**Module 1 - Python Basics**

1. Your first program
2. Types
3. Expressions and Variables
4. String Operations

**Module 2 - Python Data Structures**

1. Lists and Tuples
2. Sets
3. Dictionaries

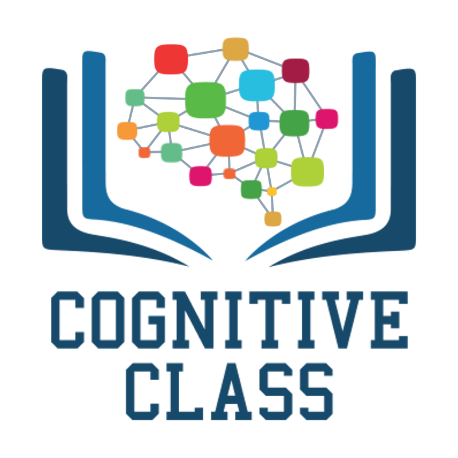
**Module 3 - Python Programming Fundamentals**

1. Conditions and Branching
2. Loops
3. Functions
4. Objects and Classes

**Module 4 - Working with Data in Python**

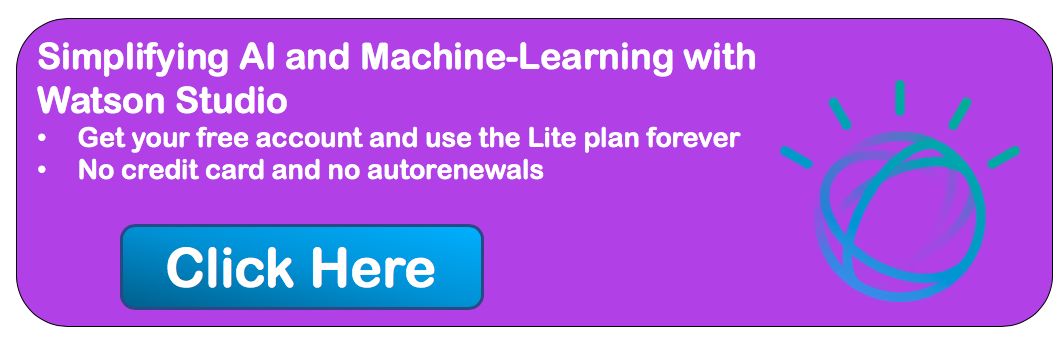
1. Reading files with open
2. Writing files with open
3. Loading data with Pandas
4. Working with and Saving data with Pandas
5. Numpy

**Module 5 - Final Project**

[[](https://cognitiveclass.ai/)](https://cognitiveclass.ai/)

**Python - Writing Your First Python Code!**

**Welcome!** This notebook will teach you the basics of the Python programming language. Although the information presented here is quite basic, it is an important foundation that will help you read and write Python code. By the end of this notebook, you'll know the basics of Python, including how to write basic commands, understand some basic types, and how to perform simple operations on them.

[](https://cocl.us/topNotebooksPython101Coursera)

**Table of Contents**

* [Say "Hello" to the world in Python](https://jupyterlab-3.labs.cognitiveclass.ai/user/ujwalsah/lab#hello)
  + [What version of Python are we using?](https://jupyterlab-3.labs.cognitiveclass.ai/user/ujwalsah/files/labs/PY0101EN/version?_xsrf=2%7Ce59ab274%7C5cd3669346f719901abec03916eb1636%7C1584939039)
  + [Writing comments in Python](https://jupyterlab-3.labs.cognitiveclass.ai/user/ujwalsah/files/labs/PY0101EN/comments?_xsrf=2%7Ce59ab274%7C5cd3669346f719901abec03916eb1636%7C1584939039)
  + [Errors in Python](https://jupyterlab-3.labs.cognitiveclass.ai/user/ujwalsah/files/labs/PY0101EN/errors?_xsrf=2%7Ce59ab274%7C5cd3669346f719901abec03916eb1636%7C1584939039)
  + [Does Python know about your error before it runs your code?](https://jupyterlab-3.labs.cognitiveclass.ai/user/ujwalsah/files/labs/PY0101EN/python_error?_xsrf=2%7Ce59ab274%7C5cd3669346f719901abec03916eb1636%7C1584939039)
  + [Exercise: Your First Program](https://jupyterlab-3.labs.cognitiveclass.ai/user/ujwalsah/files/labs/PY0101EN/exercise?_xsrf=2%7Ce59ab274%7C5cd3669346f719901abec03916eb1636%7C1584939039)
* [Types of objects in Python](https://jupyterlab-3.labs.cognitiveclass.ai/user/ujwalsah/lab#types_objects)
  + [Integers](https://jupyterlab-3.labs.cognitiveclass.ai/user/ujwalsah/files/labs/PY0101EN/int?_xsrf=2%7Ce59ab274%7C5cd3669346f719901abec03916eb1636%7C1584939039)
  + [Floats](https://jupyterlab-3.labs.cognitiveclass.ai/user/ujwalsah/files/labs/PY0101EN/float?_xsrf=2%7Ce59ab274%7C5cd3669346f719901abec03916eb1636%7C1584939039)
  + [Converting from one object type to a different object type](https://jupyterlab-3.labs.cognitiveclass.ai/user/ujwalsah/files/labs/PY0101EN/convert?_xsrf=2%7Ce59ab274%7C5cd3669346f719901abec03916eb1636%7C1584939039)
  + [Boolean data type](https://jupyterlab-3.labs.cognitiveclass.ai/user/ujwalsah/files/labs/PY0101EN/bool?_xsrf=2%7Ce59ab274%7C5cd3669346f719901abec03916eb1636%7C1584939039)
  + [Exercise: Types](https://jupyterlab-3.labs.cognitiveclass.ai/user/ujwalsah/files/labs/PY0101EN/exer_type?_xsrf=2%7Ce59ab274%7C5cd3669346f719901abec03916eb1636%7C1584939039)
* [Expressions and Variables](https://jupyterlab-3.labs.cognitiveclass.ai/user/ujwalsah/lab#expressions)
  + [Expressions](https://jupyterlab-3.labs.cognitiveclass.ai/user/ujwalsah/files/labs/PY0101EN/exp?_xsrf=2%7Ce59ab274%7C5cd3669346f719901abec03916eb1636%7C1584939039)
  + [Exercise: Expressions](https://jupyterlab-3.labs.cognitiveclass.ai/user/ujwalsah/files/labs/PY0101EN/exer_exp?_xsrf=2%7Ce59ab274%7C5cd3669346f719901abec03916eb1636%7C1584939039)
  + [Variables](https://jupyterlab-3.labs.cognitiveclass.ai/user/ujwalsah/files/labs/PY0101EN/var?_xsrf=2%7Ce59ab274%7C5cd3669346f719901abec03916eb1636%7C1584939039)
  + [Exercise: Expression and Variables in Python](https://jupyterlab-3.labs.cognitiveclass.ai/user/ujwalsah/files/labs/PY0101EN/exer_exp_var?_xsrf=2%7Ce59ab274%7C5cd3669346f719901abec03916eb1636%7C1584939039)

