**Computational Thinking**

Computational thinking, or CT, can be broken down in two ways. First, CT involves thinking about how computers solve problems. Second, CT involves thinking about how you can transform a problem you have into one that a computer can solve.

On a high-level, computers typically solve problems with three main tools: functions, variables, and logic. These three things make up the vast majority of Python programs. Python is also strongly-typed, which means types matter a lot in Python programs.

We can transform a problem into one that a computer could solve by performing a four-stage process called ADTO: Abstraction, Decomposition, Translation, and Optimization.

* Abstraction is the stage in which you identify what information is relevant to solving our problem.
* Decomposition is where you break down the relevant information you've identified into distinct steps.
* Translation is where you turn those distinct steps into steps written in Python code.
* Optimization is where you make sure that the sequence of Python steps isn’t too complicated, long, or expensive to run.

You’ll learn about each of these four steps in this subunit.

omputational thinking is super useful to data scientists. Firstly, data scientists can only use computers to solve problems and implement ideas if they know how computers will execute this work. Secondly, data scientists want to build models to solve real-world problems. This requires transforming a problem into one that a computer could solve (by running algorithms).

To achieve the best results, data scientists need to know how to abstract from the problem at hand, decompose a solution into distinct steps, translate those steps into Python, and optimize the sequence of steps. This article will teach you how to do all that and more!

DSC Prep 2.3 Project. A little History

##### A little history

A cipher is defined in the Cambridge Dictionary as 'a system of writing that prevents most people from understanding the message'. A cipher is basically a method of transforming a message to conceal its meaning.

All ciphers involve either **rearrangement** or **substitution**, or both: the former just takes the initial symbols and puts them in a different order, and the latter substitutes those symbols for others without changing the order of the initial symbols. Regardless of the system, the changes are (in theory) understood only by the sender and the receiver of the message. The first cipher device was used by the ancient Greeks around 400BC to facilitate military communication.

The Caesar Cipher is named after Julius Caesar, who also used it for military purposes. The Roman historian Suetonius writes: "If he had anything confidential to say, he wrote it in cipher, that is, by so changing the order of the letters of the alphabet, that not a word could be made out. If anyone wishes to decipher these, and get at their meaning, he must substitute the fourth letter of the alphabet, namely D, for A, and so with the others." - Suetonius, Life of Julius Caesar 56 .

Essentially, what the Caesar Cipher does is take a message, comprising a sequence of characters, and a shift, comprising a number, and shifts each of those characters forward by that number. For example:

[Diagram

Description automatically generated](https://cdn.filestackcontent.com/V7M1kpIsQLOwnOhbfWhH)

So in this example, each character in the initial message is shifted by 2 characters. Notice that the initial message is already encoded: multiple ciphers can be used in series to make a larger, more powerful cipher!

***Caesar Cipher Project***

In this project, you'll try your hand at writing your first ever **cyber security program**! While the code you need to write isn't long, it'll be fairly challenging; the aim is to get as far as you can with it. Wrestling with it will really improve your coding and data manipulation skills, and will make you a much better data scientist. If you finish the task, think about how you can make your solution more efficient, elegant or readable. Above all, have fun applying all the things you've learned so far!

This project will have you applying the following skills and concepts:

* function definition
* control-flow statements like if-statements
* for loops
* list manipulation
* basic string manipulation
* type conversions
* basic maths in Python
* independent Google research and debugging skills.

DSC Prep 2.3 Project. A little Theory

##### A little theory

To implement this function in Python, you'll need an introduction to the way Python, and programming languages more widely, already encode characters.

In Python, every character - from English letters like 'a', 'c', and ‘z’, to punctuation like ‘;’ and ‘,’ and symbols like ‘%’ and ‘$’ - has a unique identifier; what’s known as its [Unicode](https://unicode.org/standard/WhatIsUnicode.html). Unicodes can be long and unwieldy (like 'U+0058' for the letter ‘X’) but we can assign each Unicode an integer instead. If you want to get an integer for the place of a certain character’s Unicode, use the built-in function Python function ord(). For example:

ord('a')

Outputs 97,

ord('b')

outputs 98, and

ord('c')

outputs 99.

You might be wondering why the letter ‘a’ is the 97th item in the Unicode system, but the reason is just that other characters - like the numerals ‘0’, ‘1’, and ‘2’ - come before them.

Now in the challenge, you may have to research exactly how to get the next letter in the alphabet given a character in Python. Do some digging. All programmers and data scientists do!

**Note**: it's possible that neither the input nor output string to your function is an intelligible English word; all that's required is that both are strings. Some of these strings might have special characters in them, as well as punctuation marks and capital letters. Remember: all of these characters have a Unicode. For each character in the string (including punctuation marks, special characters and capital letters) our function should get that character's Unicode v, and replace the original character by the character with the Unicode v + shift.

We also want our code to account for inputs that aren't strings, and respond to the user accordingly.

Let's take a few examples. Suppose our function is named cCipher, and we've accounted for non-string inputs by printing out the message 'The input is not a string'. Then the code:

print(cCipher('abc', 1))  
print(cCipher('123', 3))  
print(cCipher(44, 3))  
print(cCipher('143Hg!)>#', 2))  
print(cCipher("Here's 2 U MRS Robinson", 1))

Should output:

bcd

456

The input is not a string

365Ji#+@%

Ifsf(t!3!V!NST!Spcjotpo

That last example tells us that the successor of the Unicode of the whitespace character ' ' is actually the Unicode of the exclamation mark character '!'. To show this, the code:

print(ord(' '))  
print(ord('!'))

Outputs:

32

33

Their Unicodes are adjacent, which is why shifting the whitespace characters in the original string by 1 gave rise to exclamation marks.

Excited? Let's get going!

##### **Remember the basics:**

Remember how to declare functions syntactically. Don't forget your basic syntax. Look at examples that work. Check:

* Your basic control-flow structure syntax, like if-statement syntax.
* Your list manipulation syntax.
* Your function declaration syntax: def, return and arguments.
* Your colons and your indentation. Make sure your code blocks align.
* Your debugging cool. Keep calm, make printouts at strategic places to see where the Python interpreter is getting to, and what the values of certain variables are.

Tasks

**Brief** Write a Caesar Cipher function. That is, write a function which:

* takes two inputs, a string for the message, and an integer for the shift
* returns a string that's just the input string with all of its characters shifted by the amount specified in the shift argument
* for each character in the string (including punctuation marks, special characters and capital letters) gets that character's Unicode *v*, and replaces it by the character with the Unicode *v + shift*
* accounts for user inputs that aren't strings
* is as elegant and efficient as possible

**Submission** Once you've created your function, please submit it and discuss it with your mentor during your next call. To submit your function, click the 'Sharing' button in the toolbar on the left-hand side, two buttons above the 'Settings' gear button. Select the option to copy the URL, and paste it into the project submission box in the curriculum.

Excellent work. You're a star!

def cCipher(message, shift):

    encode = ""

    if type(message) != str:

        return "The input is not a string"

    for i in range(0, len(message)):

        encode = encode + chr(ord(message[i]) + shift)

    return encode

print(cCipher('abc' , 1))

print(cCipher('123' , 3))

print(cCipher(44, 3))

print(cCipher('143Hg!)>#', 2))

print(cCipher("Here's 2 U MRS Robinson", 1))

Background pattern

Description automatically generated

Object-oriented programming is a **way** of programming. It involves thinking of code in terms of 'objects' belonging to pre-defined types with certain pre-defined abilities and combining those objects to make them do cool things.

Python is a strongly-typed language — this means that the type of a certain piece of data matters (think integers, strings, etc.). Object-oriented programming is just thinking in terms of pre-defined types: everything we do in Python involves manipulating examples of types, so learning object-oriented programming will help you become a better Python programmer and better data scientist.

