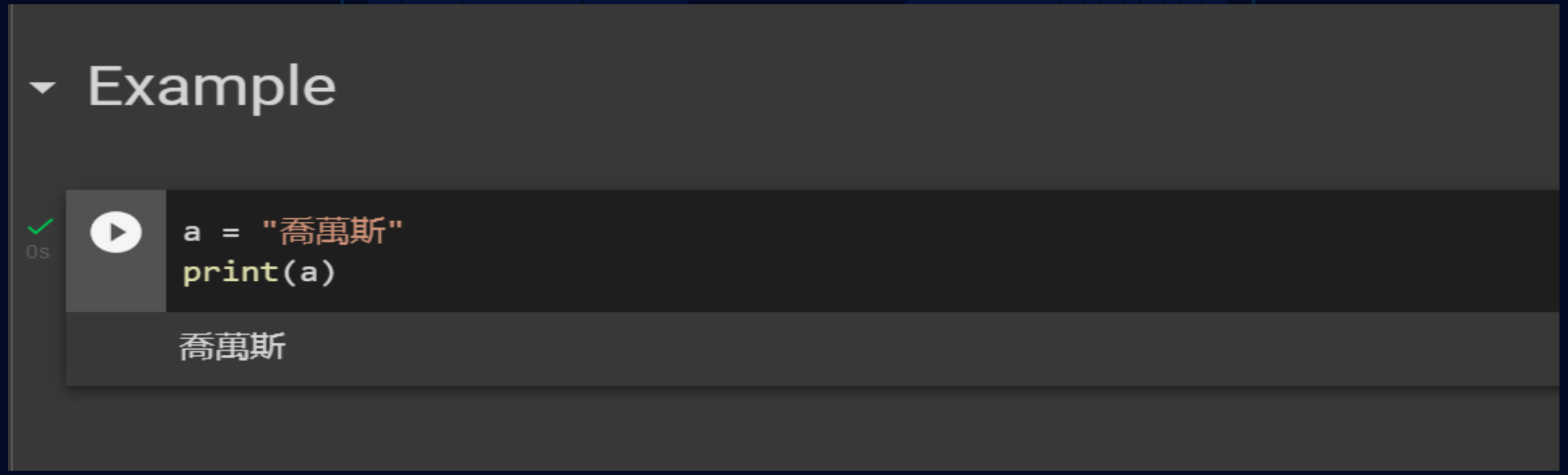
You can display a string literal with the print() function:



Python Strings

Assign String to a Variable

Assigning a string to a variable is done with the variable name followed by an equal sign and the string:



Python Strings

Multiline Strings

You can assign a multiline string to a variable by using three quotes:

Example

You can use three double quotes:

```
a = """Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua."""
print(a)
```

```
Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua.
```

Python Strings

Or three single quotes:

Example

```
a = '''Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua.'''
print(a)
```

```
Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua.
```

Note: in the result, the line breaks are inserted at the same position as in the code.

Python Strings

Strings are Arrays

Like many other popular programming languages, strings in Python are arrays of bytes representing unicode characters.

However, Python does not have a character data type, a single character is simply a string with a length of 1.

Square brackets can be used to access elements of the string.

Python Strings

Example

Get the character at position 1 (remember that the first character has the position 0):

```
a = "Hello, World!"
print(a[1])
```

```
e
```

Python Strings

Looping Through a String

Since strings are arrays, we can loop through the characters in a string, with a for loop.

Example

Loop through the letters in the word "喬萬斯":

```
for x in "喬萬斯":
  print(x)
```

```
喬
萬
斯
```

Python Strings

String Length

To get the length of a string, use the len() function.

Example

The len() function returns the length of a string:

```
a = "Hello, World!"
print(len(a))
```

```
13
```

Python Strings

Check String

To check if a certain phrase or character is present in a string, we can use the keyword in.

Example

Check if "free" is present in the following text:

```
txt = "My Great Teacher in Python!"
print("Python!" in txt)
```

```
True
```

Python Strings

Use it in an if statement:

Example

Print only if "free" is present:

```
txt = "My Great Teacher in Python!"
if "Python!" in txt:
  print("Yes, 'Python!' is present.")
```

```
Yes, 'Python!' is present.
```

Python Strings

Check if NOT

To check if a certain phrase or character is NOT present in a string, we can use the keyword not in.

Example

Check if "expensive" is NOT present in the following text:

```
txt = "My Great Teacher in Python!"
print("Good" not in txt)
```

```
True
```

Python Strings

Use it in an if statement:

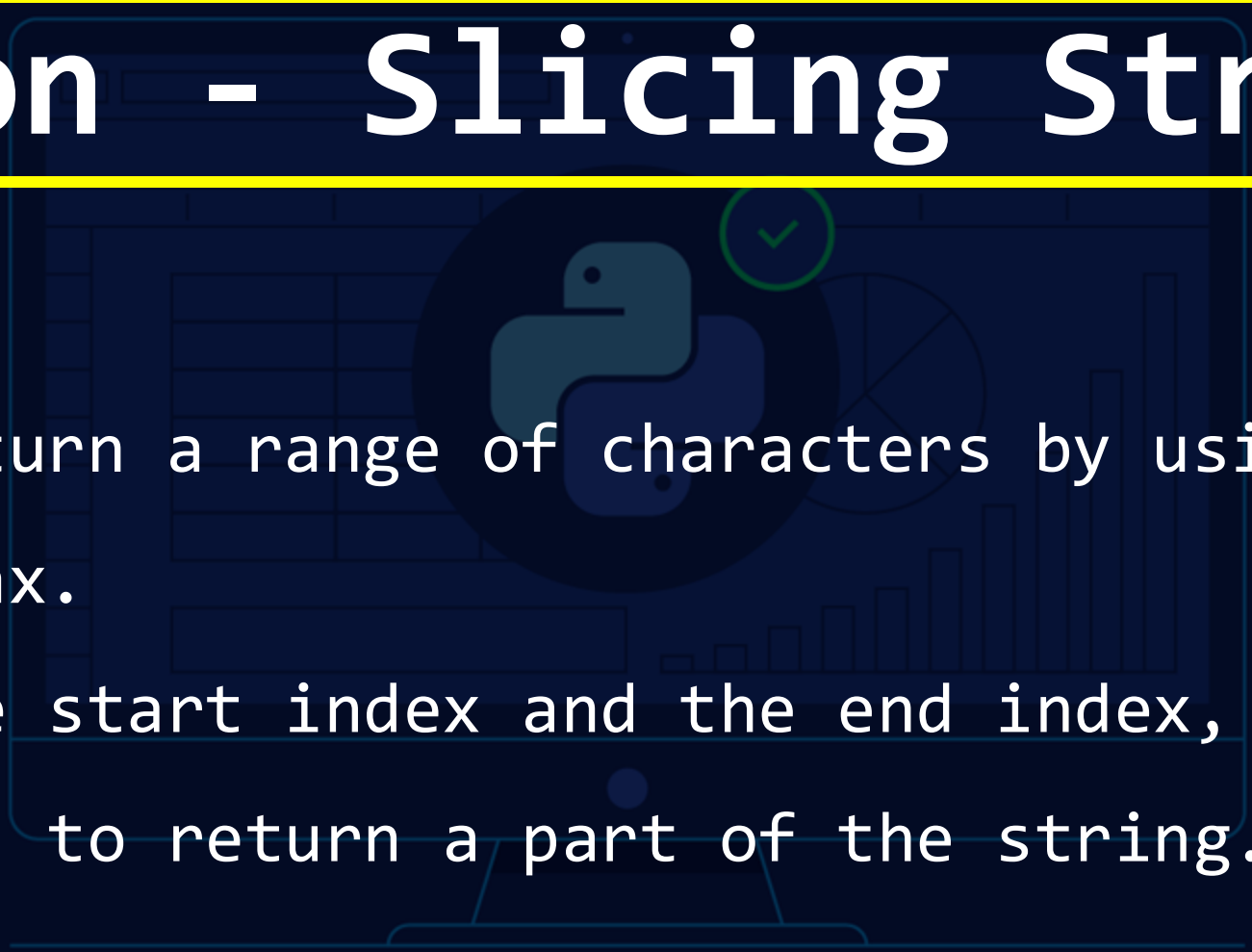
Example

print only if "Bad" is NOT present:

```
txt = "My Great Teacher in Python!"
if "Bad" not in txt:
  print("No, 'Bad' is NOT present.")
```

```
No, 'Bad' is NOT present.
```

Python - Slicing Strings



Slicing

You can return a range of characters by using the slice syntax.

Specify the start index and the end index, separated by a colon, to return a part of the string.

Python - Slicing Strings

Example

Get the characters from position 2 to position 5 (not included):

```
b = "My Great Teacher in Python!"
print(b[2:5])
```

```
Gr
```

Note: The first character has index 0.

Python - Slicing Strings

Slice From the Start

By leaving out the start index, the range will start at the first character:

Example

Get the characters from the start to position 16 (not included):

```
b = "My Great Teacher in Python!"
print(b[:16])
```

```
My Great Teacher
```

Python - Slicing Strings

Slice To the End

By leaving out the *end* index, the range will go to the end:

Example

Get the characters from position 2, and all the way to the end:

```
b = "My Great Teacher in Python!"
print(b[2:])
```

```
Great Teacher in Python!
```

Python - Slicing Strings

Negative Indexing

Use negative indexes to start the slice from the end of the string:

Example

Get the characters:

From: "o" in "World!" (position -5)

To, but not included: "d" in "World!" (position -2):

```
b = "My Great Teacher in Python!"
print(b[-5:-2])
```

```
tho
```

Python - Modify Strings

Upper Case

Example

The upper() method returns the string in upper case:

```
a = "My Great Teacher in Python!"
print(a.upper())
```

```
MY GREAT TEACHER IN PYTHON!
```

Python - Modify Strings

Lower Case

Example

The lower() method returns the string in lower case:

```
a = "My Great Teacher in Python!"
print(a.lower())
```

```
my great teacher in python!
```

Python - Modify Strings

Remove Whitespace

Whitespace is the space before and/or after the actual text, and very often you want to remove this space.

Example

The strip() method removes any whitespace from the beginning or the end:

```
a = " My Great Teacher in Python! "
print(a.strip())
```

```
My Great Teacher in Python!
```

Python - Modify Strings

Replace String

Example

The replace() method replaces a string with another string:

```
a = "My Great Teacher in Python!"
print(a.replace("M", "G"))
```

```
Gy Great Teacher in Python!
```

Python - Modify Strings

Split String

The split() method returns a list where the text between the specified separator becomes the list items.

Example

The split() method splits the string into substrings if it finds instances of the separator:

```
a = "My Great Teacher, in Python!"
b = a.split(",")
print(b)
```

```
['My Great Teacher', ' in Python!']
```

Python - String Concatenation

String Concatenation

To concatenate, or combine, two strings you can use the + operator.

Example

Merge variable a with variable b into variable c:

```
a = "4110E234"
b = "喬萬斯"
c = a + b
print(c)
```

```
4110E234喬萬斯
```

Python - String Concatenation

Example

To add a space between them, add a " ":

```
a = "4110E234"
b = "喬萬斯"
c = a + " " + b
print(c)
```

```
4110E234 喬萬斯
```

Python - Format - Strings

String Format

As we learned in the Python Variables chapter, we cannot combine strings and numbers like this:

Example

```
age = 18
txt = "My name is Joaquin, I am " + age
print(txt)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-54-358c28bb6507> in <module>
      1 age = 18
----> 2 txt = "My name is Joaquin, I am " + age
      3 print(txt)

TypeError: can only concatenate str (not "int") to str
```

SEARCH STACK OVERFLOW

Python - Format - Strings

But we can combine strings and numbers by using the format() method!

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

Example

Use the format() method to insert numbers into strings:

```
age = 18
txt = "My name is Joaquin, and I am {}"
print(txt.format(age))
```

```
My name is Joaquin, and I am 18
```

Python - Format - Strings

The format() method takes unlimited number of arguments, and are placed into the respective placeholders:

Example

```
quantity = 3
itemno = 567
price = 49.95
myorder = "I want {} pieces of item {} for {} dollars."
print(myorder.format(quantity, itemno, price))
```

```
I want 3 pieces of item 567 for 49.95 dollars.
```

Python - Format - Strings

You can use index numbers {0} to be sure the arguments are placed in the correct placeholders:

Example

```
quantity = 3
itemno = 567
price = 49.95
myorder = "I want to pay {2} dollars for {0} pieces of item {1}."
print(myorder.format(quantity, itemno, price))
```

```
I want to pay 49.95 dollars for 3 pieces of item 567.
```

Python - Escape Characters

Escape Character

To insert characters that are illegal in a string, use an escape character.

An escape character is a backslash \ followed by the character you want to insert.

An example of an illegal character is a double quote inside a string that is surrounded by double quotes:

Python - Escape Characters

Example

You will get an error if you use double quotes inside a string that is surrounded by double quotes:

```
txt = "We are the so-called "Filipino" from the Philippines."
```

```
  File "<ipython-input-59-4bc614800db7>", line 1
    txt = "We are the so-called "Filipino" from the Philippines."
                                ^
SyntaxError: invalid syntax
```

SEARCH STACK OVERFLOW

Python - Escape Characters

To fix this problem, use the escape character \":

Example

The escape character allows you to use double quotes when you normally would not be allowed:

```
txt = "We are the so-called \"Filipino\" from Philippines."
print(txt)
```

```
We are the so-called "Filipino" from Philippines.
```

Python - Escape Characters

Escape Characters

Other escape characters used in Python:

Code	Result	Try it
\'	Single Quote	Try it »
\\	Backslash	Try it »
\n	New Line	Try it »
\r	Carriage Return	Try it »

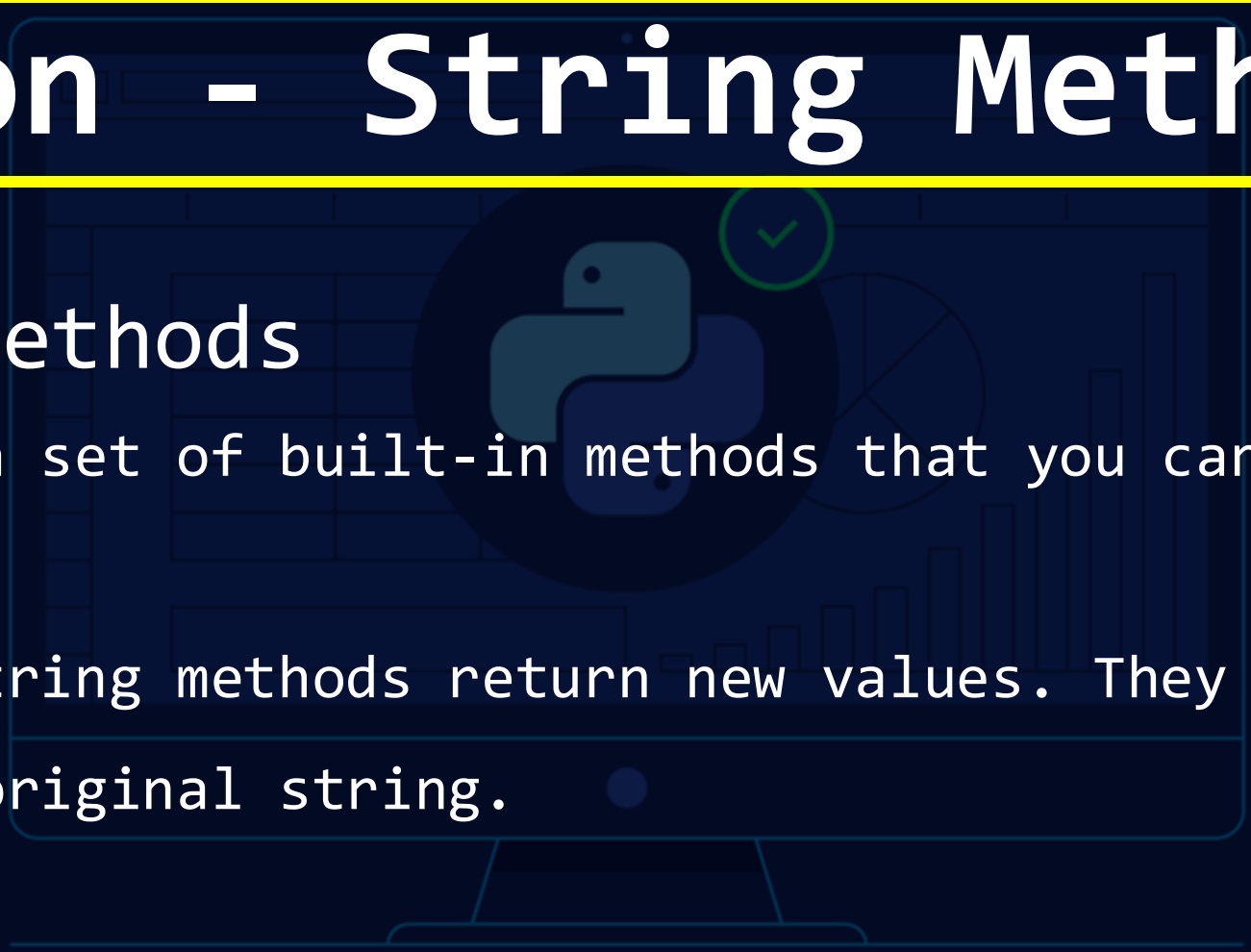
Python - Escape Characters

Escape Characters

Other escape characters used in Python:

\b	Backspace	Try it »
\f	Form Feed	
\ooo	Octal value	Try it »
\xhh	Hex value	Try it »

Python - String Methods



String Methods

Python has a set of built-in methods that you can use on strings.