Estimated time needed: **25 min**

**Say "Hello" to the world in Python**

When learning a new programming language, it is customary to start with an "hello world" example. As simple as it is, this one line of code will ensure that we know how to print a string in output and how to execute code within cells in a notebook.

[Tip]: To execute the Python code in the code cell below, click on the cell to select it and press Shift + Enter.

[1]:



*# Try your first Python output*

​

print('Hello, Python!')

Hello, Python!

After executing the cell above, you should see that Python prints Hello, Python!. Congratulations on running your first Python code!

[Tip:] print() is a function. You passed the string 'Hello, Python!' as an argument to instruct Python on what to print.

**What version of Python are we using?**

There are two popular versions of the Python programming language in use today: Python 2 and Python 3. The Python community has decided to move on from Python 2 to Python 3, and many popular libraries have announced that they will no longer support Python 2.

Since Python 3 is the future, in this course we will be using it exclusively. How do we know that our notebook is executed by a Python 3 runtime? We can look in the top-right hand corner of this notebook and see "Python 3".

We can also ask directly Python and obtain a detailed answer. Try executing the following code:

[2]:



*# Check the Python Version*

​

**import** sys

print(sys.version)

3.6.7 | packaged by conda-forge | (default, Nov 6 2019, 16:19:42)

[GCC 7.3.0]

[Tip:] sys is a built-in module that contains many system-specific parameters and functions, including the Python version in use. Before using it, we must explictly import it.

**Writing comments in Python**

In addition to writing code, note that it's always a good idea to add comments to your code. It will help others understand what you were trying to accomplish (the reason why you wrote a given snippet of code). Not only does this help **other people** understand your code, it can also serve as a reminder **to you** when you come back to it weeks or months later.

To write comments in Python, use the number symbol # before writing your comment. When you run your code, Python will ignore everything past the # on a given line.

[3]:



*# Practice on writing comments*

​

print('Hello, Python!') *# This line prints a string*

*# print('Hi')*

Hello, Python!

After executing the cell above, you should notice that This line prints a string did not appear in the output, because it was a comment (and thus ignored by Python).

The second line was also not executed because print('Hi') was preceded by the number sign (#) as well! Since this isn't an explanatory comment from the programmer, but an actual line of code, we might say that the programmer *commented out* that second line of code.

**Errors in Python**

Everyone makes mistakes. For many types of mistakes, Python will tell you that you have made a mistake by giving you an error message. It is important to read error messages carefully to really understand where you made a mistake and how you may go about correcting it.

For example, if you spell print as frint, Python will display an error message. Give it a try:

[4]:



*# Print string as error message*

​

frint("Hello, Python!")

---------------------------------------------------------------------------

NameError Traceback (most recent call last)

<ipython-input-4-313a1769a8a5> in <module>

**1** # Print string as error message

**2**

----> 3 frint("Hello, Python!")

NameError: name 'frint' is not defined

The error message tells you:

1. where the error occurred (more useful in large notebook cells or scripts), and
2. what kind of error it was (NameError)

Here, Python attempted to run the function frint, but could not determine what frint is since it's not a built-in function and it has not been previously defined by us either.

You'll notice that if we make a different type of mistake, by forgetting to close the string, we'll obtain a different error (i.e., a SyntaxError). Try it below:

[5]:



*# Try to see build in error message*

​

print("Hello, Python!)

File "<ipython-input-5-63a21a726720>", line 3

print("Hello, Python!)

^

SyntaxError: EOL while scanning string literal

**Does Python know about your error before it runs your code?**

Python is what is called an *interpreted language*. Compiled languages examine your entire program at compile time, and are able to warn you about a whole class of errors prior to execution. In contrast, Python interprets your script line by line as it executes it. Python will stop executing the entire program when it encounters an error (unless the error is expected and handled by the programmer, a more advanced subject that we'll cover later on in this course).

Try to run the code in the cell below and see what happens:

[6]:



*# Print string and error to see the running order*

​

print("This will be printed")

frint("This will cause an error")

print("This will NOT be printed")

This will be printed

---------------------------------------------------------------------------

NameError Traceback (most recent call last)

<ipython-input-6-af59af1b345d> in <module>

**2**

**3** print("This will be printed")

----> 4 frint("This will cause an error")

**5** print("This will NOT be printed")

NameError: name 'frint' is not defined

**Exercise: Your First Program**

Generations of programmers have started their coding careers by simply printing "Hello, world!". You will be following in their footsteps.

In the code cell below, use the print() function to print out the phrase: Hello, world!

[7]:



*# Write your code below and press Shift+Enter to execute*

print('Hello,world!')

Hello,world!

Double-click **here** for the solution.

Now, let's enhance your code with a comment. In the code cell below, print out the phrase: Hello, world! and comment it with the phrase Print the traditional hello world all in one line of code.