[Skip Navigation](https://next.tech/projects/5a56813b-91ad-40f2-b6a1-76a5ff10271e?access_token=DD9B2D4456DFA3CE97AEB8D7178BFD21&internal=false#left-sidebar)

3.1.2.1 Python Dictionaries

Unlike a physical dictionary like *Merriam-Webster*, a Python dictionary is an unordered collection of elements, where each element is a key-value pair. They're great fun to work with and are very useful for data that doesn't have an obvious order, but is related. Storing someone's contact details, for example, makes sense in a dictionary - you need to be able to access specific pieces of information (such as a telephone number) directly, and no order of number - email - postcode makes more sense than any other. Dictionaries are - like lists but unlike tuples - mutable (so we can update and remove elements), but they aren't ordered.

Dictionaries are created using curly brackets, and we can access the value of a particular key-value element using its key. We can either declare them on the same line, or across multiple lines, like so:

excitingbirds = {  
  "a":"barn owl",  
  "b":"guinea fowl",  
  "c":"parakeet",  
  "d":"woodpecker",  
  "e":"blue tit"  
}

Since dictionaries are unordered, we can't access the value "barn owl" with excitingbirds[0]. We need to use excitingbirds["a"].

Thanks to their mutability, we can update values in a dictionary, just as we can with lists:

excitingbirds["c"] = "sea eagle"

And create new dictionary entries by assigning a value to a key that doesn't exist yet:

excitingbirds["f"] = "white-backed vulture"

More needs to be said about what it is for a value to ‘repeat’ (i.e: occur as the ‘value’ part of a key-value pair more than once within the same dictionary).

In dictionaries, one and the same value, for example, 'barn owl' can occur as the value of more than one element. But no two different elements of the dictionary can have the same key without those elements being ignored. For example, the following is a dictionary whose length is 6.

excitingbirds = {  
  "a":"barn owl",  
  "b":"guinea fowl",  
  "c":"parakeet",  
  "d":"woodpecker",  
  "e":"blue tit",  
  "f": "guinea fowl"  
}

If we run:

print(excitingbirds)

We get output: {'a': 'barn owl', 'b': 'guinea fowl', 'c': 'parakeet', 'd': 'woodpecker', 'e': 'blue tit', 'f': 'guinea fowl'}

However if we have instead:

excitingbirds = {  
  "a":"barn owl",  
  "b":"guinea fowl",  
  "b":"parakeet",  
  "b":"woodpecker",  
  "b":"blue tit",  
  "f": "guinea fowl"  
}

Its length is actually 3. Suppose we run:

print(excitingbirds)

We get the output: {'a': 'barn owl', 'b': 'blue tit', 'f': 'guinea fowl'} While we don't get a syntax error, only the last element with the key b is recognised; the rest of them are ignored.

The reason Python behaves like this is that dictionaries are supposed to be unique identifiers for things. More than one thing can be identified by these identifiers, but no one identifier can identify more than one thing, and still be unique. It's like this: a given person can have more than form of valid identification. But no form of valid identification should pick out more than one person.

A quick word on (im)mutability, to consolidate what we know about it.

An immutable data type is one which, once declared, cannot be updated. E.g: tuples, strings, numbers. E.g, suppose we have a string:

s = 'Harry'

Because there are parallels between the way strings and lists behave, we can get the first character as follows:

print(s[0])

This outputs: H

But we can’t update this first character. If we try, with a statement like:

s[0] = 'L'

We get an error.

By contrast, a list is mutable. Suppose we have the list:

aList = ['Harry', 'Jerry']

And we try to update the first element:

aList[0] = 'Larry'

This works fine. If we do a printout of this list, we get: Larry, Jerry

Ready to create your first dictionary?

Quick note: when you complete the second task, please repair the syntax of the test\_dict before proceeding to the next exercise.

Tasks

Create a dictionary that contains resume data for a user, name that dictionary: user\_profile, and input the following elements:

| **Keys** | **Values** |
| --- | --- |
| 'name' | 'Mark' |
| 'age' | 16 |
| 'skills' | ['Python', 'Java'] |

Feedback

Click the checkbox above to attempt this task.

Checks

Unit TestIncomplete

Checking if the user\_profile is correctly defined

In the dictionary you've been provided with, which is called test\_dict, remove the **colon** that is located between the first element and the first value. (1 : "Hello" -> 1 "Hello") Try to execute the code.

Feedback

Click the checkbox above to attempt this task.

Checks

Code PatternIncomplete

No column between key and value

3.1.2.1 Python Dictionaries

Unlike a physical dictionary like *Merriam-Webster*, a Python dictionary is an unordered collection of elements, where each element is a key-value pair. They're great fun to work with and are very useful for data that doesn't have an obvious order, but is related. Storing someone's contact details, for example, makes sense in a dictionary - you need to be able to access specific pieces of information (such as a telephone number) directly, and no order of number - email - postcode makes more sense than any other. Dictionaries are - like lists but unlike tuples - mutable (so we can update and remove elements), but they aren't ordered.

Dictionaries are created using curly brackets, and we can access the value of a particular key-value element using its key. We can either declare them on the same line, or across multiple lines, like so:

excitingbirds = {  
  "a":"barn owl",  
  "b":"guinea fowl",  
  "c":"parakeet",  
  "d":"woodpecker",  
  "e":"blue tit"  
}

Since dictionaries are unordered, we can't access the value "barn owl" with excitingbirds[0]. We need to use excitingbirds["a"].

Thanks to their mutability, we can update values in a dictionary, just as we can with lists:

excitingbirds["c"] = "sea eagle"

And create new dictionary entries by assigning a value to a key that doesn't exist yet:

excitingbirds["f"] = "white-backed vulture"

More needs to be said about what it is for a value to ‘repeat’ (i.e: occur as the ‘value’ part of a key-value pair more than once within the same dictionary).

In dictionaries, one and the same value, for example, 'barn owl' can occur as the value of more than one element. But no two different elements of the dictionary can have the same key without those elements being ignored. For example, the following is a dictionary whose length is 6.

excitingbirds = {  
  "a":"barn owl",  
  "b":"guinea fowl",  
  "c":"parakeet",  
  "d":"woodpecker",  
  "e":"blue tit",  
  "f": "guinea fowl"  
}

If we run:

print(excitingbirds)

We get output: {'a': 'barn owl', 'b': 'guinea fowl', 'c': 'parakeet', 'd': 'woodpecker', 'e': 'blue tit', 'f': 'guinea fowl'}

However if we have instead:

excitingbirds = {  
  "a":"barn owl",  
  "b":"guinea fowl",  
  "b":"parakeet",  
  "b":"woodpecker",  
  "b":"blue tit",  
  "f": "guinea fowl"  
}

Its length is actually 3. Suppose we run:

print(excitingbirds)

We get the output: {'a': 'barn owl', 'b': 'blue tit', 'f': 'guinea fowl'} While we don't get a syntax error, only the last element with the key b is recognised; the rest of them are ignored.

The reason Python behaves like this is that dictionaries are supposed to be unique identifiers for things. More than one thing can be identified by these identifiers, but no one identifier can identify more than one thing, and still be unique. It's like this: a given person can have more than form of valid identification. But no form of valid identification should pick out more than one person.

A quick word on (im)mutability, to consolidate what we know about it.

An immutable data type is one which, once declared, cannot be updated. E.g: tuples, strings, numbers. E.g, suppose we have a string:

s = 'Harry'

Because there are parallels between the way strings and lists behave, we can get the first character as follows:

print(s[0])

This outputs: H

But we can’t update this first character. If we try, with a statement like:

s[0] = 'L'

We get an error.

By contrast, a list is mutable. Suppose we have the list:

aList = ['Harry', 'Jerry']

And we try to update the first element:

aList[0] = 'Larry'

This works fine. If we do a printout of this list, we get: Larry, Jerry

Ready to create your first dictionary?