Note: All string methods return new values. They do not change the original string.

Python - String Methods

Method	Description
capitalize()	Converts the first character to upper case
casefold()	Converts string into lower case
center()	Returns a centered string
count()	Returns the number of times a specified value occurs in a string
encode()	Returns an encoded version of the string
endswith()	Returns true if the string ends with the specified value
expandtabs()	Sets the tab size of the string
find()	Searches the string for a specified value and returns the position of where it was found
format()	Formats specified values in a string
format_map()	Formats specified values in a string

Python - String Methods

index()	Searches the string for a specified value and returns the position of where it was found
isalnum()	Returns True if all characters in the string are alphanumeric
isalpha()	Returns True if all characters in the string are in the alphabet
isdecimal()	Returns True if all characters in the string are decimals
isdigit()	Returns True if all characters in the string are digits
isidentifier()	Returns True if the string is an identifier
islower()	Returns True if all characters in the string are lower case
isnumeric()	Returns True if all characters in the string are numeric
isprintable()	Returns True if all characters in the string are printable
isspace()	Returns True if all characters in the string are whitespaces
istitle()	Returns True if the string follows the rules of a title

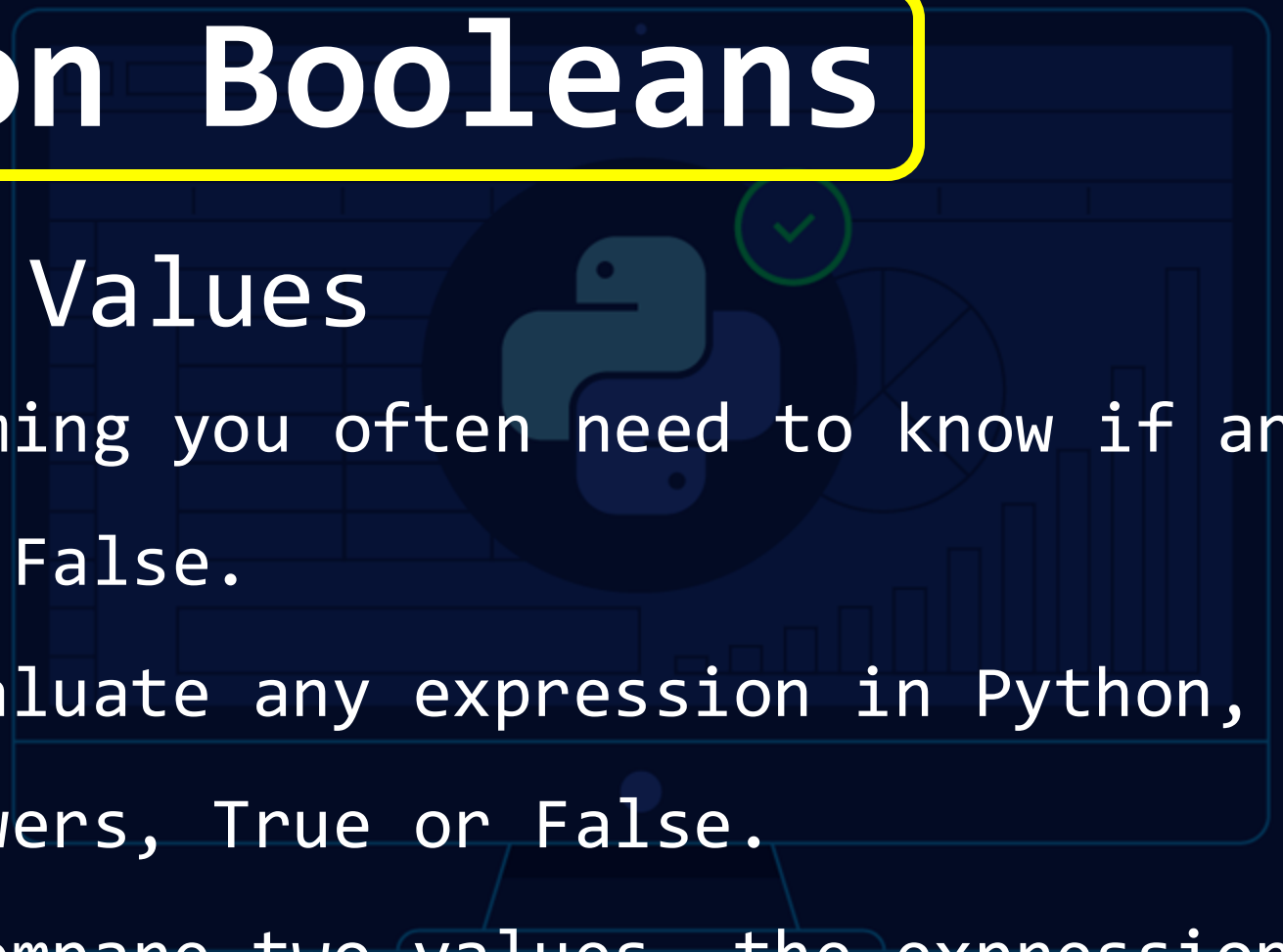
Python - String Methods

isupper()	Returns True if all characters in the string are upper case
join()	Joins the elements of an iterable to the end of the string
ljust()	Returns a left justified version of the string
lower()	Converts a string into lower case
lstrip()	Returns a left trim version of the string
maketrans()	Returns a translation table to be used in translations
partition()	Returns a tuple where the string is parted into three parts
replace()	Returns a string where a specified value is replaced with a specified value
rfind()	Searches the string for a specified value and returns the last position of where it was found
rindex()	Searches the string for a specified value and returns the last position of where it was found
rjust()	Returns a right justified version of the string
rpartition()	Returns a tuple where the string is parted into three parts
rsplit()	Splits the string at the specified separator, and returns a list

Python - String Methods

split()	Splits the string at the specified separator, and returns a list
splitlines()	Splits the string at line breaks and returns a list
startswith()	Returns true if the string starts with the specified value
strip()	Returns a trimmed version of the string
swapcase()	Swaps cases, lower case becomes upper case and vice versa
title()	Converts the first character of each word to upper case
translate()	Returns a translated string
upper()	Converts a string into upper case
zfill()	Fills the string with a specified number of 0 values at the beginning

Python Booleans



Boolean Values

In programming you often need to know if an expression is True or False.

You can evaluate any expression in Python, and get one of two answers, True or False.

When you compare two values, the expression is evaluated and Python returns the Boolean answer:

Python Booleans

Example

```
print(10 > 9)
print(10 == 9)
print(10 < 9)
```

```
True
False
False
```

Python Booleans

When you run a condition in an if statement, Python returns True or False:

Example

Print a message based on whether the condition is True or False:

```
a = 793
b = 68

if b > a:
  print("b is greater than a")
else:
  print("b is not greater than a")
```

```
b is not greater than a
```

Python Booleans

Evaluate Values and Variables

The bool() function allows you to evaluate any value, and give you True or False in return,

Example

Evaluate a string and a number:

```
print(bool("喬萬斯"))
print(bool(234))
```

```
True
True
```

Python Booleans

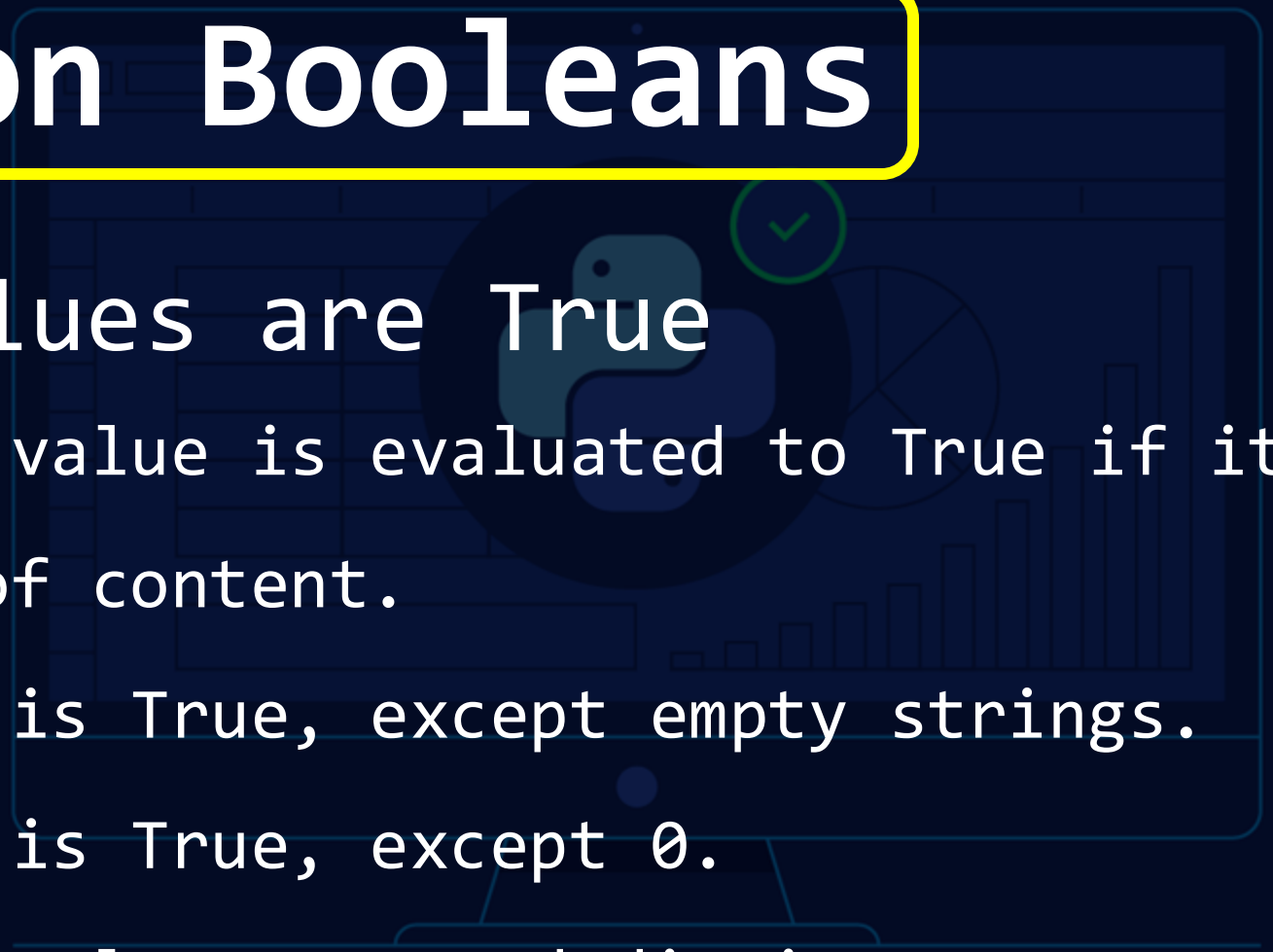
Example

Evaluate two variables:

```
x = "喬萬斯"
y = 234

print(bool(x))
print(bool(y))
```

```
True
True
```

Python Booleans



Most Values are True

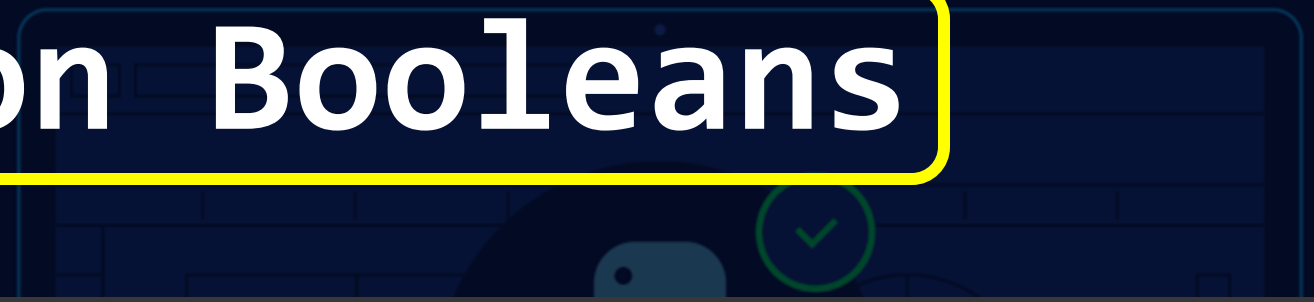
Almost any value is evaluated to True if it has some sort of content.

Any string is True, except empty strings.

Any number is True, except 0.

Any list, tuple, set, and dictionary are True, except empty ones.

Python Booleans



Example

The following will return True:

```
print(bool("ZED"))
print(bool(369))
print(bool(["anger", "fear", "sadness"]))
```

```
True
True
True
```

Python Booleans

Some Values are False

In fact, there are not many values that evaluate to False, except empty values, such as (), [], {}, "", the number 0, and the value None.

And of course the value False evaluates to False.