[9]:



*# Write your code below and press Shift+Enter to execute*

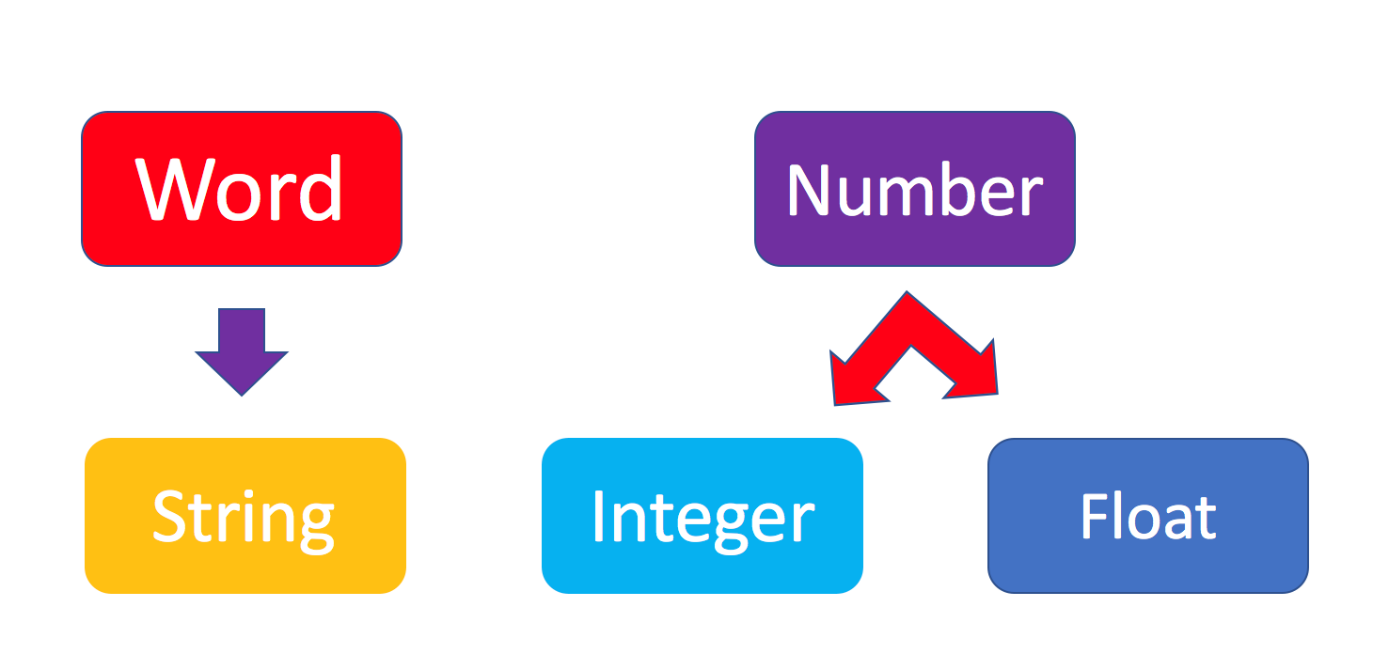
print('Hello, World!') *#Print the traditional hello world*

Hello, World!

Double-click **here** for the solution.

**Types of objects in Python**

Python is an object-oriented language. There are many different types of objects in Python. Let's start with the most common object types: *strings*, *integers* and *floats*. Anytime you write words (text) in Python, you're using *character strings* (strings for short). The most common numbers, on the other hand, are *integers* (e.g. -1, 0, 100) and *floats*, which represent real numbers (e.g. 3.14, -42.0).



The following code cells contain some examples.

[10]:



*# Integer*

​

11

[10]:

11

[11]:



*# Float*

​

2.14

[11]:

2.14

[12]:



*# String*

​

"Hello, Python 101!"

[12]:

'Hello, Python 101!'

You can get Python to tell you the type of an expression by using the built-in type() function. You'll notice that Python refers to integers as int, floats as float, and character strings as str.

[13]:



*# Type of 12*

​

type(12)

[13]:

int

[14]:



*# Type of 2.14*

​

type(2.14)

[14]:

float

[15]:



*# Type of "Hello, Python 101!"*

​

type("Hello, Python 101!")

[15]:

str

In the code cell below, use the type() function to check the object type of 12.0.

[17]:



*# Write your code below. Don't forget to press Shift+Enter to execute the cell*

type(12.0)

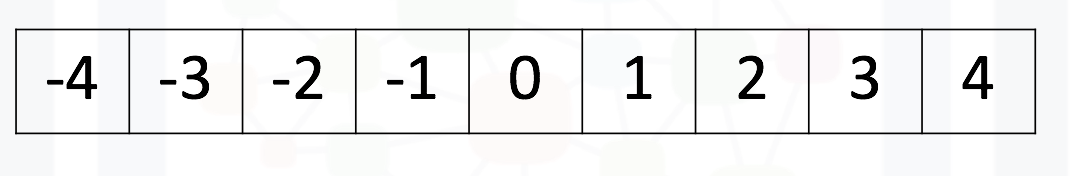
[17]:

float

Double-click **here** for the solution.

**Integers**

Here are some examples of integers. Integers can be negative or positive numbers:



We can verify this is the case by using, you guessed it, the type() function:

[18]:



*# Print the type of -1*

​

type(**-**1)

[18]:

int

[19]:



*# Print the type of 4*

​

type(4)

[19]:

int

[20]:



*# Print the type of 0*

​

type(0)

[20]:

int

**Floats**

Floats represent real numbers; they are a superset of integer numbers but also include "numbers with decimals". There are some limitations when it comes to machines representing real numbers, but floating point numbers are a good representation in most cases. You can learn more about the specifics of floats for your runtime environment, by checking the value of sys.float\_info. This will also tell you what's the largest and smallest number that can be represented with them.

Once again, can test some examples with the type() function:

[23]:



*# Print the type of 1.0*

​

type(1.0) *# Notice that 1 is an int, and 1.0 is a float*

[23]:

float

[24]:



*# Print the type of 0.5*

​

type(0.5)

[24]:

float

[25]:



*# Print the type of 0.56*

​

type(0.56)

[25]:

float

[26]:



*# System settings about float type*

​

sys.float\_info

[26]:

sys.float\_info(max=1.7976931348623157e+308, max\_exp=1024, max\_10\_exp=308, min=2.2250738585072014e-308, min\_exp=-1021, min\_10\_exp=-307, dig=15, mant\_dig=53, epsilon=2.220446049250313e-16, radix=2, rounds=1)

**Converting from one object type to a different object type**

You can change the type of the object in Python; this is called typecasting. For example, you can convert an *integer* into a *float* (e.g. 2 to 2.0).