Quick note: when you complete the second task, please repair the syntax of the test\_dict before proceeding to the next exercise.

Tasks

Create a dictionary that contains resume data for a user, name that dictionary: user\_profile, and input the following elements:

| **Keys** | **Values** |
| --- | --- |
| 'name' | 'Mark' |
| 'age' | 16 |
| 'skills' | ['Python', 'Java'] |

Feedback

Great work!

Checks

Unit TestComplete

Checking if the user\_profile is correctly defined

Test Output

{'name': 'Mark', 'age': 16, 'skills': ['Python', 'Java']}

.

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

Ran 1 test in 0.000s

OK

Test Contents

t\_dict = {"name":"Mark", "age":16, "skills":["Python", "Java"]}

class UnitTests(unittest.TestCase):

def test\_unit\_test(self):

for key in t\_dict.keys():

self.assertEqual(t\_dict[key], user\_profile[key])

In the dictionary you've been provided with, which is called test\_dict, remove the **colon** that is located between the first element and the first value. (1 : "Hello" -> 1 "Hello") Try to execute the code.

Feedback

Well done!

When creating a Python dictionary, the syntax is always the same: **{key : value, key: value}**.

If you remove or forget the **colon** between a key and its value, you'll get an error that looks like this: **SyntaxError: invalid syntax**.

The same rule holds true for the comma between pairs in a dictionary. If you forget to put a comma between two pairs, Python will throw the same error.

Both cases are easy to solve — you just need to find the element that is causing the problem and repair the syntax.

Checks

Code PatternComplete

No column between key and value

Description

Searched your code for a specific pattern:

test\_dict\s\*=\s\*{\s\*1\s\*"Hello"\s\*,\s\*2\s\*:\s\*"World"\s\*,\s\*3\s\*:\s\*3\s\*}

You can learn more about regular expressions [here](https://ruby-doc.org

3.1.2.2. Accessing elements

As you've learned, you generally use indexing to access values when you're working with other container types. When you're working with dictionaries, you use keys to access values.

You can use a key by placing it inside square brackets or by using the get() method.

Check out this example of a key in action:

person = {'name':'Mark', 'age': 16}  
   
#Accessing dictionary item by name  
print(person['name'])  
   
#Output:  
Mark  
   
#Accessing dictionary using get() method  
print(person.get('age'))  
   
#Output:  
16

When using square brackets, you'll see KeyError if the key can't be found. If you choose to use the get() method, instead, you'll see None instead of KeyError. This is pretty much the only difference between these two methods, but it can be very useful. We can even write our own custom error messages if the keys we're looking for don't exist.

For example, suppose we have the dictionary:

giantHeron = {'height': 1.61, 'weight': 59, 'speed': 'fast'}

Suppose we then execute:

print(giantHeron['gender'])

Because there’s no element whose key is ‘gender’, we get a KeyError. By contrast, with get(), we can write the following and account for this:

heronGender = giantHeron.get('gender', 'No gender value exists for this heron.')  
   
print(heronGender)

The above, when executed, outputs: No gender value exists for this heron.

`

Tasks

Using square brackets, access and print to the console the user 'age' inside the user\_profile that you created in the previous step.

Feedback

Click the checkbox above to attempt this task.

Checks

Test CaseIncomplete

Checking if you have correctly accessed age information of the user

Attempt this task to view the check results.

Use the get() method and print the'name' inside the user\_profile that you created in the previous step.

Feedback

Click the checkbox above to attempt this task.

Checks

Test CaseIncomplete

Checking if user's name is correctly accessed

Attempt this task to view the check results.

Code PatternIncomplete

Getting user name with get method

Attempt this task to view the check results.

[Skip Navigation](https://next.tech/projects/5a56813b-91ad-40f2-b6a1-76a5ff10271e?access_token=DD9B2D4456DFA3CE97AEB8D7178BFD21&internal=false&step_id=ccbc3164-1cba-4739-a6c5-da8d71f29105#left-sidebar)

Tasks

Using square brackets, access and print to the console the user 'age' inside the user\_profile that you created in the previous step.

Feedback

Fantastic!

Checks

Test CaseComplete

Checking if you have correctly accessed age information of the user

Input

Output

Examples of excitingbirds: {'a': 'barn owl', 'b': 'woodpecker', 'e': 'blue tit'}

Examples of excitingbirds: {'a': 'barn owl', 'b': 'woodpecker', 'e': 'blue tit', 'c': 'sea eagle'}

Examples of excitingbirds: {'a': 'barn owl', 'b': 'woodpecker', 'e': 'blue tit', 'c': 'sea eagle', 'f': 'white-backed vulture'}

{'name': 'Mark', 'age': 16, 'skills': ['Python', 'Java']}

16

Results

16

Expected Output

16

0

Use the get() method and print the'name' inside the user\_profile that you created in the previous step.

Feedback

Well done!

Checks

Test CaseComplete

Checking if user's name is correctly accessed

Input

Output

Mark

Results

Mark

Code PatternComplete

Getting user name with get method

Description

Searched your code for a specific pattern:

user\_profile.get

You can learn more about regular expressions [here](https://ruby-doc.org/core-2.1.1/Regexp.html).

3.1.2.3. Adding new elements and changing elements in a dictionary

Let's get clear on how to profit from the mutability of dictionaries and add and change new elements with the assignment operator.

If the key is already present when adding or changing elements, the value will get updated. Otherwise, a new (key:value) element will be added to the dictionary.

For example:

my\_dict = {'name':'Jack', 'age': 26}  
   
# update value  
my\_dict['age'] = 27  
   
#Output: {'age': 27, 'name': 'Jack'}  
print(my\_dict)  
   
# add item  
my\_dict['address'] = 'Downtown'

Tasks

Copy and paste the following code into the text editor. Try to understand exactly what it's doing.

convDict = {}  
first = input('Favourite college? ')  
convDict['College Preference'] = first  
print(convDict)

Feedback

Click the checkbox above to attempt this task.

Checks

Code PatternIncomplete

Dictionary conversation check

Attempt this task to view the check results.

Update the user's 'age' to 26 using the method discussed in the exercise.

Feedback

Click the checkbox above to attempt this task.

Checks

Unit TestIncomplete

Checking if user's age is properly updated to 26

Attempt this task to view the check results.

Using the method demonstrated in the example in this exercise, add the user's 'address' to the user\_profile dictionary.

Set his address to 'Downtown'

Feedback

Click the checkbox above to attempt this task.

Checks

Unit TestIncomplete

Checking user's address

Attempt this task to view the check results.

dict['address'] = 'Downtown'

Tasks

Copy and paste the following code into the text editor. Try to understand exactly what it's doing.

convDict = {}  
first = input('Favourite college? ')  
convDict['College Preference'] = first  
print(convDict)

Feedback

Well done!

Checks

Code PatternComplete

Dictionary conversation check

Description

Searched your code for a specific pattern:

convDict\s\*=\s\*{}[\n\t\s]\*first\s\*=\s\*input\s\*\(\s\*['"].\*['"]\s\*\)[\n\t\s]\*convDict\s\*\[\s\*['"].\*['"]\s\*\]\s\*=\s\*first[\n\t\s]\*print\s\*\(.\*\)

You can learn more about regular expressions [here](https://ruby-doc.org/core-2.1.1/Regexp.html).

Update the user's 'age' to 26 using the method discussed in the exercise.

Feedback

Woohoo!

Checks

Unit TestComplete

Checking if user's age is properly updated to 26

Test Output

{'name': 'Mark', 'age': 26, 'skills': ['Python', 'Java']}

.

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

Ran 1 test in 0.000s

OK

Test Contents

class UnitTests(unittest.TestCase):

def test\_unit\_test(self):

self.assertEqual(user\_profile['age'], 26)

Using the method demonstrated in the example in this exercise, add the user's 'address' to the user\_profile dictionary.