Python Booleans

Example

The following will return False:

```
print(bool(False))
print(bool(None))
print(bool(0))
print(bool(""))
print(bool(()))
print(bool([]))
print(bool({}))
```

```
False
False
False
False
False
False
False
```

Python Booleans

One more value, or object in this case, evaluates to False, and that is if you have an object that is made from a class with a __len__ function that returns 0 or False:

Example

```
class myclass():
  def __len__(self):
    return 0

myobj = myclass()
print(bool(myobj))
```

```
False
```

Python Booleans

Functions can Return a Boolean

You can create functions that returns a Boolean Value:

Example

Print the answer of a function:

```
def myFunction() :
  return True

print(myFunction())
```

```
True
```

Python Booleans

You can execute code based on the Boolean answer of a function:

Example

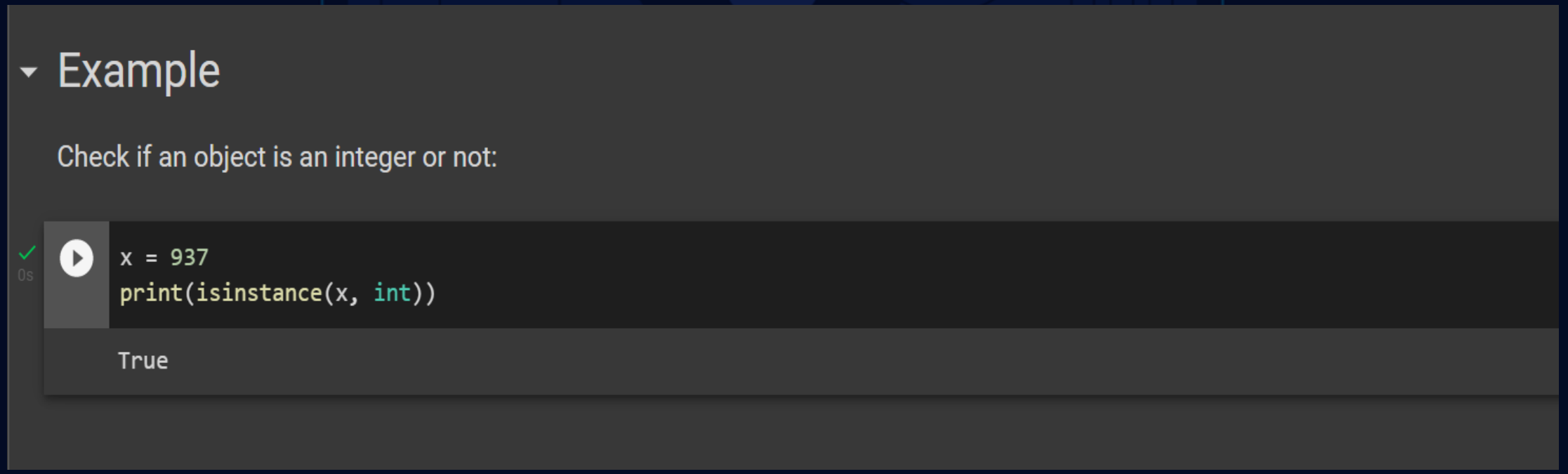
Print "YES!" if the function returns True, otherwise print "NO!":

```
def myFunction() :
  return True

if myFunction():
  print("YES!")
else:
  print("NO!")
```

```
YES!
```

Python Booleans

Python also has many built-in functions that return a boolean value, like the isinstance() function, which can be used to determine if an object is of a certain data type:

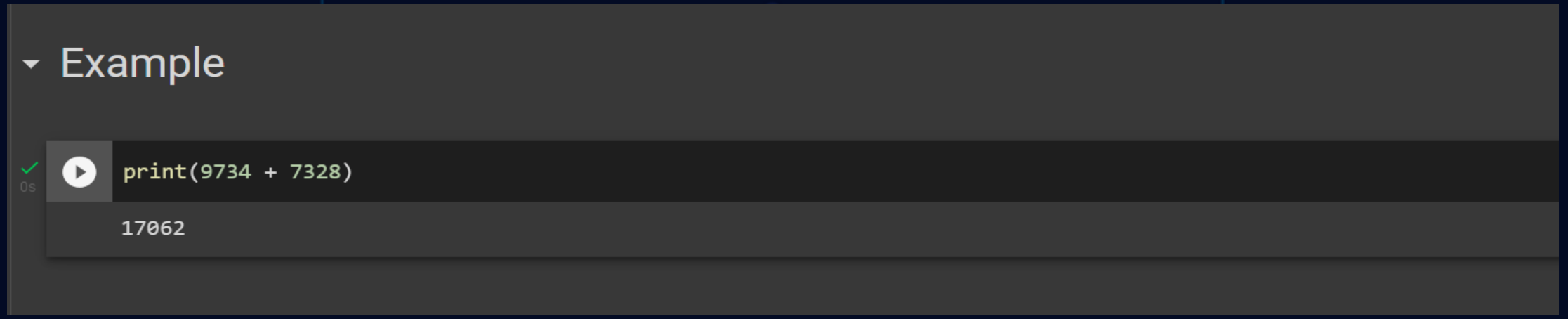


Python Operators

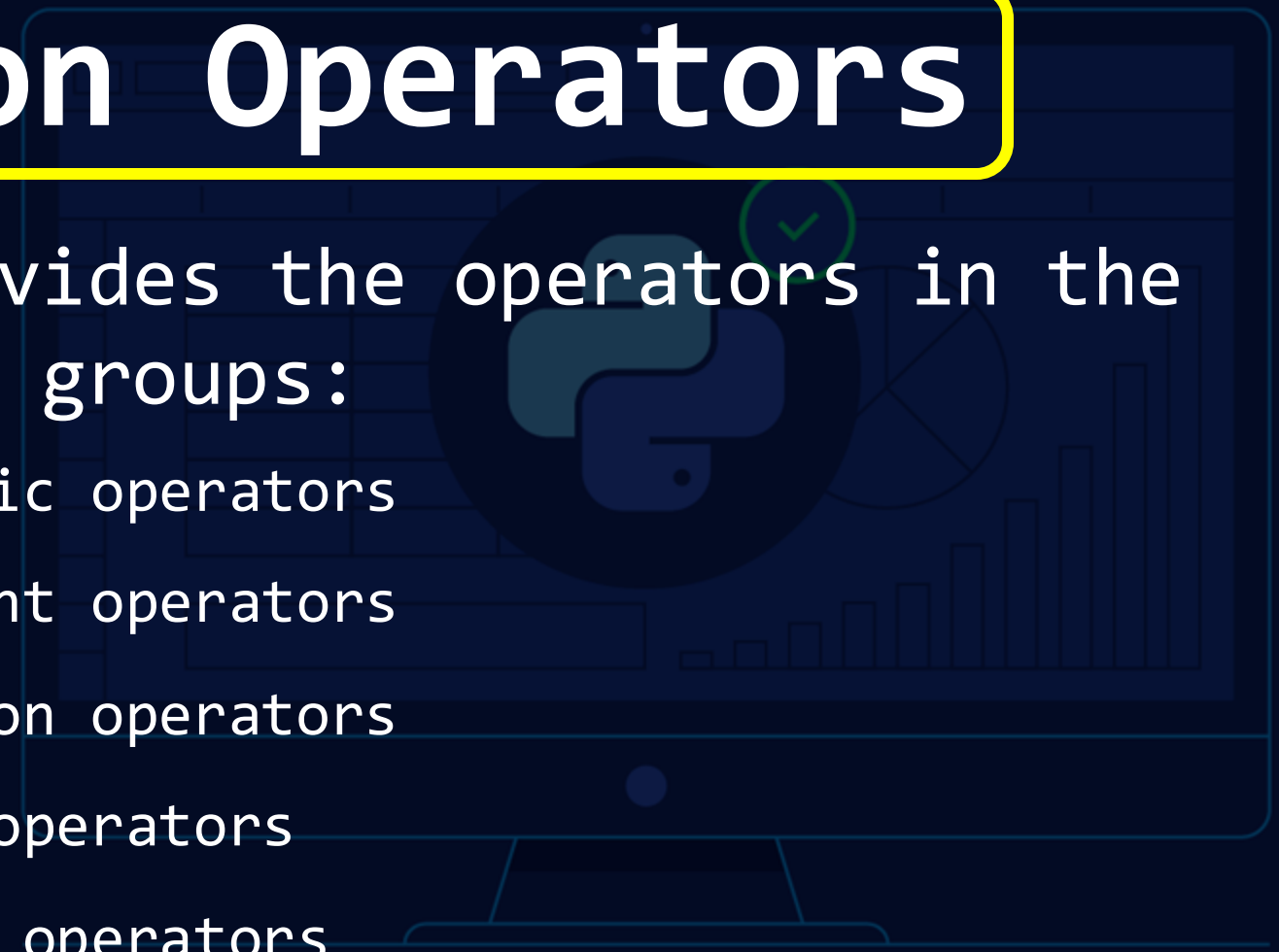
Python Operators

Operators are used to perform operations on variables and values.

In the example below, we use the + operator to add together two values:



Python Operators



Python divides the operators in the following groups:

- Arithmetic operators
- Assignment operators
- Comparison operators
- Logical operators
- Identity operators
- Membership operators
- Bitwise operators

Python Arithmetic Operators

Python Arithmetic Operators

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

Operator	Name	Example
+	Addition	x + y
-	Subtraction	x - y
*	Multiplication	x * y
/	Division	x / y
%	Modulus	x % y
**	Exponentiation	x ** y
//	Floor division	x // y

Python Assignment Operators

Python Assignment Operators

Assignment operators are used to assign values to variables:

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3

Python Assignment Operators

Python Assignment Operators

Assignment operators are used to assign values to variables:

//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

Python Comparison Operators

Python Comparison Operators

Comparison operators are used to compare two values:

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

Python Logical 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)

Python Identity Operators

Python Identity Operators

Identity operators are used to compare the objects, not if they are equal, but if they are actually the same object, with the same memory location:

Operator	Description	Example
is	Returns True if both variables are the same object	x is y
is not	Returns True if both variables are not the same object	x is not y

Python Membership Operators

Python Membership Operators

Membership operators are used to test if a sequence is presented in an object:

Operator	Description	Example
in	Returns True if a sequence with the specified value is present in the object	x in y
not in	Returns True if a sequence with the specified value is not present in the object	x not in y

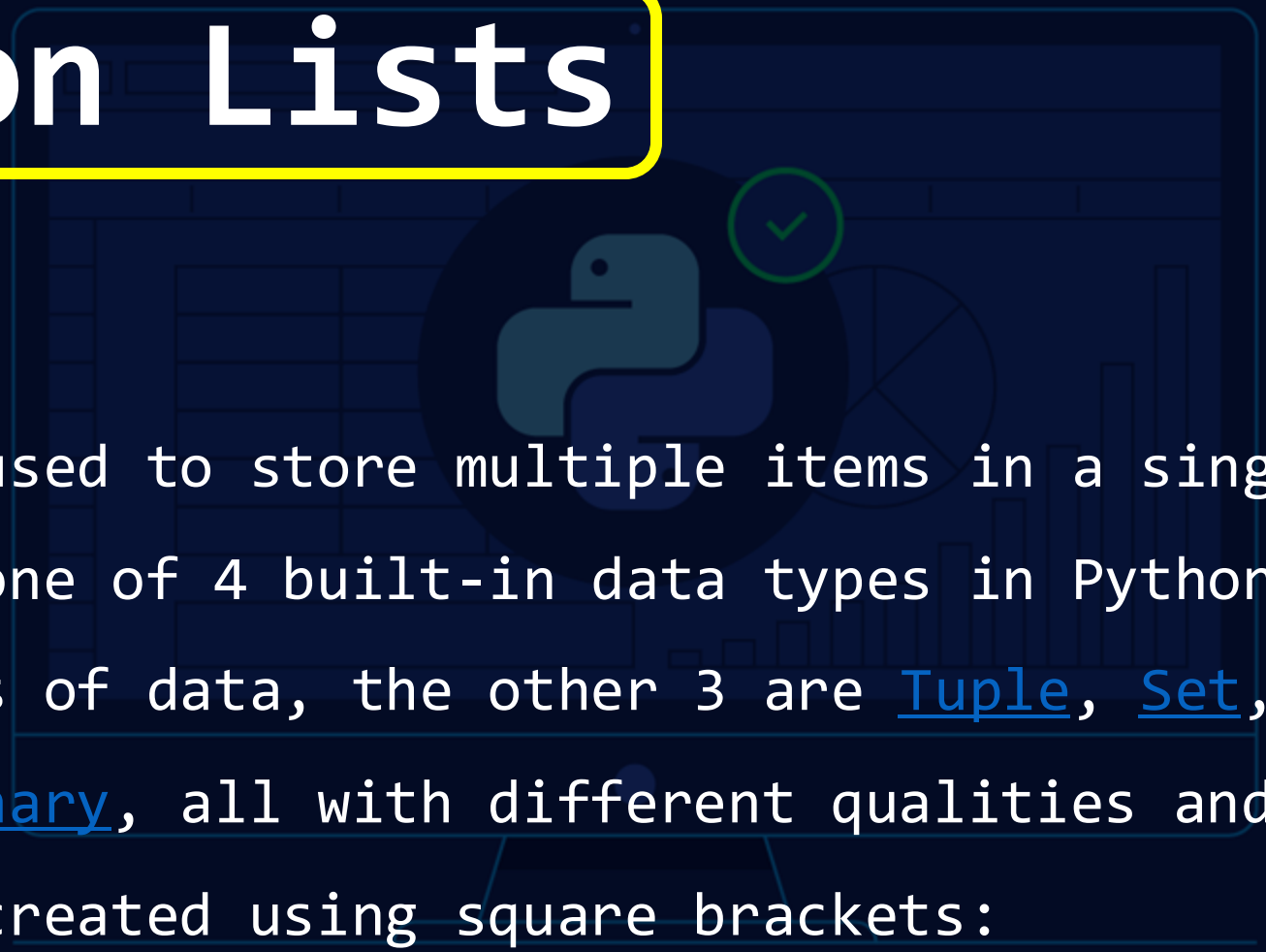
Python Bitwise Operators

Python Bitwise Operators

Bitwise operators are used to compare (binary) numbers:

Operator	Name	Description
&	AND	Sets each bit to 1 if both bits are 1
|	OR	Sets each bit to 1 if one of two bits is 1
^	XOR	Sets each bit to 1 if only one of two bits is 1
~	NOT	Inverts all the bits
<<	Zero fill left shift	Shift left by pushing zeros in from the right and let the leftmost bits fall off
>>	Signed right shift	Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off

Python Lists



List

Lists are used to store multiple items in a single variable.

Lists are one of 4 built-in data types in Python used to store collections of data, the other 3 are Tuple, Set, and Dictionary, all with different qualities and usage.

Lists are created using square brackets:

Python Lists

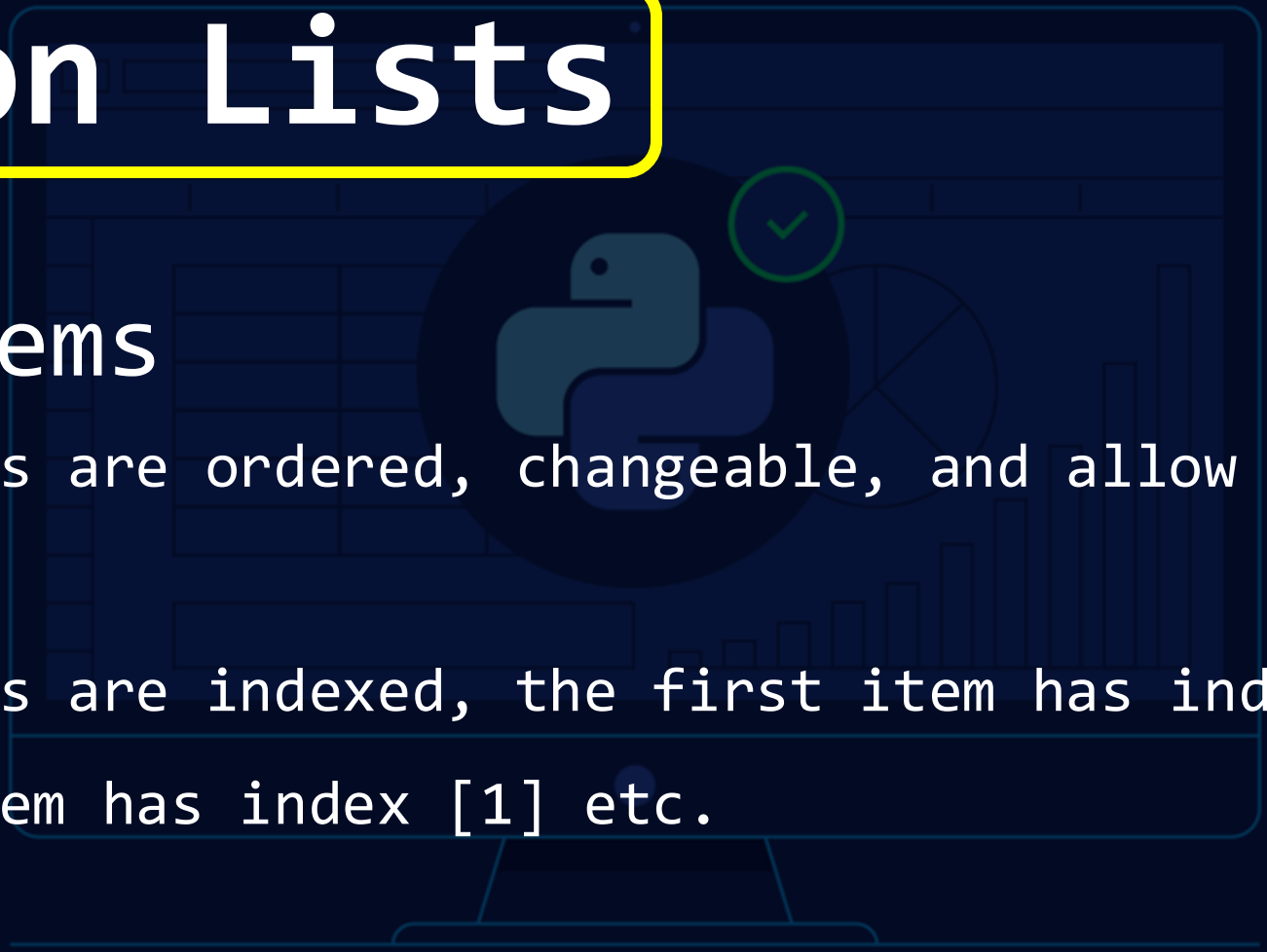
Example

Create a List:

```
thislist = ["anger", "fear", "sadness"]
print(thislist)
```

```
['anger', 'fear', 'sadness']
```

Python Lists



List Items

List items are ordered, changeable, and allow duplicate values.