Let's try it:

[27]:



*# Verify that this is an integer*

​

type(2)

[27]:

int

**Converting integers to floats**

Let's cast integer 2 to float:

[28]:



*# Convert 2 to a float*

​

float(2)

[28]:

2.0

[29]:



*# Convert integer 2 to a float and check its type*

​

type(float(2))

[29]:

float

When we convert an integer into a float, we don't really change the value (i.e., the significand) of the number. However, if we cast a float into an integer, we could potentially lose some information. For example, if we cast the float 1.1 to integer we will get 1 and lose the decimal information (i.e., 0.1):

[30]:



*# Casting 1.1 to integer will result in loss of information*

​

int(1.1)

[30]:

1

**Converting from strings to integers or floats**

Sometimes, we can have a string that contains a number within it. If this is the case, we can cast that string that represents a number into an integer using int():

[31]:



*# Convert a string into an integer*

​

int('1')

[31]:

1

But if you try to do so with a string that is not a perfect match for a number, you'll get an error. Try the following:

[32]:



*# Convert a string into an integer with error*

​

int('1 or 2 people')

---------------------------------------------------------------------------

ValueError Traceback (most recent call last)

<ipython-input-32-b78145d165c7> in <module>

**1** # Convert a string into an integer with error

**2**

----> 3 int('1 or 2 people')

ValueError: invalid literal for int() with base 10: '1 or 2 people'

You can also convert strings containing floating point numbers into *float* objects:

[33]:



*# Convert the string "1.2" into a float*

​

float('1.2')

[33]:

1.2

[Tip:] Note that strings can be represented with single quotes ('1.2') or double quotes ("1.2"), but you can't mix both (e.g., "1.2').

**Did you know? IBM Watson Studio lets you build and deploy an AI solution, using the best of open source and IBM software and giving your team a single environment to work in.**[**Learn more here.**](https://cocl.us/ibm_watson_studio_infobox)

**Converting numbers to strings**

If we can convert strings to numbers, it is only natural to assume that we can convert numbers to strings, right?

[34]:



*# Convert an integer to a string*

​

str(1)

[34]:

'1'

And there is no reason why we shouldn't be able to make floats into strings as well:

[35]:



*# Convert a float to a string*

​

str(1.2)

[35]:

'1.2'

**Boolean data type**

*Boolean* is another important type in Python. An object of type *Boolean* can take on one of two values: True or False:

[36]:



*# Value true*

​

**True**

[36]:

True

Notice that the value True has an uppercase "T". The same is true for False (i.e. you must use the uppercase "F").

[37]:



*# Value false*

​

**False**

[37]:

False

When you ask Python to display the type of a boolean object it will show bool which stands for *boolean*:

[38]:



*# Type of True*

​

type(**True**)

[38]:

bool

[39]:



*# Type of False*

​

type(**False**)

[39]:

bool

We can cast boolean objects to other data types. If we cast a boolean with a value of True to an integer or float we will get a one. If we cast a boolean with a value of False to an integer or float we will get a zero. Similarly, if we cast a 1 to a Boolean, you get a True. And if we cast a 0 to a Boolean we will get a False. Let's give it a try:

[41]:



*# Convert True to int*

​

int(**True**)

[41]:

1

[42]:



*# Convert 1 to boolean*

​

bool(1)

[42]:

True

[43]:



*# Convert 0 to boolean*

​

bool(0)

[43]:

False

[44]:



*# Convert True to float*

​

float(**True**)

[44]:

1.0

**Exercise: Types**

What is the data type of the result of: 6 / 2?

[45]:



*# Write your code below. Don't forget to press Shift+Enter to execute the cell*

type(6**/**2)

[45]:

float

Double-click **here** for the solution.

What is the type of the result of: 6 // 2? (Note the double slash //.)

[46]:



*# Write your code below. Don't forget to press Shift+Enter to execute the cell*

type(6**//**2)

[46]:

int

Double-click **here** for the solution.

**Expression and Variables**

**Expressions**

Expressions in Python can include operations among compatible types (e.g., integers and floats). For example, basic arithmetic operations like adding multiple numbers:

[49]:



*# Addition operation expression*

​

43 **+** 60 **+** 16 **+** 41

[49]:

160

We can perform subtraction operations using the minus operator. In this case the result is a negative number:

[50]:



*# Subtraction operation expression*

​

50 **-** 60

[50]:

-10

We can do multiplication using an asterisk:

[51]:



*# Multiplication operation expression*

​

5 **\*** 5

[51]:

25

We can also perform division with the forward slash:

[52]:



*# Division operation expression*

​

25 **/** 5

[52]:

5.0

[53]:



*# Division operation expression*

​

25 **/** 6

[53]:

4.166666666666667

As seen in the quiz above, we can use the double slash for integer division, where the result is rounded to the nearest integer:

[54]:



*# Integer division operation expression*

​

25 **//** 5

[54]:

5

[55]:



*# Integer division operation expression*

​

25 **//** 6

[55]:

4

**Exercise: Expression**

Let's write an expression that calculates how many hours there are in 160 minutes:

[57]:



*# Write your code below. Don't forget to press Shift+Enter to execute the cell*

print(160**/**60)

2.6666666666666665

Double-click **here** for the solution.

Python follows well accepted mathematical conventions when evaluating mathematical expressions. In the following example, Python adds 30 to the result of the multiplication (i.e., 120).

[58]:



*# Mathematical expression*

​

30 **+** 2 **\*** 60

[58]:

150

And just like mathematics, expressions enclosed in parentheses have priority. So the following multiplies 32 by 60.

[59]:



*# Mathematical expression*

​

(30 **+** 2) **\*** 60

[59]:

1920

**Variables**

Just like with most programming languages, we can store values in *variables*, so we can use them later on. For example:

[61]:



*# Store value into variable*

​

x **=** 43 **+** 60 **+** 16 **+** 41

To see the value of x in a Notebook, we can simply place it on the last line of a cell:

[62]:



*# Print out the value in variable*

​

x

[62]:

160

We can also perform operations on x and save the result to a new variable:

[63]:



*# Use another variable to store the result of the operation between variable and value*

​

y **=** x **/** 60

y

[63]:

2.6666666666666665

If we save a value to an existing variable, the new value will overwrite the previous value:

[64]:



*# Overwrite variable with new value*

​

x **=** x **/** 60

x

[64]:

2.6666666666666665

It's a good practice to use meaningful variable names, so you and others can read the code and understand it more easily:

[65]:



*# Name the variables meaningfully*

​

total\_min **=** 43 **+** 42 **+** 57 *# Total length of albums in minutes*

total\_min

[65]:

142

[66]:



*# Name the variables meaningfully*

​

total\_hours **=** total\_min **/** 60 *# Total length of albums in hours*

total\_hours

[66]:

2.3666666666666667

In the cells above we added the length of three albums in minutes and stored it in total\_min. We then divided it by 60 to calculate total length total\_hours in hours. You can also do it all at once in a single expression, as long as you use parenthesis to add the albums length before you divide, as shown below.