Set his address to 'Downtown'

Feedback

Fantastic work!

Checks

Unit TestComplete

Checking user's address

Test Output

{'name': 'Mark', 'age': 16, 'skills': ['Python', 'Java'], 'address': 'Downtown'}

.

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

Ran 1 test in 0.000s

OK

Test Contents

class UnitTests(unittest.TestCase):

def test\_unit\_test(self):

self.assertEqual(user\_profile['address'], 'Downtown')

3.1.2.4. Removing items from a dictionary using pop() method

You can remove a certain item in a dictionary by using the pop() method. This method will remove the item with the provided key and return the value. There are other ways - for example del - to remove elements from dictionaries, but the pop() method is uniquely useful because it *returns* the removed value, so we can go on to do things with it rather than just remove it. If we had a dictionary called dict containing an element whose key was called abc, the following would make sense:

poppedItem = dict.pop('abc')

Whereas the following would throw an error:

deletedItem = del dict['abc']

We can also be specific with pop() about which element we want removed. Passing a number *n* into the brackets of pop() removes the element with the key *n* and returns the value that corresponded to *n*:

# create a dictionary  
squares = {1:1, 2:4, 3:9, 4:16, 5:25}    
   
# removes the item 4:16  
print(squares.pop(4))    
# Outputs: 16  
   
print(squares)  
# Output: {1: 1, 2: 4, 3: 9, 5: 25}

The popitem() method removes the last element in the list and returns the removed item. For example:

# remove the last element '5:25' from squares, and prints  it  
print(squares.popitem())  
#Output: (5, 25)  
   
print(squares)  
# Output: {1: 1, 2: 4, 3: 9}

Let's look at a more real use-case of pop().

Suppose we've built a social media platform, and have made a dictionary holding information about its users:

users = {  
    1: {  
        'fname': 'Timothy',  
        'lname': 'McGregor',  
        'age': 22,  
        'gender': 'male',  
        'public profile': True  
    },  
   
    2: {  
        'fname': 'Lauren',  
        'lname': 'Matthews',  
        'age': 41,  
        'gender': 'female',  
        'public profile': False  
    },  
   
    3: {  
        'fname': 'Keith',  
        'lname': 'Rogers',  
        'age': 30,  
        'gender': 'male',  
        'public profile': False  
    }  
}

Notice that the keys for our users dictionary are numbers, and the values are themselves dictionaries. We'll make use of cool structures like this (also known as **nested dictionaries**) in the end of unit project!

Suppose that user 3 misbehaved: we need to remove them. But we also want to send them a message telling them this.

We'll pop them off and send them a printed message informing them of their being blocked from the service:

poppedUser = users[3]  
print('Dear '+ poppedUser['fname'] + ' ' + poppedUser['lname'] + '.' + '\nYou have been removed from the service.')

Tasks

Using the pop() method, remove the "address" item from the user\_profile dictionary.

Feedback

Click the checkbox above to attempt this task.

Checks

Unit TestIncomplete

Checking if the address key is deleted from the use\_profile

Attempt this task to view the check results.

Code PatternIncomplete

Using the pop() function

Attempt this task to view the check results.

Using the print() function, print out the user\_profile dictionary.

Feedback

Click the checkbox above to attempt this task.

Checks

Code PatternIncomplete

Print out the dictionary

Attempt this task to view the check results.

Tasks

Using the pop() method, remove the "address" item from the user\_profile dictionary.

Feedback

Way to go!

Checks

Unit TestComplete

Checking if the address key is deleted from the use\_profile

Test Output

{'name': 'Mark', 'age': 16, 'skills': ['Python', 'Java']}

.

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

Ran 1 test in 0.000s

OK

Test Contents

class UnitTests(unittest.TestCase):

def test\_unit\_test(self):

self.assertTrue("address" not in list(user\_profile.keys()))

Code PatternComplete

Using the pop() function

Description

Searched your code for a specific pattern:

user\_profile\.pop\s\*\(['"]address['"]\)

You can learn more about regular expressions [here](https://ruby-doc.org/core-2.1.1/Regexp.html).

Using the print() function, print out the user\_profile dictionary.

Feedback

1 out of 1 checks passed. Great work!

Checks

Code PatternComplete

Print out the dictionary

Description

Searched your code for a specific pattern:

print\s\*\(\s\*user\_profile\s\*\)

You can learn more about regular expressions [here](<https://ruby-doc.org/core-2.1.1/Regexp.html>).

3.1.2.5. Deleting and clearing

Want to delete items from your dictionary or scrap your dictionary completely? You can use the del keyword to remove individual items or get rid of the entire dictionary.

Take a look what happens when you delete items from a dictionary:

# create a dictionary  
squares = {1:1, 2:4, 3:9, 4:16, 5:25}    
   
# delete a particular item  
del squares[5]    
   
# Output: {1:1, 2:4, 3:9, 4:16}  
print(squares)

When removing an item from a dictionary (either with the .pop() method or with the del method) you should always provide the key that exists inside that dictionary.

If you try to delete an element with a key that does not exist, Python will throw the KeyError. Let's take a look at an example:

 squares = {1:1, 2:4, 3:9, 4:16, 5:25}   
 del squares[10]

The output ends up being:

KeyError: 10

Similarly, all the items in a dictionary can be removed at once using the clear() method. Check it out:

# create a dictionary  
squares = {1:1, 2:4, 3:9, 4:16, 5:25}    
   
# remove all items  
squares.clear()

Tasks

Using the del built-in Python function, remove the 'age' key item from the user\_profile dictionary.

Feedback

Click the checkbox above to attempt this task.

Checks

Unit TestIncomplete

Checking if the age key is deleted from the use\_profile

Attempt this task to view the check results.

Code PatternIncomplete

Using del built-in Python function

Attempt this task to view the check results.

Using the print() function, print out the user\_profile dictionary.

Feedback

1 out of 1 checks passed. Great work!

Checks

Code PatternComplete

Print out the dictionary

Description

Searched your code for a specific pattern:

print\s\*\(\s\*user\_profile\s\*\)

You can learn more about regular expressions [here](https://ruby-doc.org/

[Skip Navigation](https://next.tech/projects/5a56813b-91ad-40f2-b6a1-76a5ff10271e?access_token=DD9B2D4456DFA3CE97AEB8D7178BFD21&internal=false&step_id=b7d02c35-d0d8-4d30-8c36-b4e5add51f33#left-sidebar)

3.1.2.6. Iterating through a dictionary

One way to iterate through a dictionary you're working with is by using its keys. To get all keys in your dictionary, you can simply say: dict.keys()

Suppose we have the dictionary:

my\_dict = {1:2, 2:4, 3:6}

We can get all the keys like this:

for i in my\_dict.keys():  
    print(i)

So the above, when executed, outputs:

1

2

3

Now by default, when we iterate through dictionaries, keys are returned, not whole elements (key-value pairs). I.e, the following code outputs the same thing as the above:

for i in my\_dict:  
    print(i)

If we want the key-value pairs themselves, we can iterate through dictionaries with the method items(). Suppose we have:

excitingbirds = {  
  "a":"barn owl",  
  "b":"guinea fowl",  
  "c":"parakeet",  
  "d":"woodpecker",  
  "e":"blue tit"  
}

We can get all the key-value pairs as follows:

for k, v in excitingbirds.items():  
    print(k, v)

Note: k and v are like counter variables like i and j in for loops: we could have called them anything at all.

As you might expect, there's also a dictionary method for getting just the values of the elements. No surprises: this method is called values()! It works thus:

for i in excitingbirds.values():  
    print(i)

This outputs:

barn owl

guinea fowl

parakeet

woodpecker

blue tit

Tasks

Make a dictionary called friends containing as its keys your 3 best friends, and as its values their favorite countries. Make both the keys and the values strings.