List items are indexed, the first item has index [0], the second item has index [1] etc.

Python Lists

Ordered

When we say that lists are ordered, it means that the items have a defined order, and that order will not change.

If you add new items to a list, the new items will be placed at the end of the list.

Note: There are some list methods that will change the order, but in general: the order of the items will not change.

Python Lists

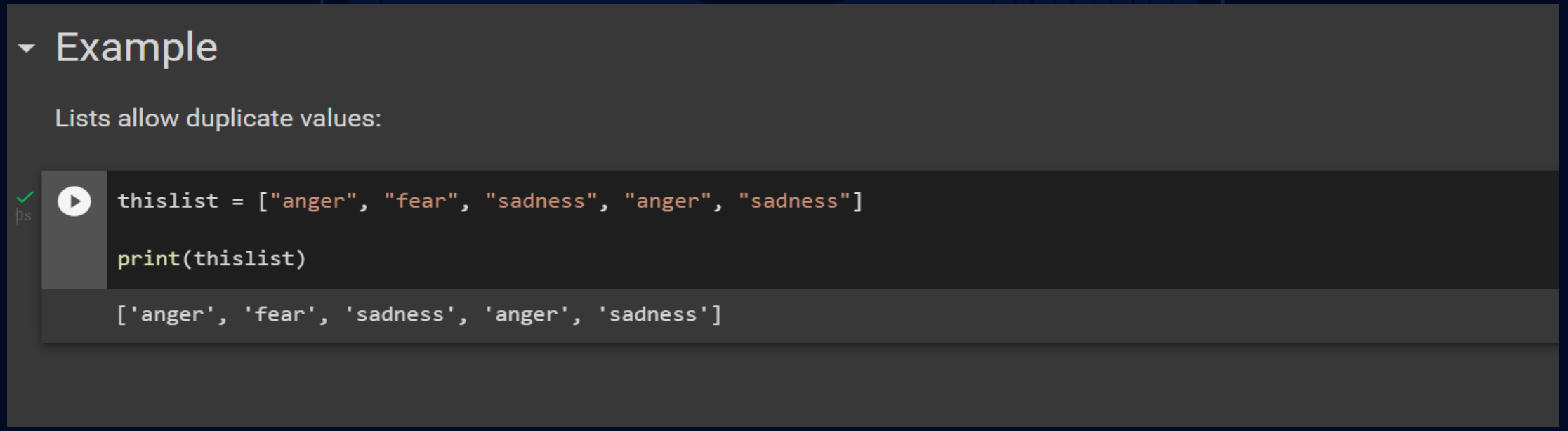
Changeable

The list is changeable, meaning that we can change, add, and remove items in a list after it has been created.

Python Lists

Allow Duplicates

Since lists are indexed, lists can have items with the same value:



Python Lists

List Length

To determine how many items a list has, use the `len()` function:

Example

Print the number of items in the list:

```
thislist = ["anger", "fear", "sadness"]
print(len(thislist))
```

```
3
```

Python Lists

List Items - Data Types

List items can be of any data type:

Example

String, int and boolean data types:

```
list1 = ["anger", "fear", "sadness"]
list2 = [3, 6, 7, 9, 12]
list3 = [True, False, False]

print(list1)
print(list2)
print(list3)
```

```
['anger', 'fear', 'sadness']
[3, 6, 7, 9, 12]
[True, False, False]
```

Python Lists

A list can contain different data types:

Example

A list with strings, integers and boolean values:

```
list1 = ["zyx", 34, True,  68, "喬萬斯"]

print(list1)
```

```
['zyx', 34, True, 68, '喬萬斯']
```

Python Lists

type()

From Python's perspective, lists are defined as objects with the data type 'list':

```
<class 'list'>
```

Example

What is the data type of a list?

```
mylist = ["anger", "fear", "sadness"]

print(type(mylist))
```

```
<class 'list'>
```

Python Lists

The list() Constructor

It is also possible to use the `list()` constructor when creating a new list.

Example

Using the list() constructor to make a List:

```
thislist = list(("anger", "fear", "sadness"))
print(thislist)
```

```
['anger', 'fear', 'sadness']
```

Python Lists

Python Collections (Arrays)

There are four collection data types in the Python programming language:

- **List** is a collection which is ordered and changeable. Allows duplicate members.
- Tuple is a collection which is ordered and unchangeable. Allows duplicate members.
- Set is a collection which is unordered, unchangeable*, and unindexed. No duplicate members.
- Dictionary is a collection which is ordered** and changeable. No duplicate members.

Python - Access List Items

Access Items

List items are indexed and you can access them by referring to the index number:

Example

Print the second item of the list:

```
thislist = ["anger", "fear", "sadness"]
print(thislist[1])
```

```
fear
```

Note: The first item has index 0.

Python - Access List Items

Negative Indexing

Negative indexing means start from the end
-1 refers to the last item, -2 refers to the second last item etc.

Example

Print the last item of the list:

```
thislist = ["anger", "fear", "sadness"]
print(thislist[-1])
```

```
sadness
```

Python - Access List Items

Range of Indexes

You can specify a range of indexes by specifying where to start and where to end the range.

When specifying a range, the return value will be a new list with the specified items.

Example

Return the third, fourth, and fifth item:

```
thislist = ["anger", "fear", "sadness", "hapiness", "surprise", "disgust", "trust"]
print(thislist[2:5])
```

```
['sadness', 'hapiness', 'surprise']
```

Note: The search will start at index 2 (included) and end at index 5 (not included).

Remember that the first item has index 0.

Python - Access List Items

By leaving out the start value, the range will start at the first item:

Example

This example returns the items from the beginning to, but NOT including, "surprise":

```
thislist = ["anger", "fear", "sadness", "happiness", "surprise", "disgust", "trust"]
print(thislist[:4])
```

```
['anger', 'fear', 'sadness', 'happiness']
```

Python - Access List Items

By leaving out the end value, the range will go on to the end of the list:

Example

This example returns the items from "sadness" to the end:

```
thislist = ["anger", "fear", "sadness", "happiness", "surprise", "disgust", "trust"]
print(thislist[2:])
```

```
['sadness', 'happiness', 'surprise', 'disgust', 'trust']
```

Python - Access List Items

Range of Negative Indexes

Specify negative indexes if you want to start the search from the end of the list:

Example

This example returns the items from "happiness" (-4) to, but NOT including "trust" (-1):

```
thislist = ["anger", "fear", "sadness", "happiness", "surprise", "disgust", "trust"]
print(thislist[-4:-1])
```

```
['happiness', 'surprise', 'disgust']
```

Python - Access List Items

Check if Item Exists

To determine if a specified item is present in a list use the in keyword:

Example

Check if "apple" is present in the list:

```
thislist = ["anger", "fear", "sadness"]
if "anger" in thislist:
  print("Yes, 'anger' is in the emotion list")
```

```
Yes, 'anger' is in the emotion list
```

Python - Change List Items

Change Item Value

To change the value of a specific item, refer to the index number:

Example

Change the second item:

```
thislist = ["anger", "fear", "sadness"]
thislist[1] = "happiness"

print(thislist)
```

```
['anger', 'happiness', 'sadness']
```

Python - Change List Items

Change a Range of Item Values

To change the value of items within a specific range, define a list with the new values, and refer to the range of index numbers where you want to insert the new values:

Example

Change the values "fear" and "sadness" with the values "trust" and "anticipation":

```
thislist = ["anger", "fear", "sadness", "happiness", "surprise", "disgust"]

thislist[1:3] = ["trust", "anticipation"]

print(thislist)
```

```
['anger', 'trust', 'anticipation', 'happiness', 'surprise', 'disgust']
```

Python - Change List Items

If you insert *more* items than you replace, the new items will be inserted where you specified, and the remaining items will move accordingly:

Example

Change the second value by replacing it with two new values:

```
thislist = ["anger", "fear", "sadness"]

thislist[1:2] = ["trust", "anticipation"]

print(thislist)
```

```
['anger', 'trust', 'anticipation', 'sadness']
```

Note: The length of the list will change when the number of items inserted does not match the number of items replaced.

Python - Change List Items

If you insert *less* items than you replace, the new items will be inserted where you specified, and the remaining items will move accordingly:

Example

Change the second and third value by replacing it with one value:

```
thislist = ["anger", "fear", "sadness"]

thislist[1:3] = ["trust"]

print(thislist)
```

```
['anger', 'trust']
```

Python - Change List Items

Insert Items

To insert a new list item, without replacing any of the existing values, we can use the insert() method.
The insert() method inserts an item at the specified index:

Example

Insert "trust" as the third item:

```
thislist = ["anger", "fear", "sadness"]

thislist.insert(2, "trust")

print(thislist)
```

```
['anger', 'fear', 'trust', 'sadness']
```

Note: As a result of the example above, the list will now contain 4 items.

Python - Add List Items

Append Items

To add an item to the end of the list, use the append() method:

Example

Using the append() method to append an item:

```
thislist = ["anger", "fear", "sadness"]

thislist.append("surprise")

print(thislist)
```

```
['anger', 'fear', 'sadness', 'surprise']
```

Python - Add List Items

Insert Items

To insert a list item at a specified index, use the `insert()` method.
The `insert()` method inserts an item at the specified index:

Example

Insert an item as the second position:

```
thislist = ["anger", "fear", "sadness"]
thislist.insert(1, "surprise")
print(thislist)
```

```
['anger', 'surprise', 'fear', 'sadness']
```

Note: As a result of the examples above, the lists will now contain 4 items.

Python - Add List Items

Extend List

To append elements from *another list* to the current list, use the extend() method.

Example

Add the elements of emottion to thislist:

```
thislist = ["anger", "fear", "sadness"]
emotion = ["happiness", "surprise", "disgust"]

thislist.extend(emotion)

print(thislist)
```

```
['anger', 'fear', 'sadness', 'happiness', 'surprise', 'disgust']
```

Python - Add List Items

Add Any Iterable

The extend() method does not have to append *lists*, you can add any iterable object (tuples, sets, dictionaries etc.).

Example

Add elements of a tuple to a list:

```
thislist = ["anger", "fear", "sadness"]
thistuple = ("happiness", "surprise")

thislist.extend(thistuple)

print(thislist)
```

```
['anger', 'fear', 'sadness', 'happiness', 'surprise']
```

Python - Remove List Items

Remove Specified Item

The remove() method removes the specified item.

Example

Remove "banana":

```
thislist = ["anger", "fear", "sadness"]
thislist.remove("fear")
print(thislist)
```

```
['anger', 'sadness']
```

Python - Remove List Items

Remove Specified Index

The pop() method removes the specified index.

Example

Remove the second item:

```
thislist = ["anger", "fear", "sadness"]
thislist.pop(1)
print(thislist)
```

```
['anger', 'sadness']
```

Python - Remove List Items

If you do not specify the index, the pop() method removes the last item.

Example

Remove the last item:

```
thislist = ["anger", "fear", "sadness"]
thislist.pop()
print(thislist)
```

```
['anger', 'fear']
```

Python - Remove List Items

The del keyword also removes the specified index:

Example

Remove the first item:

```
thislist = ["anger", "fear", "sadness"]
del thislist[0]
print(thislist)
```

```
['fear', 'sadness']
```

Python - Remove List Items

The del keyword can also delete the list completely.

Example

Delete the entire list:

```
thislist = ["anger", "fear", "sadness"]
del thislist
print(thislist)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-40-a03aaca960bc> in <module>
      1 thislist = ["anger", "fear", "sadness"]
      2 del thislist
----> 3 print(thislist)

NameError: name 'thislist' is not defined
```

SEARCH STACK OVERFLOW

Python - Remove List Items

Clear the List

The clear() method empties the list.

The list still remains, but it has no content.

Example

Clear the list content:

```
thislist = ["anger", "fear", "sadness"]
thislist.clear()
print(thislist)
```

```
[]
```

Python - Loop Lists

Loop Through a List

You can loop through the list items by using a `for` loop:

Example

Print all items in the list, one by one:

```
thislist = ["anger", "fear", "sadness"]
for x in thislist:
  print(x)
```

```
anger
fear
sadness
```

Python - Loop Lists

Loop Through the Index Numbers

You can also loop through the list items by referring to their index number.

Use the range() and len() functions to create a suitable iterable.

Example

Print all items by referring to their index number:

```
thislist = ["anger", "fear", "sadness"]
for i in range(len(thislist)):
  print(thislist[i])
```

```
anger
fear
sadness
```

Python - Loop Lists

Using a While Loop

You can loop through the list items by using a while loop.

Use the len() function to determine the length of the list, then start at 0 and loop your way through the list items by referring to their indexes.

Remember to increase the index by 1 after each iteration.

Python - Loop Lists

Example

Print all items, using a while loop to go through all the index numbers

```
thislist = ["anger", "fear", "sadness"]
i = 0
while i < len(thislist):
  print(thislist[i])
  i = i + 1
```

```
anger
fear
sadness
```

Python - Loop Lists

Looping Using List Comprehension

List Comprehension offers the shortest syntax for looping through lists:

Example

A short hand for loop that will print all items in a list:

```
thislist = ["anger", "fear", "sadness"]
[print(x) for x in thislist]
```

```
anger
fear
sadness
[None, None, None]
```

Python - List Comprehension

List Comprehension

List comprehension offers a shorter syntax when you want to create a new list based on the values of an existing list.

Example:

Based on a list of fruits, you want a new list, containing only the fruits with the letter "a" in the name.

Without list comprehension you will have to write a for statement with a conditional test inside:

Python - List Comprehension

Example

```
emotiions = ["anger", "fear", "sadness", "happiness", "surprise"]
newlist = []

for x in emotiions:
  if "a" in x:
    newlist.append(x)

print(newlist)
```

```
['anger', 'fear', 'sadness', 'happiness']
```

Python - List Comprehension

With list comprehension you can do all that with only one line of code:

Example

```
emotiions = ["anger", "fear", "sadness", "happiness", "surprise", "disgust"]
newlist = [x for x in emotiions if "a" in x]

print(newlist)
```

```
['anger', 'fear', 'sadness', 'happiness']
```

Python - List Comprehension

The Syntax

```
newlist = [expression for item in iterable if condition == True]
```

The return value is a new list, leaving the old list unchanged.