[67]:



*# Complicate expression*

​

total\_hours **=** (43 **+** 42 **+** 57) **/** 60 *# Total hours in a single expression*

total\_hours

[67]:

2.3666666666666667

If you'd rather have total hours as an integer, you can of course replace the floating point division with integer division (i.e., //).

**Exercise: Expression and Variables in Python**

What is the value of x where x = 3 + 2 \* 2

[68]:



*# Write your code below. Don't forget to press Shift+Enter to execute the cell*

x **=** 3 **+** 2 **\*** 2

x

[68]:

7

Double-click **here** for the solution.

What is the value of y where y = (3 + 2) \* 2?

[69]:



*# Write your code below. Don't forget to press Shift+Enter to execute the cell*

y **=** (3 **+** 2) **\*** 2

y

[69]:

10

Double-click **here** for the solution.

What is the value of z where z = x + y?

[70]:



*# Write your code below. Don't forget to press Shift+Enter to execute the cell*

z **=** x **+** y

z

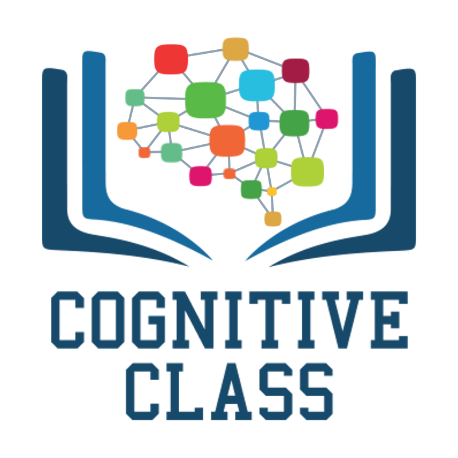
[70]:

17

Double-click **here** for the solution.

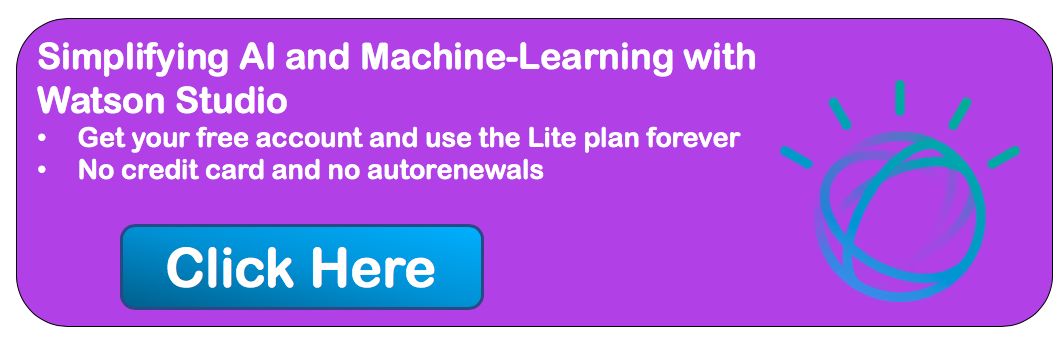
**The last exercise!**

Congratulations, you have completed your first lesson and hands-on lab in Python. However, there is one more thing you need to do. The Data Science community encourages sharing work. The best way to share and showcase your work is to share it on GitHub. By sharing your notebook on GitHub you are not only building your reputation with fellow data scientists, but you can also show it off when applying for a job. Even though this was your first piece of work, it is never too early to start building good habits. So, please read and follow [this article](https://cognitiveclass.ai/blog/data-scientists-stand-out-by-sharing-your-notebooks/) to learn how to share your work.

[[](https://cognitiveclass.ai/)](https://cognitiveclass.ai/)

# String Operations

**Welcome!** This notebook will teach you about the string operations in the Python Programming Language. By the end of this notebook, you'll know the basics string operations in Python, including indexing, escape sequences and operations.

[](https://cocl.us/topNotebooksPython101Coursera)

## Table of Contents

* [What are Strings?](https://jupyterlab-1.labs.cognitiveclass.ai/user/ujwalsah/lab#strings)
* [Indexing](https://jupyterlab-1.labs.cognitiveclass.ai/user/ujwalsah/lab#index)
  + [Negative Indexing](https://jupyterlab-1.labs.cognitiveclass.ai/user/ujwalsah/files/labs/PY0101EN/neg?_xsrf=2%7C31e12ccb%7C220a46492f67b6371523c0f65c6c7d5a%7C1585023330)
  + [Slicing](https://jupyterlab-1.labs.cognitiveclass.ai/user/ujwalsah/files/labs/PY0101EN/slice?_xsrf=2%7C31e12ccb%7C220a46492f67b6371523c0f65c6c7d5a%7C1585023330)
  + [Stride](https://jupyterlab-1.labs.cognitiveclass.ai/user/ujwalsah/files/labs/PY0101EN/stride?_xsrf=2%7C31e12ccb%7C220a46492f67b6371523c0f65c6c7d5a%7C1585023330)
  + [Concatenate Strings](https://jupyterlab-1.labs.cognitiveclass.ai/user/ujwalsah/files/labs/PY0101EN/concat?_xsrf=2%7C31e12ccb%7C220a46492f67b6371523c0f65c6c7d5a%7C1585023330)
* [Escape Sequences](https://jupyterlab-1.labs.cognitiveclass.ai/user/ujwalsah/lab#escape)
* [String Operations](https://jupyterlab-1.labs.cognitiveclass.ai/user/ujwalsah/lab#operations)
* [Quiz on Strings](https://jupyterlab-1.labs.cognitiveclass.ai/user/ujwalsah/lab#quiz)

Estimated time needed: **15 min**

## What are Strings?

The following example shows a string contained within 2 quotation marks:

[1]:



*# Use quotation marks for defining string*

​

"Michael Jackson"

[1]:

'Michael Jackson'

We can also use single quotation marks:

[2]:



*# Use single quotation marks for defining string*

​

'Michael Jackson'

[2]:

'Michael Jackson'

A string can be a combination of spaces and digits:

[3]:



*# Digitals and spaces in string*

​

'1 2 3 4 5 6 '

[3]:

'1 2 3 4 5 6 '

A string can also be a combination of special characters :

[6]:



*# Special characters in string*

​

'@#2\_#]&\*^%$'

[6]:

'@#2\_#]&\*^%$'

We can print our string using the print statement:

[7]:



*# Print the string*

​

print("hello!")

hello!