Using a for loop with counter variables k, v that iterates through the friends.items(), print out the following string for each iteration:

k + "'s favourite country is: " + v

If no two of your friends already have the same favorite country, make two of them have the same favorite country. Print all of your friends’ favorite countries.

For example, if your dictionary looked like this:

friends = {  
    'Cuth':'USA',  
    'Billy':'Italy',  
    'James':'Portugal'  
}

You could update your dictionary to make two of your friends' favorite countries the same like this:

friends['Cuth'] = 'Italy'  
print(friends)

Make a Python dictionary that acts as a glossary for all the Python terms you’ve learned. Use new line breaks \n to insert a blank line between each word-meaning pair.

Feedback

Click the checkbox above to complete this task.

3.1.2.7. Converting a dictionary to a Python list

Occasionally, you may run into instances where you need to convert a dictionary to a list. Good news: this is pretty easy to do.

In fact, all you need to add is list(dict.items())

See list() in the above bit of code? That's a built-in Python that converts the input to a Python list.

Take a look:

excitingbirds = {  
  "a":"barn owl",  
  "b":"guinea fowl",  
  "c":"parakeet",  
  "d":"woodpecker",  
  "e":"blue tit"  
}  
   
list\_dict = list(excitingbirds.items())  
   
print(list\_dict)

Output: [('a', 'barn owl'), ('b', 'guinea fowl'), ('c', 'parakeet'), ('d', 'woodpecker'), ('e', 'blue tit')]

The elements of this list are now actually *tuples*! Cool huh?

Tasks

Using the method explained in the exercise material above, covert the user\_profile dictionary into a list.

If you've lost what the user\_profile should be looking like, here it is again for you:

user\_profile = {  
    'name': 'Mark',  
    'age': 16,  
    'skills': ['Python', 'Java']  
}

NOTE: Name the list user\_list

Feedback

Click the checkbox above to attempt this task.

Checks

Code PatternIncomplete

Converting user\_profile into list

After you convert the user\_profile into a list named user\_list, print it out, so you can see its structure.

Feedback

Click the checkbox above to attempt this task.

Checks

Code PatternIncomplete

Print out the list version of user\_profile

Attempt this task to view the check results.

Tasks

Using the method explained in the exercise material above, covert the user\_profile dictionary into a list.

If you've lost what the user\_profile should be looking like, here it is again for you:

user\_profile = {  
    'name': 'Mark',  
    'age': 16,  
    'skills': ['Python', 'Java']  
}

NOTE: Name the list user\_list

Feedback

Great job!

Checks

Code PatternComplete

Converting user\_profile into list

After you convert the user\_profile into a list named user\_list, print it out, so you can see its structure.

Feedback

You got it!

Checks

Code PatternComplete

Print out the list version of user\_profile

Description

Searched your code for a specific pattern:

print\s\*\(\s\*user\_list\s\*\)

You can learn more about regular expressions [here](https://ruby-doc.org/core-2.1.1/Regexp.html).

3.1.2.8. Converting a Python list to a dictionary

Once you've converted your dictionary to a list, you can turn your list back into a dictionary.

We have to make a distinction now between two categories of list in Python (although these aren't strictly two different data types). A **structured list** is one all of whose members are tuples, and if a list isn't structured because it has at least one element that isn't a tuple, it's called **unstructured**.

Let's handle the case of converting a structured list to a dictionary. We can actually use the user\_list that was the output of the previous page's exercise.

To convert this list into a dictionary, simply add a built-in function called dict(). This function creates dictionaries from its arguments:

my\_list = [(1, 1), (2, 2), (3, 3)]  
   
my\_dict = dict(my\_list)  
   
print(my\_dict)

Outputs: {1:1, 2:2, 3:3}

As you can see, in this list, each element is a tuple. In each bracket, the element on the left will become the dictionary key and the element on the right will become its value.

Here, take a look at this more explicit example:

my\_list\_2 = [('dish', 'pizza'), ('price', '40$')]  
   
my\_dict\_2 = dict(my\_list\_2)  
   
print(my\_dict\_2)

Output: {'dish':'pizza', 'price':'40$'}

As for converting unstructured lists to dictionaries, this is fairly involved. Only if you're interested, check out this [link](https://stackoverflow.com/questions/6900955/python-convert-list-to-dictionary).

Ready to try this out for yourself? We thought so. Have at it!

Tasks

Using the dict() function, covert the user\_list back into a dictionary. Name it user\_dict.

Hint: Reread the introductory materials of this exercise — it'll show you how to complete this task.

Feedback

Click the checkbox above to attempt this task.

Checks

Code PatternIncomplete

Covert user\_list back to the user\_profile

Using the print() function, print out the user\_dict to see the structure of the dicitonary.

Feedback

Click the checkbox above to attempt this task.

Checks

Code PatternIncomplete

Print out the newly created user\_dict

Attempt this task to view the check results.

Tasks

Using the dict() function, covert the user\_list back into a dictionary. Name it user\_dict.

Hint: Reread the introductory materials of this exercise — it'll show you how to complete this task.

Feedback

You're on fire!

Checks

Code PatternComplete

Covert user\_list back to the user\_profile

Using the print() function, print out the user\_dict to see the structure of the dicitonary.

Feedback

Excellent job!

Checks

Code PatternComplete

Print out the newly created user\_dict

Description

Searched your code for a specific pattern:

print\s\*\(\s\*user\_dict\s\*\)

You can learn more about regular expressions [here](https://ruby-doc.org/core-2.1.1/Regexp.html).

"""

test\_dict = {1"Hello", 2:"World", 3:3}

print("This is my test\_dict:", test\_dict)

#Change code above this line only if the task is asking you to

excitingbirds = {

  "a":"barn owl",

  "b":"guinea fowl",

  "b":"parakeet",

  "b":"woodpecker",

  "e":"blue tit"

}

print("Examples of excitingbirds:",  excitingbirds)

excitingbirds["c"] = 'sea eagle'

print("Examples of excitingbirds:",  excitingbirds)

excitingbirds["f"] = 'white-backed vulture'

print("Examples of excitingbirds:",  excitingbirds)

user\_profile = {

    'name':'Mark',

    'age': 16,

    'skills': ['Python' , 'Java']

}

#user\_profile['age'] = 26

user\_profile['address'] = 'Downtown'

#poppedItem = user\_profile.pop('address')

del user\_profile['age']

print(user\_profile)

#print(user\_profile)

#print(user\_profile['age'])

#print(user\_profile.get('name'))

my\_dict = {'name':'Jack', 'age': 26}

# update value

my\_dict['age'] = 27

#Output: {'age': 27, 'name': 'Jack'}

print(my\_dict)

# add item

my\_dict['address'] = 'Downtown'

print(my\_dict)

convDict = {}

first = input('Favourite college? ')

convDict['College Preference'] = first

print(convDict)

# create a dictionary

squares = {1:1, 2:4, 3:9, 4:16, 5:25}

# removes the item 4:16

print(squares.pop(4))

# Outputs: 16

print(squares)

# Output: {1: 1, 2: 4, 3: 9, 5: 25}

users = {

    1: {

        'fname': 'Timothy',

        'lname': 'McGregor',

        'age': 22,

        'gender': 'male',

        'public profile': True

    },

    2: {

        'fname': 'Lauren',

        'lname': 'Matthews',

        'age': 41,

        'gender': 'female',

        'public profile': False

    },

    3: {

        'fname': 'Keith',

        'lname': 'Rogers',

        'age': 30,

        'gender': 'male',

        'public profile': False

    }

}

poppedUser = users[3]

print('Dear '+ poppedUser['fname'] + ' ' + poppedUser['lname'] + '.' + '\nYou have been removed from the service.')