Python - List Comprehension

Condition

The *condition* is like a filter that only accepts the items that valuate to True.

Example

Only accept items that are not "anger":

```
[51] emotiions = ["anger", "fear", "sadness", "happiness", "surprise", "disgust"]

     newlist = [x for x in emotiions if x != "anger"]

     print(newlist)
```

```
['fear', 'sadness', 'happiness', 'surprise', 'disgust']
```

Python - List Comprehension

The condition `if x != "apple"` will return True for all elements other than "apple", making the new list contain all fruits except "apple".
The *condition* is optional and can be omitted:

Example

With no if statement:

```
emotiions = ["anger", "fear", "sadness", "happiness", "surprise", "disgust"]

newlist = [x for x in emotiions]

print(newlist)
```

Python - List Comprehension

Iterable

The *iterable* can be any iterable object, like a list, tuple, set etc.

Example

You can use the range() function to create an iterable:

```
newlist = [x for x in range(19)]

print(newlist)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18]
```

Python - List Comprehension

Same example, but with a condition:

Example

Accept only numbers lower than 5:

```
newlist = [x for x in range(10) if x < 5]

print(newlist)
```

```
[0, 1, 2, 3, 4]
```

Python - List Comprehension

Expression

The *expression* is the current item in the iteration, but it is also the outcome, which you can manipulate before it ends up like a list item in the new list:

Example

Set the values in the new list to upper case:

```
emotions = ["anger", "fear", "sadness", "happiness", "surprise", "disgust"]

newlist = [x.upper() for x in emotions]

print(newlist)
```

```
['ANGER', 'FEAR', 'SADNESS', 'HAPPINESS', 'SURPRISE', 'DISGUST']
```

Python - List Comprehension

You can set the outcome to whatever you like:

Example

Set all values in the new list to 'hello':

```
emotions = ["anger", "fear", "sadness", "happiness", "surprise", "disgust"]

newlist = ['hello' for x in emotions]

print(newlist)
```

```
['hello', 'hello', 'hello', 'hello', 'hello', 'hello']
```

Python - List Comprehension

The *expression* can also contain conditions, not like a filter, but as a way to manipulate the outcome:

Example

Return "orange" instead of "banana":

```
emotions = ["anger", "fear", "sadness", "happiness", "surprise", "disgust"]

newlist = [x if x != "fear" else "trust" for x in emotions]

print(newlist)
```

```
['anger', 'trust', 'sadness', 'happiness', 'surprise', 'disgust']
```

The *expression* in the example above says:
"Return the item if it is not fear, if it is fear return trust".

Python - Sort Lists

Sort List Alphanumerically

List objects have a sort() method that will sort the list alphanumerically, ascending, by default:

Example

Sort the list alphabetically:

```
thislist = ["anger", "fear", "sadness", "happiness", "surprise", "disgust"]

thislist.sort()

print(thislist)
```

```
['anger', 'disgust', 'fear', 'happiness', 'sadness', 'surprise']
```

Python - Sort Lists

Example

Sort the list numerically:

```
thislist = [199, 547, 643, 422, 230]

thislist.sort()

print(thislist)
```

```
[199, 230, 422, 547, 643]
```

Python - Sort Lists

Sort Descending

To sort descending, use the keyword argument reverse = True:

Example

Sort the list descending:

```
thislist = ["anger", "fear", "sadness", "happiness", "surprise", "disgust"]

thislist.sort(reverse = True)

print(thislist)
```

```
['surprise', 'sadness', 'happiness', 'fear', 'disgust', 'anger']
```

Python - Sort Lists

Example

Sort the list descending:

```
thislist = [199, 547, 643, 422, 230]

thislist.sort(reverse = True)

print(thislist)
```

```
[643, 547, 422, 230, 199]
```

Python - Sort Lists

Customize Sort Function

You can also customize your own function by using the keyword argument key = *function*.

The function will return a number that will be used to sort the list (the lowest number first):

Example

Sort the list based on how close the number is to 50:

```
def myfunc(n):
  return abs(n - 199)

thislist = [199, 547, 643, 422, 230]

thislist.sort(key = myfunc)

print(thislist)
```

```
[199, 230, 422, 547, 643]
```

Python - Sort Lists

Case Insensitive Sort

By default the sort() method is case sensitive, resulting in all capital letters being sorted before lower case letters:

Example

Case sensitive sorting can give an unexpected result:

```
thislist = ["anger", "fear", "sadness", "happiness", "surprise", "disgust"]

thislist.sort()

print(thislist)
```

```
['anger', 'disgust', 'fear', 'happiness', 'sadness', 'surprise']
```

Python - Sort Lists

So if you want a case-insensitive sort function, use str.lower as a key function:

Example

Perform a case-insensitive sort of the list:

```
thislist = ["anger", "fear", "sadness", "happiness", "surprise", "disgust"]

thislist.sort(key = str.lower)

print(thislist)
```

```
['anger', 'disgust', 'fear', 'happiness', 'sadness', 'surprise']
```

Python - Sort Lists

Reverse Order

What if you want to reverse the order of a list, regardless of the alphabet?
The reverse() method reverses the current sorting order of the elements.

Example

Reverse the order of the list items:

```
thislist = ["anger", "fear", "sadness", "happiness", "surprise", "disgust"]

thislist.reverse()

print(thislist)
```

```
['disgust', 'surprise', 'happiness', 'sadness', 'fear', 'anger']
```

Python - Copy Lists

Copy a List

You cannot copy a list simply by typing `list2 = list1`, because: `list2` will only be a *reference* to `list1`, and changes made in `list1` will automatically also be made in `list2`.
There are ways to make a copy, one way is to use the built-in List method `copy()`.

Example

Make a copy of a list with the copy() method:

```
thislist = ["anger", "fear", "sadness"]
mylist = thislist.copy()
print(mylist)
```

```
['anger', 'fear', 'sadness']
```

Python - Copy Lists

Another way to make a copy is to use the built-in method list().

Example

Make a copy of a list with the list() method:

```
thislist = ["anger", "fear", "sadness"]
mylist = list(thislist)
print(mylist)
```

```
['anger', 'fear', 'sadness']
```

Python - Join Lists

Join Two Lists

There are several ways to join, or concatenate, two or more lists in Python.

One of the easiest ways are by using the + operator.

Example

Join two list:

```
list1 = ["a", "b", "c"]
list2 = [1, 2, 3]

list3 = list1 + list2
print(list3)
```

```
['a', 'b', 'c', 1, 2, 3]
```

Python - Join Lists

Another way to join two lists is by appending all the items from list2 into list1, one by one:

Example

Append list2 into list1:

```
list1 = ["a", "b" , "c"]
list2 = [1, 2, 3]

for x in list2:
  list1.append(x)

print(list1)
```

```
['a', 'b', 'c', 1, 2, 3]
```

Python - Join Lists

Or you can use the extend() method, which purpose is to add elements from one list to another list:

Example

Use the extend() method to add list2 at the end of list1:

```
list1 = ["a", "b" , "c"]
list2 = [1, 2, 3]

list1.extend(list2)
print(list1)
```

```
['a', 'b', 'c', 1, 2, 3]
```

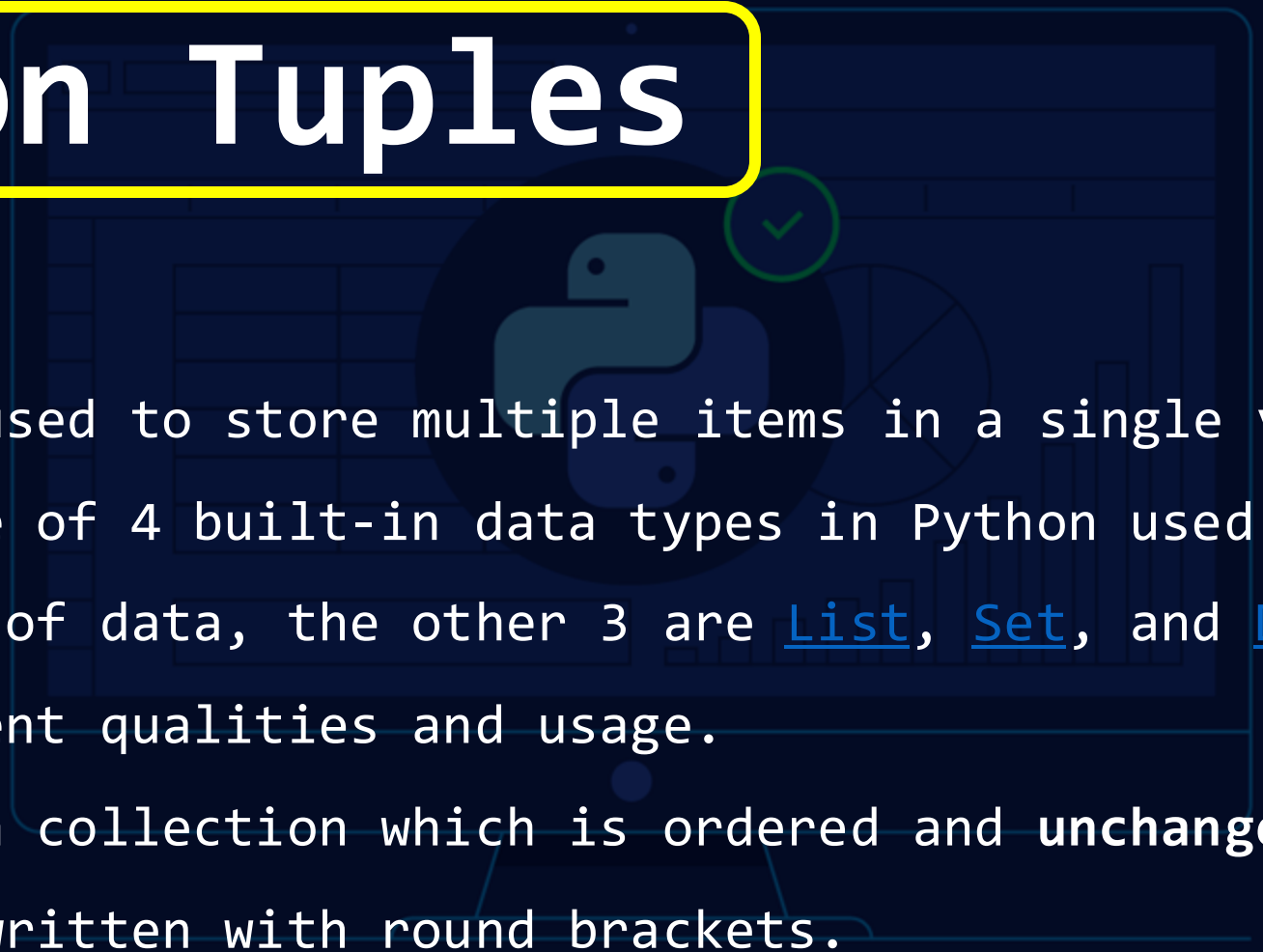
Python - List Methods

List Methods

Python has a set of built-in methods that you can use on lists.

Method	Description
append()	Adds an element at the end of the list
clear()	Removes all the elements from the list
copy()	Returns a copy of the list
count()	Returns the number of elements with the specified value
extend()	Add the elements of a list (or any iterable), to the end of the current list
index()	Returns the index of the first element with the specified value
insert()	Adds an element at the specified position
pop()	Removes the element at the specified position
remove()	Removes the item with the specified value
reverse()	Reverses the order of the list
sort()	Sorts the list

Python Tuples



Tuple

Tuples are used to store multiple items in a single variable.

Tuple is one of 4 built-in data types in Python used to store collections of data, the other 3 are List, Set, and Dictionary, all with different qualities and usage.

A tuple is a collection which is ordered and **unchangeable**.

Tuples are written with round brackets.

Python Tuples

Example

Create a Tuple:

```
thistuple = ("anger", "fear", "sadness")
print(thistuple)
```

```
('anger', 'fear', 'sadness')
```

Python Tuples

Tuple Items

Tuple items are ordered, unchangeable, and allow duplicate values.

Tuple items are indexed, the first item has index `[0]`, the second item has index `[1]` etc.

Ordered

When we say that tuples are ordered, it means that the items have a defined order, and that order will not change.

Unchangeable

Tuples are unchangeable, meaning that we cannot change, add or remove items after the tuple has been created.

Python Tuples

Allow Duplicates

Since tuples are indexed, they can have items with the same value:

Example

Tuples allow duplicate values:

```
thistuple = ("anger", "fear", "sadness", "anger", "sadness")
print(thistuple)
```

```
('anger', 'fear', 'sadness', 'anger', 'sadness')
```

Python Tuples

Tuple Length

To determine how many items a tuple has, use the `len()` function:

Example

Print the number of items in the tuple:

```
thistuple = tuple(("anger", "fear", "sadness"))
print(len(thistuple))
```

```
3
```

Python Tuples

Create Tuple With One Item

To create a tuple with only one item, you have to add a comma after the item, otherwise Python will not recognize it as a tuple.

Example

One item tuple, remember the comma:

```
thistuple = ("anger",)
print(type(thistuple))

#NOT a tuple
thistuple = ("anger")
print(type(thistuple))
```

```
<class 'tuple'>
<class 'str'>
```

Python Tuples

Tuple Items - Data Types

Tuple items can be of any data type:

Example

String, int and boolean data types:

```
tuple1 = ("anger", "fear", "sadness")
tuple2 = (1, 5, 7, 9, 3)
tuple3 = (True, False, False)

print(tuple1)
print(tuple2)
print(tuple3)
```

```
('anger', 'fear', 'sadness')
(1, 5, 7, 9, 3)
(True, False, False)
```

Python Tuples

A tuple can contain different data types:

Example

A tuple with strings, integers and boolean values:

```
tuple1 = ("abc", 34, True, 40, "male")

print(tuple1)
```

```
('abc', 34, True, 40, 'male')
```

Python Tuples

type()

From Python's perspective, tuples are defined as objects with the data type 'tuple':

```
<class 'tuple'>
```

Example

Print the number of items in the tuple:

```
thistuple = tuple(("anger", "fear", "sadness"))
print(len(thistuple))
```

```
3
```

Python Tuples

The tuple() Constructor

It is also possible to use the tuple() constructor to make a tuple.

Example

Using the tuple() method to make a tuple:

```
thistuple = tuple(("anger", "fear", "sadness"))
print(thistuple)
```

```
('anger', 'fear', 'sadness')
```

Python - Access Tuple Items

Access Tuple Items

You can access tuple items by referring to the index number, inside square brackets:

Example

Print the second item in the tuple:

```
thistuple = ("anger", "fear", "sadness")
print(thistuple[1])
```

```
fear
```

Note: The first item has index 0.

Python - Access Tuple Items

Negative Indexing

Negative indexing means start from the end.

-1 refers to the last item, -2 refers to the second last item etc.

Example

Print the last item of the tuple:

```
thistuple = ("anger", "fear", "sadness")
print(thistuple[-1])
```

```
sadness
```

Python - Access Tuple Items

Range of Indexes

You can specify a range of indexes by specifying where to start and where to end the range.
When specifying a range, the return value will be a new tuple with the specified items.

Example

Return the third, fourth, and fifth item:

```
thistuple = ("anger", "fear", "sadness", "happiness", "surprise", "disgust")
print(thistuple[2:5])
```

```
('sadness', 'happiness', 'surprise')
```

Note: The search will start at index 2 (included) and end at index 5 (not included).
Remember that the first item has index 0.