We can bind or assign a string to another variable:

[10]:



*# Assign string to variable*

​

Name **=** "Michael Jackson"

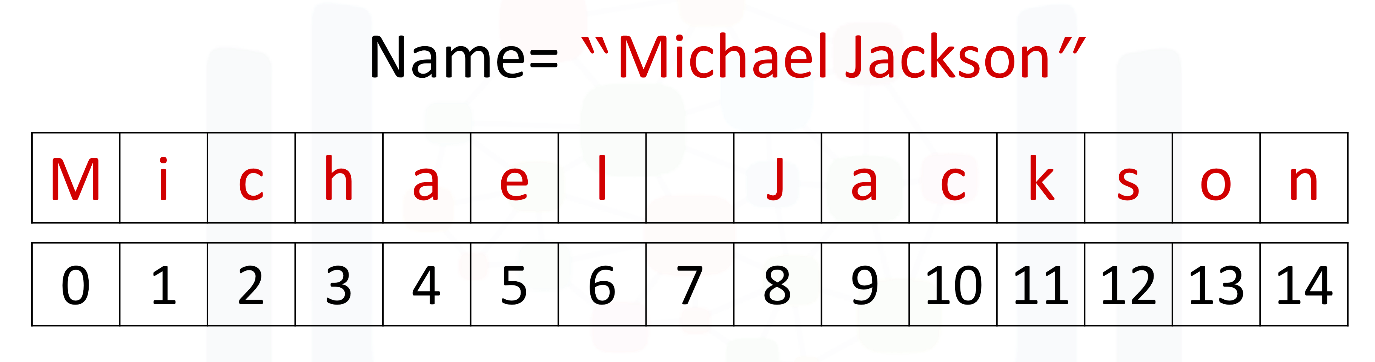
Name

[10]:

'Michael Jackson'

## Indexing

It is helpful to think of a string as an ordered sequence. Each element in the sequence can be accessed using an index represented by the array of numbers:



The first index can be accessed as follows:

[Tip]: Because indexing starts at 0, it means the first index is on the index 0.

[11]:



*# Print the first element in the string*

​

print(Name[0])

M

We can access index 6:

[14]:



*# Print the element on index 6 in the string*

​

print(Name[6])

l

Moreover, we can access the 13th index:

[15]:



*# Print the element on the 13th index in the string*

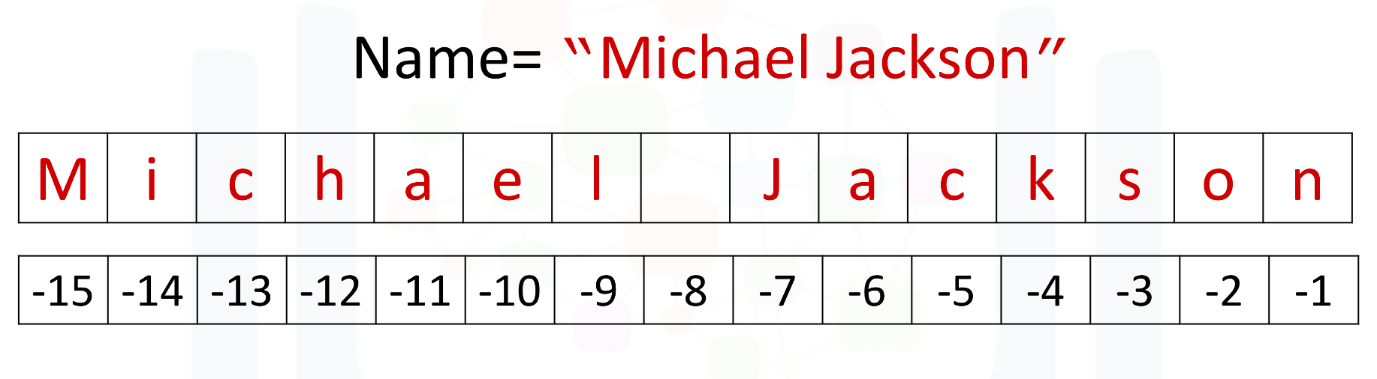
​

print(Name[13])

o

### Negative Indexing

We can also use negative indexing with strings:



Negative index can help us to count the element from the end of the string.

The last element is given by the index -1:

[16]:



*# Print the last element in the string*

​

print(Name[**-**1])

n

The first element can be obtained by index -15:

[17]:



*# Print the first element in the string*

​

print(Name[**-**15])

M

We can find the number of characters in a string by using len, short for length:

[18]:



*# Find the length of string*

​

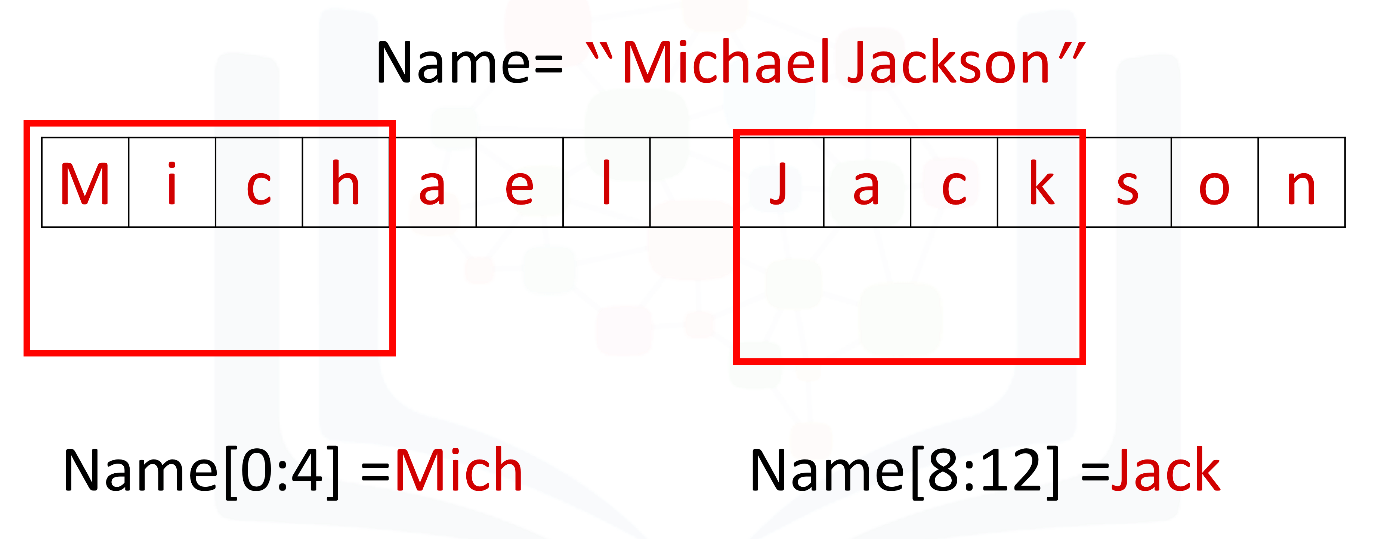
len("Michael Jackson")