friends = {

    'Cuth': 'USA',

    'Billy': 'Italy',

    'James': 'Portugal'

}

friends['Cuth'] = 'Italy'

for k,v in friends.items():

    print(k + "'s favourite country is: " + v)

pythonTerms = {

    'key': 'first object',

    'value' : 'second object',

    'get method' : 'using keys to access values',

    'pop method' : 'returns the removed value',

    'popitem' : 'removes the last element in the list',

    'del' :'removes element from the dictionary',

    'items method' : 'iterates through the dictionary and prints the key-value pairs',

    'values method' : 'gets the values of the elements'

}

for k, v in pythonTerms.items():

    print('\n',k + ' : ' +v)"""

user\_profile = {

    'name': 'Mark',

    'age': 16,

    'skills': ['Python', 'Java']

}

user\_list = list(user\_profile.items())

user\_dict = dict(user\_list)

print(user\_dict)

Numbers are not the only units of data you’re going to deal with as a data scientist. You’re likely to also work with pieces of arbitrary text. In programming parlance, these pieces of text are called "strings". For example, the word "hello" is a string; so is a sentence like "We better call the fire department 'cuz your coding skills are on fire ." In this lesson, you'll learn how to create and modify strings.

3.1.3.1. Strings

While you've worked with strings in previous lessons and projects, we haven't yet looked in detail at their methods. In this lesson, we'll take a closer look at strings and learn some of the ways that we can use them when working on different projects.

String manipulation is super powerful. Natural Language Processing (one of the booming areas of data science) requires artistry in this area. Additionally, any data science technique is only valid if it’s executed on clean data. String slicing is absolutely critical for data cleaning, which can actually be very fun if we know how to do it!

Firstly, since strings are sequences of characters, we can find out how many characters are in a string using our trusty friend: the len() method:

string\_1 = "Hello World!"  
print(len(string\_1))

Output: 12

Tasks

Using the len function, print out the length of the given string, which is called string\_1.

Feedback

Click the checkbox above to attempt this task.

Checks

Test CaseIncomplete

Checking if you have received the correct result.

Attempt this task to view the check results.

Code PatternIncomplete

Check if the lenght of the string\_1 is hard-coded

Attempt this task to view the check results.

Using the len function, print out the length of the given string, which is called string\_2.

Feedback

Click the checkbox above to attempt this task.

Checks

Code PatternIncomplete

Check the lenght of the string\_2

Attempt this task to view the check results.

Tasks

Using the len function, print out the length of the given string, which is called string\_1.

Feedback

Heck ya! You're killing it!

Checks

Test CaseComplete

Checking if you have received the correct result.

Input

Output

23

Results

23

Expected Output

23

0

Code PatternComplete

Check if the lenght of the string\_1 is hard-coded

Description

Searched your code for a specific pattern:

prin\s\*t\(\s\*23\s\*\)

You can learn more about regular expressions [here](https://ruby-doc.org/core-2.1.1/Regexp.html).

Using the len function, print out the length of the given string, which is called string\_2.

Feedback

Heck ya! You're killing it!

Checks

Code PatternComplete

Check the lenght of the string\_2

Description

Searched your code for a specific pattern:

prin\s\*t\(\s\*len\(\s\*string\_2\s\*\)\s\*\)

You can learn more about regular expressions [here](https://ruby-doc.org/core-2.1.1/Regexp.html)

3.1.3.2. Finding a character or a substring

In order to find the index of a character in a string, you can use the very appropriately named index() function. For example:

string\_2 = "Python programming language"  
print(string\_2.index('o'))

Output: 4

The output is 4 because the location of the first occurrence of the letter "o" is 4 characters away from the first character in the string. Notice how there are actually two o's in this string; this method only recognizes the first of them.

You may be asking yourself, but why didn't it print out 5? Isn't "o" the fifth character in the string? This is where the **0-indexed** rule in Python comes back into play. This rule applies to everything 'countable', which includes strings. The first character has a **0** index, not **1**.

If we're trying to access a substring that does not exist by using the index() method; It will throw an error saying that ValueError: substring not found. Bear in mind, too, that the index() method counts whitespace as characters, too.

Tasks

Print out the index of the character "i" in the string\_1.

Psst — use the index() method :)

Feedback

Click the checkbox above to attempt this task.

Checks

Test CaseIncomplete

Checking out if the index of character i in the string\_1 is correct.

Attempt this task to view the check results.

Code PatternIncomplete

Check out the index of letter "i" in the string\_1

Attempt this task to view the check results.

Print out the index of the character "P" in the string\_3.

Psst — usee the index() method :)

Feedback

Click the checkbox above to attempt this task.

Checks

Test CaseIncomplete

Checking if the index of the character P in the string\_3 is correct

Attempt this task to view the check results.

Code PatternIncomplete

Checking for capital P in the string\_3

Attempt this task to view the check results.

Tasks

Print out the index of the character "i" in the string\_1.

Psst — use the index() method :)

Feedback

Great job!

Checks

Test CaseComplete

Checking out if the index of character i in the string\_1 is correct.

Input

Output

7

Results

7

Expected Output

7

0

Code PatternComplete

Check out the index of letter "i" in the string\_1

Description

Searched your code for a specific pattern:

string\_1\.index\(\s\*['"]i['"]s\*\)

You can learn more about regular expressions [here](https://ruby-doc.org/core-2.1.1/Regexp.html).

Print out the index of the character "P" in the string\_3.

Psst — usee the index() method :)

Feedback

Awesome job!

Checks

Test CaseComplete

Checking if the index of the character P in the string\_3 is correct

Input

Output

35

Results

35

Expected Output

35

0

Code PatternComplete

Checking for capital P in the string\_3

Description

Searched your code for a specific pattern:

string\_3\.index\(\s\*['"]P['"]s\*\)

You can learn more about regular expressions [here](https://ruby-doc.org/core-2.1.1/Regexp.html).

3.1.3.3. String methods

As you saw in the previous exercise, there can be several occurrences of a single character within a string. So, how can you find out the correct number of occurrences of a given character in a string? By using an old friend: the for loop.

To find the number of occurrences of a given character in a string, use a for- loop and iterate over the given string.

Curious about how this might look? Take a peek at this example:

counter = 0  
string\_3 = "Being a Data Scientist has provided great opportunities for me!"  
   
for char in string\_3:  
    if (char == 'o'):  
        counter += 1  
   
print(counter)

Output: 4

As you can see in this example, we had to define a counter variable, iterate through the given string, and check each character to see if it matches the character we want to find (in this case, 'o'.) Each time it does, we add 1 to the counter, and then print counter to see the number of occurrences. Bear in mind: in the for loop, we could have called the char variable anything, including i, j, or chicken.

We can actually do this much more quickly with a single method: count(). The following code does the same thing as the above, i.e, checks how many occurrences of the character 'o' there are in the string 'Being a Data Scientist has provided great opportunities for me!':

print(string\_3.count('o'))

This also outputs: 4

We can also check for substrings, rather than single characters, with both index() and count(). For example:

string\_3 = "Being a Data Scientist has provided great opportunities for me!"  
# Print the index within string\_3 of the first character of the substring 'ovi'  
print(string\_3.index('ovi'))  
   
# print a count of the occurrences of 'ovi' in string\_3  
print(string\_3.count('ovi'))

This outputs:

29

1

Both the index() and the count() method can look for single characters as well as substrings, but index() returns the index within the overall string of the first character in the argument we pass to it, and count() returns the number of times the substring we pass to it occurs in the overall string.

Sweet right? Now try using this method on your own code!

Task

Using the count() method, count how many times the character u occurs in string\_1

Feedback

Click the checkbox above to attempt this task.

Checks

Test CaseIncomplete

Checking out if the number of character u found in the string\_1 is correct.

Attempt this task to view the check results.