Python - Access Tuple Items

By leaving out the start value, the range will start at the first item:

Example

This example returns the items from the beginning to, but NOT included, "surprise":

```
thistuple = ("anger", "fear", "sadness", "happiness", "surprise", "disgust")

print(thistuple[:4])
```

```
('anger', 'fear', 'sadness', 'happiness')
```

Python - Access Tuple Items

By leaving out the end value, the range will go on to the end of the list:

Example

This example returns the items from "sadness" and to the end:

```
[29] thistuple = ("anger", "fear", "sadness", "happiness", "surprise", "disgust")

     print(thistuple[2:])
```

```
('sadness', 'happiness', 'surprise', 'disgust')
```

Python - Access Tuple Items

Range of Negative Indexes

Specify negative indexes if you want to start the search from the end of the tuple:

Example

This example returns the items from index -4 (included) to index -1 (excluded)

```
thistuple = ("anger", "fear", "sadness", "happiness", "surprise", "disgust")
print(thistuple[-4:-1])
```

```
('sadness', 'happiness', 'surprise')
```

Python - Access Tuple Items

Check if Item Exists

To determine if a specified item is present in a tuple use the in keyword:

Example

Check if "anger" is present in the tuple:

```
thistuple = ("anger", "fear", "sadness")
if "anger" in thistuple:
  print("Yes, 'anger' is in the fruits tuple")
```

```
Yes, 'anger' is in the fruits tuple
```

Python - Update Tuples

Change Tuple Values

Once a tuple is created, you cannot change its values. Tuples are **unchangeable**, or **immutable** as it also is called.
But there is a workaround. You can convert the tuple into a list, change the list, and convert the list back into a tuple.

Example

Convert the tuple into a list to be able to change it:

```
x = ("anger", "fear", "sadness")
y = list(x)
y[1] = "sadness"
x = tuple(y)

print(x)
```

```
('anger', 'sadness', 'sadness')
```

Python - Update Tuples

Add Items

Since tuples are immutable, they do not have a build-in append() method, but there are other ways to add items to a tuple.

1. **Convert into a list**: Just like the workaround for *changing* a tuple, you can convert it into a list, add your item(s), and convert it back into a tuple.

Example

Convert the tuple into a list, add "orange", and convert it back into a tuple:

```
thistuple = ("anger", "fear", "sadness")
y = list(thistuple)
y.append("sadness")
thistuple = tuple(y)

print(thistuple)
```

```
('anger', 'fear', 'sadness', 'sadness')
```

Python - Update Tuples

2. **Add tuple to a tuple.** You are allowed to add tuples to tuples, so if you want to add one item, (or many), create a new tuple with the item(s), and add it to the existing tuple:

Example

Create a new tuple with the value "orange", and add that tuple:

```
thistuple = ("anger", "fear", "sadness")
y = ("sadness",)
thistuple += y

print(thistuple)
```

```
('anger', 'fear', 'sadness', 'sadness')
```

Note: When creating a tuple with only one item, remember to include a comma after the item, otherwise it will not be identified as a tuple.

Python - Update Tuples

Remove Items

Note: You cannot remove items in a tuple.

Tuples are **unchangeable**, so you cannot remove items from it, but you can use the same workaround as we used for changing and adding tuple items:

Example

Convert the tuple into a list, remove "anger", and convert it back into a tuple:

```
thistuple = ("anger", "fear", "sadness")
y = list(thistuple)
y.remove("anger")
thistuple = tuple(y)

print(thistuple)
```

```
('fear', 'sadness')
```

Python - Update Tuples

Or you can delete the tuple completely:

```
[3] thistuple = ("anger", "fear", "sadness")
    del thistuple
    print(thistuple)
```

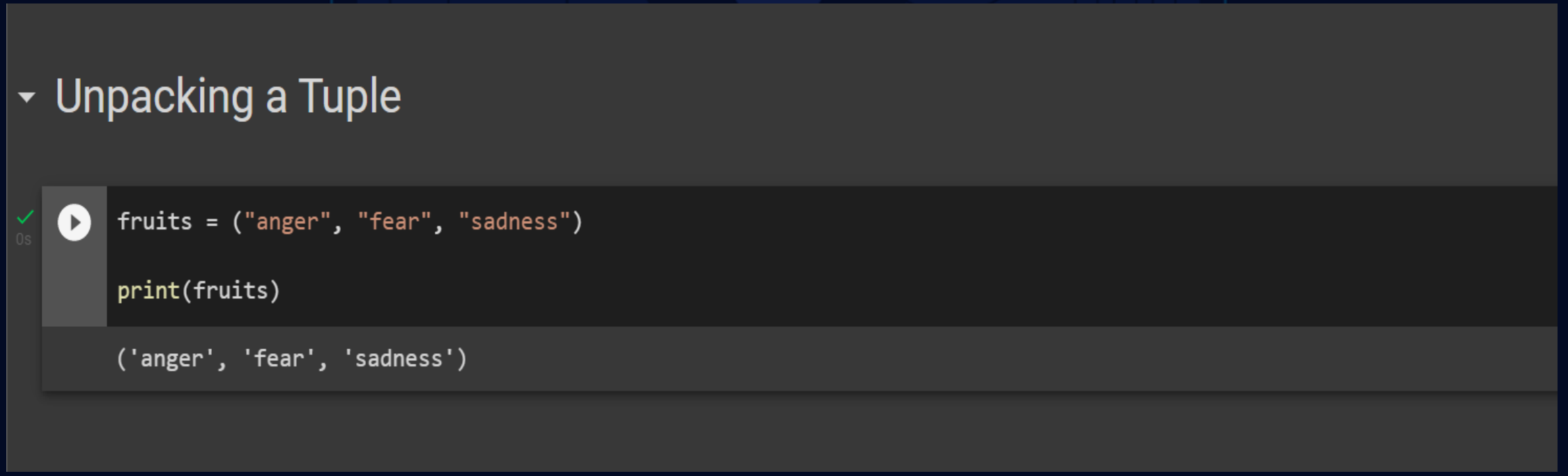
```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-3-b5fee61f7add> in <module>
      1 thistuple = ("anger", "fear", "sadness")
      2 del thistuple
----> 3 print(thistuple)

NameError: name 'thistuple' is not defined
```

SEARCH STACK OVERFLOW

Python - Unpack Tuples

Unpacking a Tuple

When we create a tuple, we normally assign values to it. This is called "packing" a tuple:



Python - Update Tuples

But, in Python, we are also allowed to extract the values back into variables. This is called "unpacking":

Example

Unpacking a tuple:

```
emotion = ("anger", "fear", "sadness")

(green, yellow, red) = emotion

print(green)
print(yellow)
print(red)
```

```
anger
fear
sadness
```

Note: The number of variables must match the number of values in the tuple, if not, you must use an asterisk to collect the remaining values as a list.

Python - Unpack Tuples

Using Asterisk*

If the number of variables is less than the number of values, you can add an * to the variable name and the values will be assigned to the variable as a list:

Example

Assign the rest of the values as a list called "red":

```
emotions = ("anger", "fear", "sadness", "happiness", "surprise", "disgust")

(green, yellow, *red) = emotions

print(green)
print(yellow)
print(red)
```

```
anger
fear
['sadness', 'happiness', 'surprise', 'disgust']
```

Python - Unpack Tuples

If the asterisk is added to another variable name than the last, Python will assign values to the variable until the number of values left matches the number of variables left.

Example

Add a list of values the "tropic" variable:

```
emotions = ("anger", "fear", "sadness", "happiness", "surprise", "disgust")

(green, *tropic, red) = emotions

print(green)
print(tropic)
print(red)
```

```
anger
['fear', 'sadness', 'happiness', 'surprise']
disgust
```

Python - Loop Tuples

Loop Through a Tuple

You can loop through the tuple items by using a for loop.

Example

Iterate through the items and print the values:

```
thistuple = ("anger", "fear", "sadness")
for x in thistuple:
  print(x)
```

```
anger
fear
sadness
```

Python - Loop Tuples

Loop Through the Index Numbers

You can also loop through the tuple items by referring to their index number.

Use the range() and len() functions to create a suitable iterable.

Example

Print all items by referring to their index number:

```
thistuple = ("anger", "fear", "sadness")
for i in range(len(thistuple)):
  print(thistuple[i])
```

```
anger
fear
sadness
```

Python - Loop Tuples

Using a While Loop

You can loop through the list items by using a while loop.
Use the len() function to determine the length of the tuple, then start at 0 and loop your way through the tuple items by refering to their indexes.
Remember to increase the index by 1 after each iteration.

Example

Print all items, using a while loop to go through all the index numbers:

```
thistuple = ("anger", "fear", "sadness")
i = 0
while i < len(thistuple):
  print(thistuple[i])
  i = i + 1
```

```
anger
fear
sadness
```

Python - Join Tuples

Join Two Tuples

To join two or more tuples you can use the + operator:

Example

Join two tuples:

```
tuple1 = ("a", "b" , "c")
tuple2 = (1, 2, 3)

tuple3 = tuple1 + tuple2
print(tuple3)
```

```
('a', 'b', 'c', 1, 2, 3)
```

Python - Join Tuples

Multiply Tuples

If you want to multiply the content of a tuple a given number of times, you can use the * operator:

Example

Multiply the fruits tuple by 2:

```
emotions = ("anger", "fear", "sadness")
mytuple = emotions * 2

print(mytuple)
```

```
('anger', 'fear', 'sadness', 'anger', 'fear', 'sadness')
```

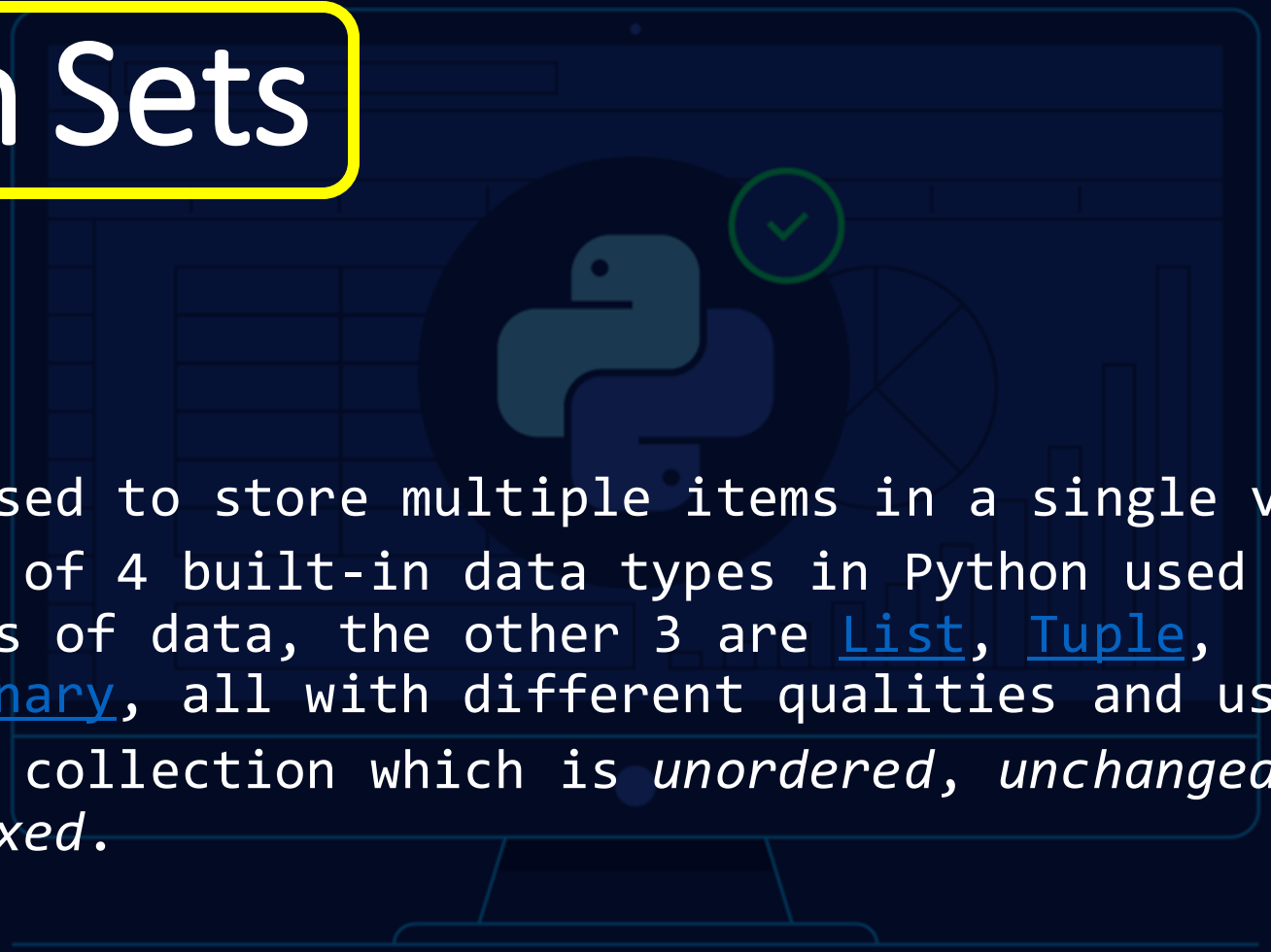
Python - Tuple Methods

Tuple Methods

Python has two built-in methods that you can use on tuples.

Method	Description
count()	Returns the number of times a specified value occurs in a tuple
index()	Searches the tuple for a specified value and returns the position of where it was found

Python Sets



Set

Sets are used to store multiple items in a single variable.

Set is one of 4 built-in data types in Python used to store collections of data, the other 3 are List, Tuple, and Dictionary, all with different qualities and usage.

A set is a collection which is *unordered*, *unchangeable**, and *unindexed*.

* **Note:** Set *items* are unchangeable, but you can remove items and add new items.

Python Sets

Sets are written with curly brackets.

Example

Create a Set:

```
thisset = {"anger", "fear", "sadness"}
print(thisset)
```

```
{'anger', 'sadness', 'fear'}
```

Note: Sets are unordered, so you cannot be sure in which order the items will appear.

Python Sets

Set Items

Set items are unordered, unchangeable, and do not allow duplicate values.

Unordered

Unordered means that the items in a set do not have a defined order.

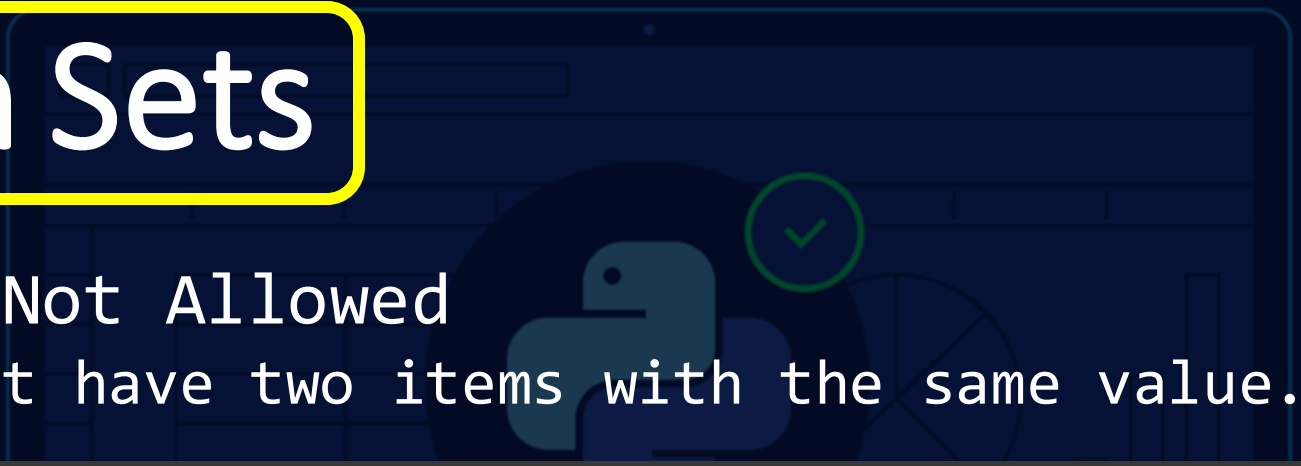
Set items can appear in a different order every time you use them, and cannot be referred to by index or key.