[18]:

15

### Slicing

We can obtain multiple characters from a string using slicing, we can obtain the 0 to 4th and 8th to the 12th element:



[Tip]: When taking the slice, the first number means the index (start at 0), and the second number means the length from the index to the last element you want (start at 1)

[2]:



*# Take the slice on variable Name with only index 0 to index 3*

Name **=** 'Michael Jackson'

Name[0:4]

[2]:

'Mich'

[3]:



*# Take the slice on variable Name with only index 8 to index 11*

​

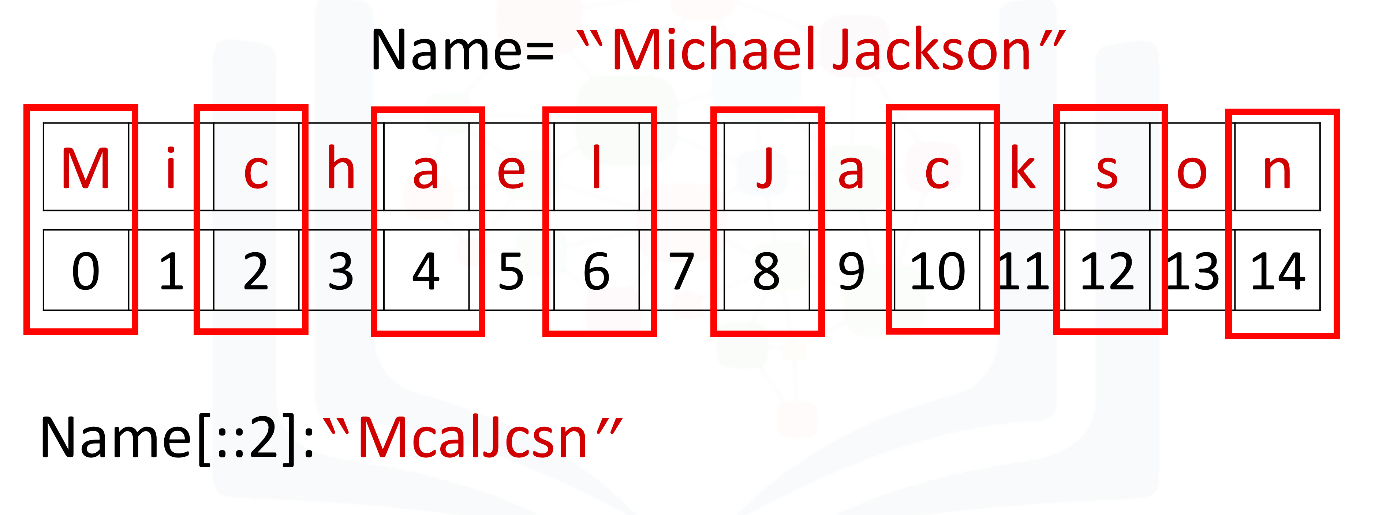
Name[8:12]

[3]:

'Jack'

### Stride

We can also input a stride value as follows, with the '2' indicating that we are selecting every second variable:



[4]:



*# Get every second element. The elments on index 1, 3, 5 ...*

​

Name[::2]

[4]:

'McalJcsn'

We can also incorporate slicing with the stride. In this case, we select the first five elements and then use the stride:

[5]:



*# Get every second element in the range from index 0 to index 4*

​

Name[0:5:2]

[5]:

'Mca'

### Concatenate Strings

We can concatenate or combine strings by using the addition symbols, and the result is a new string that is a combination of both:

[6]:



*# Concatenate two strings*

​

Statement **=** Name **+** "is the best"

Statement

[6]:

'Michael Jacksonis the best'

To replicate values of a string we simply multiply the string by the number of times we would like to replicate it. In this case, the number is three. The result is a new string, and this new string consists of three copies of the original string:

[7]:



*# Print the string for 3 times*

​

3 **\*** "Michael Jackson"

[7]:

'Michael JacksonMichael JacksonMichael Jackson'

You can create a new string by setting it to the original variable. Concatenated with a new string, the result is a new string that changes from Michael Jackson to “Michael Jackson is the best".

[8]:



*# Concatenate strings*

​

Name **=** "Michael Jackson"

Name **=** Name **+** " is the best"

Name

[8]:

'Michael Jackson is the best'

## Escape Sequences

Back slashes represent the beginning of escape sequences. Escape sequences represent strings that may be difficult to input. For example, back slash "n" represents a new line. The output is given by a new line after the back slash "n" is encountered:

[9]:



*# New line escape sequence*

​

print(" Michael Jackson \n is the best" )

Michael Jackson

is the best

Similarly, back slash "t" represents a tab:

[10]:



*# Tab escape sequence*

​

print(" Michael Jackson \t is the best" )

Michael Jackson is the best

If you want to place a back slash in your string, use a double back slash:

[45]:



*# Include back slash in string*

​

print(" Michael Jackson \\ is the best" )

Michael Jackson \ is the best

We can also place an "r" before the string to display the backslash:

[12]:



*# r will tell python that string will be display as raw string*

​

print(r" Michael Jackson \ is the best" )

Michael Jackson \ is the best

### ****Did you know?**** IBM Watson Studio lets you build and deploy an AI solution, using the best of open source and IBM software and giving your team a single environment to work in. [Learn more here.](https://cocl.us/ibm_watson_studio_infobox)

## String Operations

There are many string operation methods in Python that can be used to manipulate the data. We are going to use some basic string operations on the data.