Code PatternIncomplete

Count how many u characters are there in the string\_1

Attempt this task to view the check results.

Task

Using the count() method, count how many times the character u occurs in string\_1

Feedback

Fantastic!

Checks

Test CaseComplete

Checking out if the number of character u found in the string\_1 is correct.

Input

Output

2

Results

2

Expected Output

2

0

Code PatternComplete

Count how many u characters are there in the string\_1

Description

Searched your code for a specific pattern:

string\_1\.count\(\s\*['"]u['"]\s\*\)

You can learn more about regular expressions [here](https://ruby-doc.org/core-2.1.1/Regexp.html)

[Skip Navigation](https://next.tech/projects/76367fb4-5ee1-498c-83ed-4db4c7c2293a?access_token=C876BBF34ADFAE1C03DFAEE320978B08&internal=false&step_id=56238595-a91d-453b-9ddf-1fdbb96b475b#left-sidebar)

3.1.3.4. String slicing

As we mentioned in the first exercise of this mini-course, strings are simply sequences of characters. With that in mind, slicing works similarly for strings as it does for lists.

Can't quite remember the syntax for slicing lists? Take a look at this:

list\_1 = [1, 2, 3, 4, 5]  
   
#Slicing from index to index  
print(list\_1[1:3])  
   
#Slicing from the beginning of a list up to an index  
print(list\_1[:3])  
   
#Slicing from an index up to the ending of a list  
print(list\_1[4:])

Output:

[2, 3]

[1, 2, 3]

[5]

All the different ways you've sliced lists can be applied to slicing strings - the output of these sliced strings is a substring. Take a look at a couple of examples to understand what we're talking about.

Example 1:

string\_1 =  "Hello world!"  
   
print(string\_1[:5])

Output: Hello

Example 2:

string\_1 =  "Hello world!"  
   
print(string\_1[6:])

Output: world!

Example 3:

string\_1 =  "Hello world!"  
   
print(string\_1[1:5])

Output: ello

Your turn!

**Important note:** When you try to index a string with an index that is larger than the length of the string, Python will stop executing your code and will throw the **IndexError**. Take a look at this example:

string\_1 = "Hello world!"

print(string\_1[100]) #Trying to access a character at the index 100

Output: IndexError: string index out of range

Hints:

1. Remember: spaces are counted as characters as well! So count them as one index while trying to find the beginning and ending of the word.
2. You can use index() method to get the starting index of the letter in the desired word.

Task

Using the **slicing** technique, print out the word learn from string\_2.

Remember, the first index is 0 (zero).

Feedback

Click the checkbox above to attempt this task.

Checks

Code PatternIncomplete

Checking if a word learn is not hard-coded in a print statement.

Attempt this task to view the check results.

Test CaseIncomplete

Checking if you have successfully extracted the word learn from the string\_2

Attempt this task to view the check results.

Code PatternIncomplete

Using the slicing technique get word learn from the string\_2

Attempt this task to view the check results.

Task

Using the **slicing** technique, print out the word learn from string\_2.

Remember, the first index is 0 (zero).

Feedback

Awesome job!

Checks

Code PatternComplete

Checking if a word learn is not hard-coded in a print statement.

Description

Searched your code for a specific pattern:

print\s\*\(\s\*['"]learn['"]\s\*\)

You can learn more about regular expressions [here](https://ruby-doc.org/core-2.1.1/Regexp.html).

Test CaseComplete

Checking if you have successfully extracted the word learn from the string\_2

Input

Output

learn

Results

learn

Expected Output

learn

0

Code PatternComplete

Using the slicing technique get word learn from the string\_2

Description

Searched your code for a specific pattern:

string\_2\s\*\[

You can learn more about regular expressions [here](https://ruby-doc.org/core-2.1.1/Regexp.html)

[Skip Navigation](https://next.tech/projects/76367fb4-5ee1-498c-83ed-4db4c7c2293a?access_token=C876BBF34ADFAE1C03DFAEE320978B08&internal=false&step_id=b4c8e3ce-2ab0-49c9-8546-e201e76a7050#left-sidebar)

3.1.3.5. UPPERCASE and lowercase strings

In addition to searching and slicing strings, there are a couple of other ways that strings make themselves useful.

Two helpful string methods are upper() and lower().

The upper() method takes a string and makes all of the characters in that string uppercase. The lower() method takes a string and makes all of its characters lowercase.

These two methods come in handy much more frequently than you might think. For instance, you can use the lower() method when creating a function to check each word in a text in order to compare those with the "query" word, as your search will be dependent on the character casing of the original text.

Another great use of these functions is accounting for user inputs. Suppose we're making a rock paper scissors game where the user has to input one of those words. Well instead of making large if-statements of the form:

choice = input("Please type 'rock', 'paper', or 'scissors': " )

if (choice == 'rock' or choice == 'Rock' or choice == 'ROCK' or choice == 'RoCk'):

# Do something

We can just make the input lowercase by default, as follows:

choice = input("Please type 'rock', 'paper', or scissors': )  
choice = choice.lower()  
if (choice == 'rock'):  
    # Do something

Let's look at a simpler example:

string\_4 = "Hello World!"  
print(string\_4.upper())  
print(string\_4.lower())

Output:

HELLO WORLD!

hello world!

Remember too that both the count() and index() methods are case-sensitive, so using the lower() or upper() methods when using count() and index() will help you to find the exact number of words or characters that you're looking for.

Let's see that example:

string\_3 = "Being a Data Scientist has provided   
great opportunities for me!"  
   
print(string\_3.count('d'))

Output: 2

If you use the count method to find the letter d it will return 2 as a result. That's not what we're looking for, because there is a D in the word Data as well. With that in mind, we need to add lower().

string\_3 = "Being a Data Scientist has provided   
great opportunities for me!"  
   
print(string\_3.lower().count('d'))

Output: 3

By using the lower() method, we managed to count all occurrences of the letter d.

You'll be combining all these methods in this exercise so buckle up! You've got this.

Tasks

Use the lower() method on string\_1 and print the result to your console.

Feedback

Click the checkbox above to attempt this task.

Checks

Test CaseIncomplete

Checking if the lower version of the string\_1 is printed correctly

Attempt this task to view the check results.

Code PatternIncomplete

Checking if the lower version of the string\_1 is not hard-coded

Attempt this task to view the check results.

Code PatternIncomplete

Use lower() method on the string\_1

Attempt this task to view the check results.

Use the upper() method on string\_2 and print the results to the console.

Feedback

Click the checkbox above to attempt this task.

Checks

Test CaseIncomplete

Checking if the upper version of the string\_2 is correctly printed out

Attempt this task to view the check results.

Code PatternIncomplete

Checking if the upper version of the string\_2 is not hard-coded.

Attempt this task to view the check results.

Code PatternIncomplete

Use the upper() on the string\_2

Attempt this task to view the check results.

Tasks

Use the lower() method on string\_1 and print the result to your console.

Feedback

Woohoo! Nicely done.

Checks

Test CaseComplete

Checking if the lower version of the string\_1 is printed correctly

Input

Output

data science is future!

Results

data science is future!

Expected Output

data science is future!

0

Code PatternComplete

Checking if the lower version of the string\_1 is not hard-coded

Description

Searched your code for a specific pattern:

data science is future!

You can learn more about regular expressions [here](https://ruby-doc.org/core-2.1.1/Regexp.html).

Code PatternComplete

Use lower() method on the string\_1

Description

Searched your code for a specific pattern:

string\_1\.lower\s\*\(\s\*\)

You can learn more about regular expressions [here](https://ruby-doc.org/core-2.1.1/Regexp.html).

Use the upper() method on string\_2 and print the results to the console.

Feedback

Excellent!

Checks

Test CaseComplete

Checking if the upper version of the string\_2 is correctly printed out

Input

Output

data science is future!

EVERYBODY CAN LEARN PROGRAMMING!