Unchangeable

Set items are unchangeable, meaning that we cannot change the items after the set has been created.

Once a set is created, you cannot change its items, but you can remove items and add new items.

Python Sets



Duplicates Not Allowed

Sets cannot have two items with the same value.

Example

Duplicate values will be ignored:

```
thisset = {"anger", "fear", "sadness"}

print(thisset)
```

```
{'anger', 'sadness', 'fear'}
```

Python Sets

Get the Length of a Set

To determine how many items a set has, use the len() function.

Example

Get the number of items in a set:

```
thisset = {"anger", "fear", "sadness"}

print(len(thisset))
```

```
3
```

Python Sets

Set Items - Data Types

Set items can be of any data type:

Example

String, int and boolean data types:

```
set1 = {"anger", "fear", "sadness"}
set2 = {1, 5, 7, 9, 3}
set3 = {True, False, False}

print(set1)
print(set2)
print(set3)
```

```
{'anger', 'sadness', 'fear'}
{1, 3, 5, 7, 9}
{False, True}
```

Python Sets

A set can contain different data types:

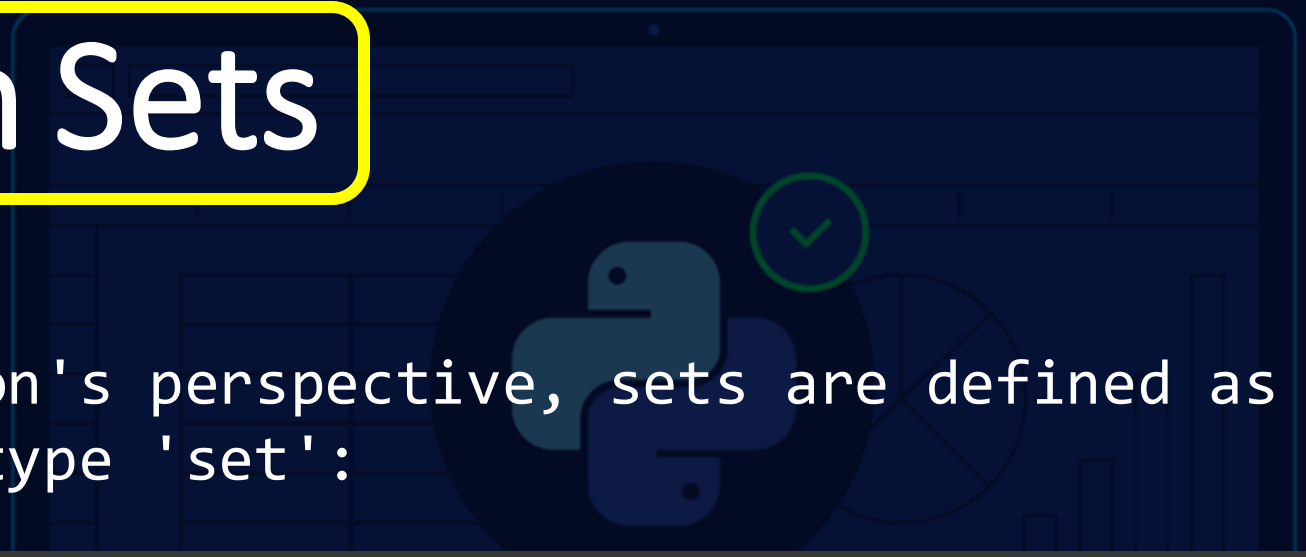
Example

A set with strings, integers and boolean values:

```
set1 = {"abc", 34, True, 40, "male"}

print(set1)
```

```
{True, 34, 40, 'male', 'abc'}
```

Python Sets



type()

From Python's perspective, sets are defined as objects with the data type 'set':

Example

What is the data type of a set?

```
myset = {"anger", "fear", "sadness"}

print(type(myset))
```

Python Sets

The set() Constructor

It is also possible to use the set() constructor to make a set.

Example

Using the set() constructor to make a set:

```
thisset = set(("anger", "fear", "sadness"))
print(thisset)
```

```
{'anger', 'sadness', 'fear'}
```

Python Sets

Access Items

You cannot access items in a set by referring to an index or a key.

But you can loop through the set items using a for loop, or ask if a specified value is present in a set, by using the in keyword.

Example

Loop through the set, and print the values:

```
thisset = {"anger", "fear", "sadness"}

for x in thisset:
  print(x)
```

```
anger
sadness
fear
```

Python - Access Set Items

Access Items

You cannot access items in a set by referring to an index or a key.
But you can loop through the set items using a for loop, or ask if a specified value is present in a set, by using the in keyword.

Example

Check if "fear" is present in the set:

```
thisset = {"anger", "fear", "sadness"}

print("fear" in thisset)
```

```
True
```

Python - Add Set Items

Add Sets

To add items from another set into the current set, use the update() method.

Example

Add elements from tropical into thisset:

```
thisset = {"anger", "fear", "sadness"}
tropical = {"happiness", "surprise", "disgust"}

thisset.update(tropical)

print(thisset)
```

```
{'sadness', 'anger', 'disgust', 'surprise', 'fear', 'happiness'}
```

Python - Add Set Items

Add Items

Once a set is created, you cannot change its items, but you can add new items.

To add one item to a set use the add() method.

Example

Add an item to a set, using the add() method:

```
thisset = {"anger", "fear", "sadness"}

thisset.add("happiness")

print(thisset)
```

```
{'anger', 'sadness', 'fear', 'happiness'}
```

Python - Add Set Items

Add Any Iterable

The object in the update() method does not have to be a set, it can be any iterable object (tuples, lists, dictionaries etc.).

Example

Add elements of a list to at set:

```
thisset = {"anger", "fear", "sadness"}
mylist = {"happiness", "surprise", "disgust"}

thisset.update(mylist)

print(thisset)
```

```
{'sadness', 'anger', 'disgust', 'surprise', 'fear', 'happiness'}
```

Python - Remove Set Items

Remove Item

To remove an item in a set, use the remove(), or the discard() method.

Example

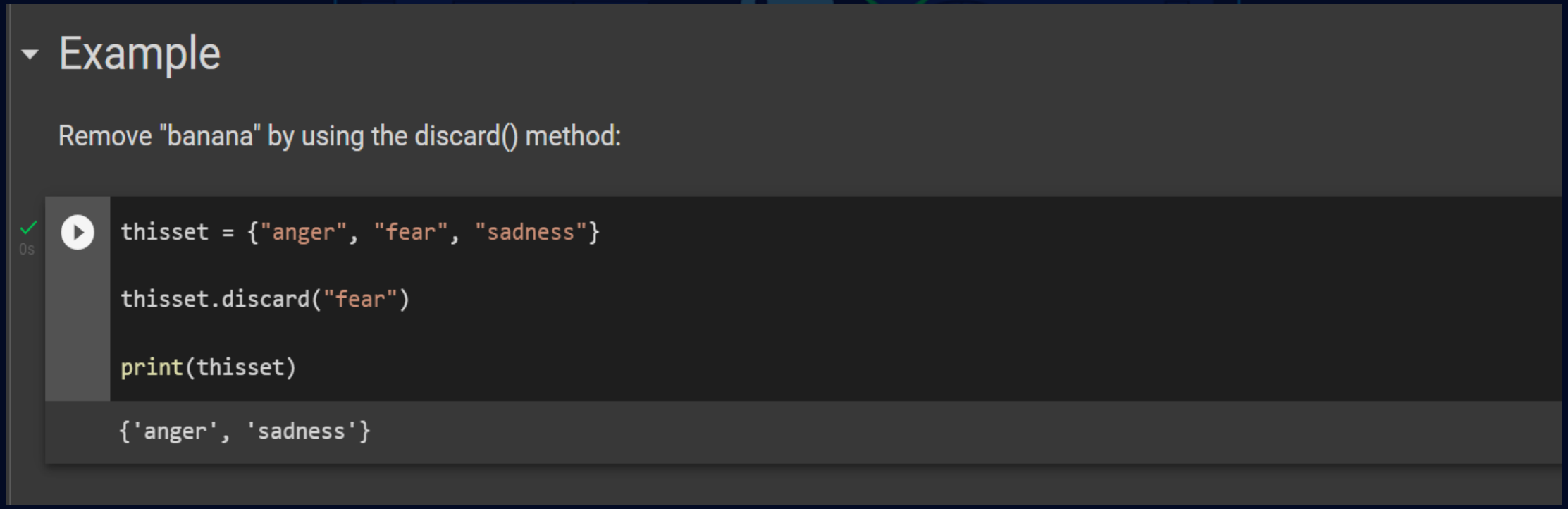
Remove "banana" by using the remove() method:

```
thisset = {"anger", "fear", "sadness"}

thisset.remove("fear")

print(thisset)
```

```
{'anger', 'sadness'}
```

Note: If the item to remove does not exist, remove() will raise an error.

Python - Remove Set Items



Note: If the item to remove does not exist, discard() will **NOT** raise an error.

Python - Remove Set Items

You can also use the pop() method to remove an item, but this method will remove the *last* item. Remember that sets are unordered, so you will not know what item that gets removed.

The return value of the pop() method is the removed item.

Example

Remove the last item by using the pop() method:

```
thisset = {"anger", "fear", "sadness"}

x = thisset.pop()

print(x)

print(thisset)
```

```
anger
{'sadness', 'fear'}
```

Note: Sets are *unordered*, so when using the pop() method, you do not know which item that gets removed.

Python - Remove Set Items

Example

The clear() method empties the set:

```
thisset = {"anger", "fear", "sadness"}

thisset.clear()

print(thisset)
```

```
set()
```

Python - Remove Set Items

Example

The del keyword will delete the set completely:

```
thisset = {"anger", "fear", "sadness"}

del thisset

print(thisset)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-34-84ecc7648a28> in <module>
      3 del thisset
      4
----> 5 print(thisset)

NameError: name 'thisset' is not defined
```

SEARCH STACK OVERFLOW

Python - Loop Sets

Loop Items

You can loop through the set items by using a for loop:

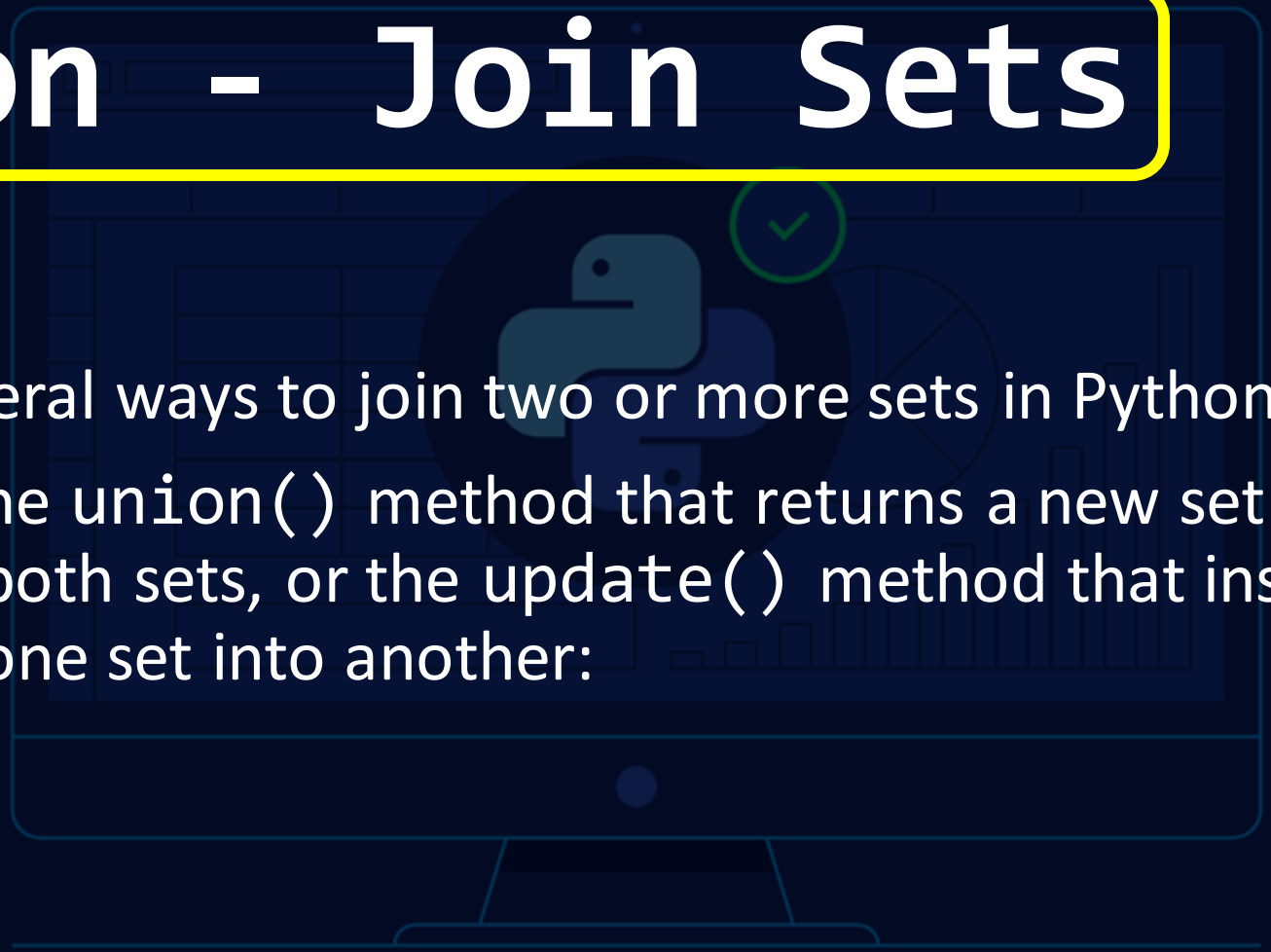
Example

Loop through the set, and print the values:

```
thisset = {"anger", "fear", "sadness"}

for x in thisset:
  print(x)
```

```
anger
sadness
fear
```

Python - Join Sets



Join Two Sets

There are several ways to join two or more sets in Python.

You can use the `union()` method that returns a new set containing all items from both sets, or the `update()` method that inserts all the items from one set into another:

Python - Join Sets

Join Two Sets

There are several ways to join two or more sets in Python.

You can use the union() method that returns a new set containing all items from both sets, or the update() method that inserts all the items from one set into another:

Example

The union() method returns a new set with all items from both sets:

```
set1 = {"a", "b" , "c"}
set2 = {1, 2, 3}

set3 = set1.union(set2)
print(set3)
```

```
{1, 2, 3, 'b', 'c', 'a'}
```

Python - Join Sets

Example

The update() method inserts the items in set2 into set1:

```
set1 = {"a", "b" , "c"}
set2 = {1, 2, 3}

set1.update(set2)
print(set1)
```

```
{1, 2, 3, 'b', 'c', 'a'}
```

Note: Both union() and update() will exclude any duplicate items.

Python - Join Sets

Keep ONLY the Duplicates

The intersection_update() method will keep only the items that are present in both sets.

Example

Keep the items that exist in both set x, and set y:

```
x = {"anger", "fear", "sadness"}
y = {"sadness", "guilt", "pain"}

x.intersection_update(y)

print(x)
```

```
{'sadness'}
```

Python - Join Sets

The intersection() method will return a *new* set, that only contains the items that are present in both sets.

Example

Return a set that contains the items that exist in both set x, and set y:

```
x = {"anger", "fear", "sadness"}
y = {"sadness", "guilt", "pain"}

z = x.intersection(y)

print(z)
```

```
{'sadness'}
```

Python - Join Sets

Keep All, But NOT the Duplicates

The symmetric_difference_update() method will keep only the elements that are NOT present in both sets.