Let's try with the method upper; this method converts lower case characters to upper case characters:

[15]:



*# Convert all the characters in string to upper case*

​

A **=** "Thriller is the sixth studio album"

print("before upper:", A)

B **=** A.upper()

print("After upper:", B)

before upper: Thriller is the sixth studio album

After upper: THRILLER IS THE SIXTH STUDIO ALBUM

The method replace replaces a segment of the string, i.e. a substring with a new string. We input the part of the string we would like to change. The second argument is what we would like to exchange the segment with, and the result is a new string with the segment changed:

[18]:



*# Replace the old substring with the new target substring is the segment has been found in the string*

​

A **=** "Michael Jackson is the best"

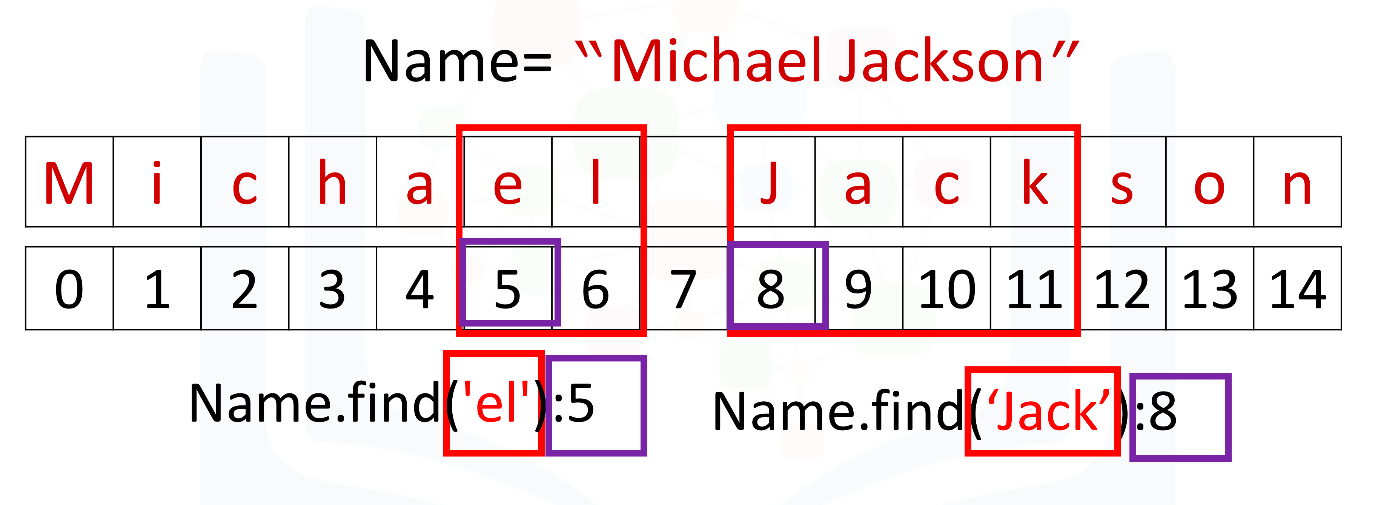
B **=** A.replace('Michael', 'Janet')

B

[18]:

'Janet Jackson is the best'

The method find finds a sub-string. The argument is the substring you would like to find, and the output is the first index of the sequence. We can find the sub-string jack or el.



[19]:



*# Find the substring in the string. Only the index of the first elment of substring in string will be the output*

​

Name **=** "Michael Jackson"

Name.find('el')

[19]:

5

[20]:



*# Find the substring in the string.*

​

Name.find('Jack')

[20]:

8

If the sub-string is not in the string then the output is a negative one. For example, the string 'Jasdfasdasdf' is not a substring:

[21]:



*# If cannot find the substring in the string*

​

Name.find('Jasdfasdasdf')

[21]:

-1

## Quiz on Strings

What is the value of the variable A after the following code is executed?

[26]:



*# Write your code below and press Shift+Enter to execute*

​

A **=** "1"

A

[26]:

'1'

Double-click **here** for the solution.

What is the value of the variable B after the following code is executed?

[27]:



*# Write your code below and press Shift+Enter to execute*

​

B **=** "2"

B

[27]:

'2'

Double-click **here** for the solution.

What is the value of the variable C after the following code is executed?

[28]:



*# Write your code below and press Shift+Enter to execute*

​

C **=** A **+** B

C

[28]:

'12'

Double-click **here** for the solution.

Consider the variable D use slicing to print out the first three elements:

[31]:



*# Write your code below and press Shift+Enter to execute*

​

D **=** "ABCDEFG"

D[0:3]

[31]:

'ABC'

Double-click **here** for the solution.

Use a stride value of 2 to print out every second character of the string E:

[32]:



*# Write your code below and press Shift+Enter to execute*

​

E **=** 'clocrkr1e1c1t'

E[::2]

[32]:

'correct'

Double-click **here** for the solution.

Print out a backslash:

[57]:



*# Write your code below and press Shift+Enter to execute*

​

print(r" \ ")

print("\\")

\

\

Double-click **here** for the solution.

Convert the variable F to uppercase:

[59]:



*# Write your code below and press Shift+Enter to execute*

​

F **=** "You are wrong"

F.upper()

[59]:

'YOU ARE WRONG'

Double-click **here** for the solution.

Consider the variable G, and find the first index of the sub-string snow:

[60]:



*# Write your code below and press Shift+Enter to execute*

​

G **=** "Mary had a little lamb Little lamb, little lamb Mary had a little lamb \

Its fleece was white as snow And everywhere that Mary went Mary went, Mary went \

Everywhere that Mary went The lamb was sure to go"

G.find('snow')

[60]:

95

Double-click **here** for the solution.

In the variable G, replace the sub-string Mary with Bob:

[61]:



*# Write your code below and press Shift+Enter to execute*

G.replace('Mary','Bob')

[61]:

'Bob had a little lamb Little lamb, little lamb Bob had a little lamb Its fleece was white as snow And everywhere that Bob went Bob went, Bob went Everywhere that Bob went The lamb was sure to go'

Double-click **here** for the solution.

## The last exercise!

Congratulations, you have completed your first lesson and hands-on lab in Python. However, there is one more thing you need to do. The Data Science community encourages sharing work. The best way to share and showcase your work is to share it on GitHub. By sharing your notebook on GitHub you are not only building your reputation with fellow data scientists, but you can also show it off when applying for a job. Even though this was your first piece of work, it is never too early to start building good habits. So, please read and follow [this article](https://cognitiveclass.ai/blog/data-scientists-stand-out-by-sharing-your-notebooks/) to learn how to share your work.