Example

Keep the items that are not present in both sets:

```
x = {"anger", "fear", "sadness"}
y = {"sadness", "guilt", "pain"}

x.symmetric_difference_update(y)

print(x)
```

```
{'fear', 'guilt', 'anger', 'pain'}
```

Python - Join Sets

The symmetric_difference() method will return a new set, that contains only the elements that are NOT present in both sets.

Example

Return a set that contains all items from both sets, except items that are present in both:

```
x = {"anger", "fear", "sadness"}
y = {"sadness", "guilt", "pain"}

z = x.symmetric_difference(y)

print(z)
```

```
{'fear', 'guilt', 'anger', 'pain'}
```

Python - Set Methods

Set Methods

Python has a set of built-in methods that you can use on sets.

Method	Description
add()	Adds an element to the set
clear()	Removes all the elements from the set
copy()	Returns a copy of the set
difference()	Returns a set containing the difference between two or more sets

Python - Set Methods

Set Methods

Python has a set of built-in methods that you can use on sets.

difference_update()	Removes the items in this set that are also included in another, specified set
discard()	Remove the specified item
intersection()	Returns a set, that is the intersection of two other sets
intersection_update()	Removes the items in this set that are not present in other, specified set(s)
isdisjoint()	Returns whether two sets have a intersection or not

Python - Set Methods

Set Methods

Python has a set of built-in methods that you can use on sets.

issubset()	Returns whether another set contains this set or not
issuperset()	Returns whether this set contains another set or not
pop()	Removes an element from the set
remove()	Removes the specified element

Python - Set Methods

Set Methods

Python has a set of built-in methods that you can use on sets.

symmetric_difference()	Returns a set with the symmetric differences of two sets
symmetric_difference_update()	inserts the symmetric differences from this set and another
union()	Return a set containing the union of sets
update()	Update the set with the union of this set and others

Python Dictionaries

Dictionary

Dictionaries are used to store data values in key:value pairs.

A dictionary is a collection which is ordered*, changeable and do not allow duplicates.

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
```

Python Dictionaries

Example

Create and print a dictionary:

```
thisdict =  {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(thisdict)
```

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
```

Python Dictionaries

Dictionary Items

Dictionary items are ordered, changeable, and does not allow duplicates.

Dictionary items are presented in key:value pairs, and can be referred to by using the key name.

Example

Print the "brand" value of the dictionary:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(thisdict["brand"])
```

```
Ford
```

Python Dictionaries

Ordered or Unordered?

As of Python version 3.7, dictionaries are *ordered*. In Python 3.6 and earlier, dictionaries are *unordered*.

When we say that dictionaries are ordered, it means that the items have a defined order, and that order will not change.

Unordered means that the items does not have a defined order, you cannot refer to an item by using an index.

Changeable

Dictionaries are changeable, meaning that we can change, add or remove items after the dictionary has been created.

Python Dictionaries

Duplicates Not Allowed

Dictionaries cannot have two items with the same key:

Example

Duplicate values will overwrite existing values:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964,
  "year": 2020
}
print(thisdict)
```

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 2020}
```

Python Dictionaries

Dictionary Length

To determine how many items a dictionary has, use the len() function:

Example

Print the number of items in the dictionary:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964,
  "year": 2020
}
print(len(thisdict))
```

```
3
```

Python Dictionaries

Dictionary Items - Data Types

The values in dictionary items can be of any data type:

Example

String, int, boolean, and list data types:

```
thisdict = {
  "brand": "Ford",
  "electric": False,
  "year": 1964,
  "colors": ["red", "white", "blue"]
}

print(thisdict)
```

```
{'brand': 'Ford', 'electric': False, 'year': 1964, 'colors': ['red', 'white', 'blue']}
```

Python Dictionaries

type()

From Python's perspective, dictionaries are defined as objects with the data type 'dict':

```
<class 'dict'>
```

Example

Print the data type of a dictionary:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(type(thisdict))
```

```
<class 'dict'>
```

Python Dictionaries

The dict() Constructor

It is also possible to use the dict() constructor to make a dictionary.

Example

Using the dict() method to make a dictionary:

```
thisdict = dict(name = "John", age = 36, country = "Norway")

print(thisdict)
```

```
{'name': 'John', 'age': 36, 'country': 'Norway'}
```

Python - Access Dictionary Items

Accessing Items

You can access the items of a dictionary by referring to its key name, inside square brackets:

Example

Get the value of the "model" key:

```
thisdict =  {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
x = thisdict["model"]
print(x)
```

```
Mustang
```

Python - Access Dictionary Items

There is also a method called get() that will give you the same result:

Example

Get the value of the "model" key:

```
thisdict =  {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
x = thisdict.get("model")
print(x)
```

```
Mustang
```

Python - Access Dictionary Items

Get Keys

The keys() method will return a list of all the keys in the dictionary.

Example

Get a list of the keys:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}

x = thisdict.keys()

print(x)
```

```
dict_keys(['brand', 'model', 'year'])
```

Python - Access Dictionary Items

The list of the keys is a *view* of the dictionary, meaning that any changes done to the dictionary will be reflected in the keys list.

Example

Add a new item to the original dictionary, and see that the keys list gets updated as well:

```
car = {
"brand": "Ford",
"model": "Mustang",
"year": 1964
}

x = car.keys()

print(x) #before the change

car["color"] = "white"

print(x) #after the change
```

```
dict_keys(['brand', 'model', 'year'])
dict_keys(['brand', 'model', 'year', 'color'])
```

Python - Access Dictionary Items

Get Values

The values() method will return a list of all the values in the dictionary.

Example

Get a list of the values:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}

x = thisdict.values()

print(x)
```

```
dict_values(['Ford', 'Mustang', 1964])
```

Python - Access Dictionary Items

The list of the values is a *view* of the dictionary, meaning that any changes done to the dictionary will be reflected in the values list.

Example

Make a change in the original dictionary, and see that the values list gets updated as well:

```
car = {
"brand": "Ford",
"model": "Mustang",
"year": 1964
}

x = car.values()

print(x) #before the change

car["year"] = 2020

print(x) #after the change
```

```
dict_values(['Ford', 'Mustang', 1964])
dict_values(['Ford', 'Mustang', 2020])
```

Python - Access Dictionary Items

Example

Add a new item to the original dictionary, and see that the values list gets updated as well:

```
car = {
"brand": "Ford",
"model": "Mustang",
"year": 1964
}

x = car.values()

print(x) #before the change

car["color"] = "red"

print(x) #after the change
```

```
dict_values(['Ford', 'Mustang', 1964])
dict_values(['Ford', 'Mustang', 1964, 'red'])
```

Python - Access Dictionary Items

Get Items

The items() method will return each item in a dictionary, as tuples in a list.

Example

Get a list of the key:value pairs

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}

x = thisdict.items()

print(x)
```

```
dict_items([('brand', 'Ford'), ('model', 'Mustang'), ('year', 1964)])
```

Python - Access Dictionary Items

The returned list is a *view* of the items of the dictionary, meaning that any changes done to the dictionary will be reflected in the items list.

Example

Make a change in the original dictionary, and see that the items list gets updated as well:

```
car = {
"brand": "Ford",
"model": "Mustang",
"year": 1964
}

x = car.items()

print(x) #before the change

car["year"] = 2020

print(x) #after the change
```

```
dict_items([('brand', 'Ford'), ('model', 'Mustang'), ('year', 1964)])
dict_items([('brand', 'Ford'), ('model', 'Mustang'), ('year', 2020)])
```

Python - Access Dictionary Items

Example

Add a new item to the original dictionary, and see that the items list gets updated as well:

```
car = {
"brand": "Ford",
"model": "Mustang",
"year": 1964
}

x = car.items()

print(x) #before the change

car["color"] = "red"

print(x) #after the change
```

```
dict_items([('brand', 'Ford'), ('model', 'Mustang'), ('year', 1964)])
dict_items([('brand', 'Ford'), ('model', 'Mustang'), ('year', 1964), ('color', 'red')])
```

Python - Access Dictionary Items

Check if Key Exists

To determine if a specified key is present in a dictionary use the in keyword:

Example

Check if "model" is present in the dictionary:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
if "model" in thisdict:
  print("Yes, 'model' is one of the keys in the thisdict dictionary")
```

```
Yes, 'model' is one of the keys in the thisdict dictionary
```

Python - Change Dictionary Items

Change Values

You can change the value of a specific item by referring to its key name:

Example

Change the "year" to 2018:

```
thisdict =  {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}

thisdict["year"] = 2018

print(thisdict)
```

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 2018}
```

Python - Change Dictionary Items

Update Dictionary

The update() method will update the dictionary with the items from the given argument.

The argument must be a dictionary, or an iterable object with key:value pairs.

Example

Update the "year" of the car by using the update() method:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.update({"year": 2020})

print(thisdict)
```

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 2020}
```

Python - Add Dictionary Items

Adding Items

Adding an item to the dictionary is done by using a new index key and assigning a value to it:

Example

```
thisdict =  {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict["color"] = "red"
print(thisdict)
```

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964, 'color': 'red'}
```

Python - Add Dictionary Items

Update Dictionary

The update() method will update the dictionary with the items from a given argument. If the item does not exist, the item will be added.

The argument must be a dictionary, or an iterable object with key:value pairs.

Example

Add a color item to the dictionary by using the update() method:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.update({"color": "red"})

print(thisdict)
```

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964, 'color': 'red'}
```

Python - Remove Dictionary Items

Removing Items

There are several methods to remove items from a dictionary:

Example

The pop() method removes the item with the specified key name:

```
thisdict =  {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.pop("model")
print(thisdict)
```

```
{'brand': 'Ford', 'year': 1964}
```

Python - Remove Dictionary Items

Example

The popitem() method removes the last inserted item (in versions before 3.7, a random item is removed instead):

```
thisdict =  {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.popitem()
print(thisdict)
```

```
{'brand': 'Ford', 'model': 'Mustang'}
```

Python - Remove Dictionary Items

Example

The del keyword removes the item with the specified key name:

```
thisdict =  {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
del thisdict["model"]
print(thisdict)
```

```
{'brand': 'Ford', 'year': 1964}
```

Python - Remove Dictionary Items

Example

The del keyword can also delete the dictionary completely:

```
thisdict =  {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
del thisdict
print(thisdict)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-68-bde8ef6f368b> in <module>
      5 }
      6 del thisdict
----> 7 print(thisdict)

NameError: name 'thisdict' is not defined
```

SEARCH STACK OVERFLOW

Python - Remove Dictionary Items

Example

The clear() method empties the dictionary:

0s

```
thisdict =  {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.clear()
print(thisdict)
```

```
{}
```

Python - Loop Dictionary

Loop Through a Dictionary

You can loop through a dictionary by using a for loop.

When looping through a dictionary, the return value are the *keys* of the dictionary, but there are methods to return the *values* as well.

Example

Print all key names in the dictionary, one by one:

```
thisdict =  {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
for x in thisdict:
  print(x)
```

```
brand
model
year
```

Python - Loop Dictionary

Loop Through a Dictionary

You can loop through a dictionary by using a for loop.

When looping through a dictionary, the return value are the *keys* of the dictionary, but there are methods to return the *values* as well.

Example

Print all key names in the dictionary, one by one:

```
thisdict =  {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
for x in thisdict:
  print(x)
```

```
brand
model
year
```

Python - Loop Dictionary

Example

Print all values in the dictionary, one by one:

```
thisdict =  {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
for x in thisdict:
  print(thisdict[x])
```

```
Ford
Mustang
1964
```

Python - Loop Dictionary

Example

You can also use the values() method to return values of a dictionary:

```
thisdict =  {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
for x in thisdict.values():
  print(x)
```

```
Ford
Mustang
1964
```

Python - Loop Dictionary

Example

You can use the keys() method to return the keys of a dictionary:

```
thisdict =  {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
for x in thisdict.keys():
  print(x)
```

```
brand
model
year
```

Python - Loop Dictionary

Example

Loop through both keys and values, by using the items() method:

```
thisdict =  {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
for x, y in thisdict.items():
  print(x, y)
```

```
brand Ford
model Mustang
year 1964
```

Python - Copy Dictionaries

Copy a Dictionary

You cannot copy a dictionary simply by typing dict2 = dict1, because: dict2 will only be a *reference* to dict1, and changes made in dict1 will automatically also be made in dict2.

There are ways to make a copy, one way is to use the built-in Dictionary method copy().

Example

Make a copy of a dictionary with the copy() method:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
mydict = thisdict.copy()
print(mydict)
```

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
```

Python - Copy Dictionaries

Another way to make a copy is to use the built-in function dict().

Example

Make a copy of a dictionary with the dict() function:

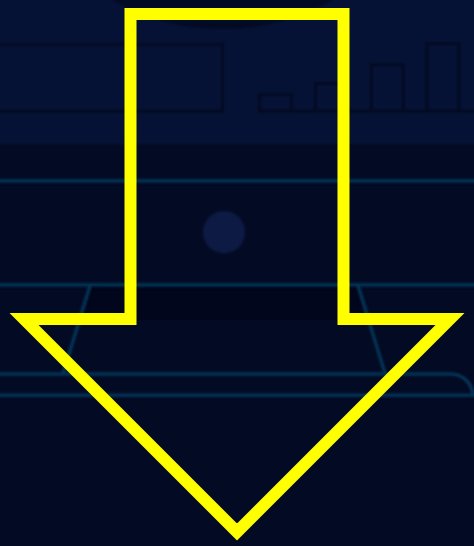
```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
mydict = dict(thisdict)
print(mydict)
```

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
```

Python - Nested Dictionaries

Nested Dictionaries

A dictionary can contain dictionaries, this is called nested dictionaries.



Python - Nested Dictionaries

Example

Create a dictionary that contain three dictionaries:

```
myfamily = {
  "child1" : {
    "name" : "Emil",
    "year" : 2004
  },
  "child2" : {
    "name" : "Tobias",
    "year" : 2007
  },
  "child3" : {
    "name" : "Linus",
    "year" : 2011
  }
}

print(myfamily)
```

```
{'child1': {'name': 'Emil', 'year': 2004}, 'child2': {'name': 'Tobias', 'year': 2007}, 'child3': {'name': 'Linus', 'year': 2011}}
```

Python - Nested Dictionaries

Or, if you want to add three dictionaries into a new dictionary:

Python - Nested Dictionaries

Example

Create three dictionaries, then create one dictionary that will contain the other three dictionaries:

```
child1 = {
  "name" : "Emil",
  "year" : 2004
}
child2 = {
  "name" : "Tobias",
  "year" : 2007
}
child3 = {
  "name" : "Linus",
  "year" : 2011
}

myfamily = {
  "child1" : child1,
  "child2" : child2,
  "child3" : child3
}

print(myfamily)
```

```
{'child1': {'name': 'Emil', 'year': 2004}, 'child2': {'name': 'Tobias', 'year': 2007}, 'child3': {'name': 'Linus', 'year': 2011}}
```

Python Dictionary Methods

Dictionary Methods

Python has a set of built-in methods that you can use on dictionaries.

Method	Description
clear()	Removes all the elements from the dictionary
copy()	Returns a copy of the dictionary
fromkeys()	Returns a dictionary with the specified keys and value
get()	Returns the value of the specified key
items()	Returns a list containing a tuple for each key value pair
keys()	Returns a list containing the dictionary's keys

Python Dictionary Methods

Dictionary Methods

Python has a set of built-in methods that you can use on dictionaries.

pop()	Removes the element with the specified key
popitem()	Removes the last inserted key-value pair
setdefault()	Returns the value of the specified key. If the key does not exist: insert the key, with the specified value
update()	Updates the dictionary with the specified key-value pairs
values()	Returns a list of all the values in the dictionary

THANK YOU!

CHINESE NAME: 喬萬斯

STUDENT ID: 4110E234

English Name: Joaquin Vasti R. Rapada

nickname: Wax

email: vastiplayer@gmail.com

GITHUB: https://github.com/4110E234/cs20220921/tree/main/python