Results

EVERYBODY CAN LEARN PROGRAMMING!

Expected Output

EVERYBODY CAN LEARN PROGRAMMING!

0

Code PatternComplete

Checking if the upper version of the string\_2 is not hard-coded.

Description

Searched your code for a specific pattern:

EVERYBODY CAN LEARN PROGRAMMING!

You can learn more about regular expressions [here](https://ruby-doc.org/core-2.1.1/Regexp.html).

Code PatternComplete

Use the upper() on the string\_2

Description

Searched your code for a specific pattern:

string\_2\.upper\s\*\(\s\*\)

You can learn more about regular expressions [here](https://ruby-doc.org/core-2.1.1/Regexp.html).

[Skip Navigation](https://next.tech/projects/76367fb4-5ee1-498c-83ed-4db4c7c2293a?access_token=C876BBF34ADFAE1C03DFAEE320978B08&internal=false&step_id=34ea1c38-3f9b-49e1-8f0c-67645ac4e494#left-sidebar)

3.1.3.6. Splitting

When you're working with text, it's pretty common to split strings into tokens, which are generally words.

The split() method takes a substring (which will be a string, be it a string containing just one character, or a longer one) and splits a string on that substring. The split() method is extremely useful for data cleaning.

If this is a tad confusing, take a look at this example:

string\_5 = "Today is a very nice day!"  
print(string\_5.split(" ")) #split on space

Output: ['Today', 'is', 'a', 'very', 'nice', 'day!']

As you can see, the output after the split function is a Python list that contains all the parts of the source string, except the substring used for a split.

Ready to experiment with splits and tokens?

Task

Use the split() method on either string\_1 or string\_2 — you get to choose your own adventure for this task :)

1. Use the split character space! (" ")
2. Print out your results to the console.

Feedback

Click the checkbox above to attempt this task.

Checks

Code PatternIncomplete

Splitting strings to words

Attempt this task to view the check results.

Task

Use the split() method on either string\_1 or string\_2 — you get to choose your own adventure for this task :)

1. Use the split character space! (" ")
2. Print out your results to the console.

Feedback

Yeehaw! Way to go!

Checks

Code PatternComplete

Splitting strings to words

Description

Searched your code for a specific pattern:

print\s\*\(\s\*string\_[213]\.split\s\*\(\s\*['"]\s['"]\s\*\)\s\*\)

You can learn more about regular expressions [here](https://ruby-doc.org/core-2.1.1/Regexp.html)

[Skip Navigation](https://next.tech/projects/76367fb4-5ee1-498c-83ed-4db4c7c2293a?access_token=C876BBF34ADFAE1C03DFAEE320978B08&internal=false&step_id=a2f25129-3052-4007-bec6-d8aff8bea2d8#left-sidebar)

3.1.3.7. Further string methods

To finish up with strings and the operations you can perform on them, let's talk about a few more methods. We'll start with startswith() and endswith().

You might be able to guess what these two methods do based on their names.

startswith() takes a substring and checks to see if a given string starts with that substring.

endswith() takes a substring and checks to see if a given string ends with that substring.

As a reminder, a substring can be a string containing a single character, a word, a jumble of characters that isn't a word, or a whole sentence!

For example:

string\_6 = "Artificial Intelligence is cool!"  
print(string\_6.startswith("Artificial"))  
print(string\_6.endswith("nice!"))

Output:

True

False

These methods always return True or False, so you can use the result inside an if-statement too.

Another surprisingly useful method is title(). Check out this example:

name = 'edgar mcdermott'  
print(name.title())

This outputs: Edgar Mcdermott It essentially makes the first character in each distinct word upper case. This can save us a lot of time in certain scenarios!

There are also some really useful things called f strings. These help us assign longer strings to variables, by making use of pre-existing variables. Take this example:

fname = 'Lewis'  
lname = 'Carroll'  
fullname = f"Hello, {fname} {lname}! Nice to meet you."  
print(fullname)

This outputs: Hello, Lewis Carroll! Nice to meet you.

The syntax above might look a bit weird. But basically all we're doing is writing the letterf, immediately followed by a double quotation mark ", then we can make a long string by typing out characters and placing variables within curly brackets.

Another neat trick is new line placements. We can implement this with the expression \n. For example:

print('Skills:\nLogic\nEngineering\nMusic\nWriting')

Outputs:

Skills:

Logic

Engineering

Music

Writing

And the final method we'll cover, my personal favorite, is the replace() method. The first argument is the substring we want to replace, and the second is the string we want to replace it by. It's super useful in almost any data science or analysis context. Check it out:

welcome = 'Hello Harry! How are you today Harry? Very nice to see you, Harry.'  
welcome = welcome.replace('Harry', 'Jerry')  
print(welcome)

This outputs: Hello Jerry! How are you today Jerry? Very nice to see you, Jerry.

Ready to apply some of these methods to your own code?

Tasks

Using the startswith() method, check if the string\_1 starts with "Data".

Please print your results to the console.

Feedback

Click the checkbox above to attempt this task.

Checks

Test CaseIncomplete

Checking if the string\_1 starts with a substring - Data

Attempt this task to view the check results.

Code PatternIncomplete

Checking the start of a string

Attempt this task to view the check results.

Using the endswith() method, check if the string\_2 ends with "Python".

Please print your results to the console.

Feedback

Click the checkbox above to attempt this task.

Checks

Test CaseIncomplete

Checking if the string\_2 ends with a substring - Python

Attempt this task to view the check results.

Code PatternIncomplete

Checking a ending of a string

Attempt this task to view the check results.

Make a string of your favorite quote by a thinker, philosopher or novelist. Print it out with interesting formatting (for example new lines for new sentences). Replace all occurrences of your favorite word in that quote by the string 'sausages'.

Feedback

Click the checkbox above to complete this task.

Tasks

Using the startswith() method, check if the string\_1 starts with "Data".

Please print your results to the console.

Feedback

Yay!

Checks

Test CaseComplete

Checking if the string\_1 starts with a substring - Data

Input

Output

True

False

Results

True

Expected Output

True

0

Code PatternComplete

Checking the start of a string

Description

Searched your code for a specific pattern:

string\_[123]\.startswith

You can learn more about regular expressions [here](https://ruby-doc.org/core-2.1.1/Regexp.html).

Using the endswith() method, check if the string\_2 ends with "Python".

Please print your results to the console.

Feedback

Great job!

Checks

Test CaseComplete

Checking if the string\_2 ends with a substring - Python

Input

Output

True

False

Results

False

Expected Output

False

0

Code PatternComplete

Checking a ending of a string

Description

Searched your code for a specific pattern:

string\_[123]\.endswith

You can learn more about regular expressions [here](https://ruby-doc.org/core-2.1.1/Regexp.html).

Make a string of your favorite quote by a thinker, philosopher or novelist. Print it out with interesting formatting (for example new lines for new sentences). Replace all occurrences of your favorite word in that quote by the string 'sausages'.

Feedback

Excellent work. Well done :)

PYTHON CONTROL FLOW PART 2

In this lesson, you'll learn about advanced tips and tricks for using loops. First, you’ll learn about nested loops, which are basically loops within loops.

* To put this in a real-world context, if you were to tell somebody to go through each room of a house and open each window, that would be you using a nested loop (loop 1 being ‘go to each room’ and loop 2 being ‘open each window once you’re in each room.’) Not sure why you're not helping them open the windows, but hey, maybe you're just a boss like that.

You’ll also learn about situations when your program needs to exit a loop prematurely because it has accomplished the task you requested (kind of like if you were looking for a book on a bookshelf and you found the book — you’d walk away from the bookshelf instead of searching through the rest of the books for the book you have in your hand).  This lesson will teach you how to break out of a loop or continue executing it depending on whether certain conditions are met.

[Skip Navigation](https://next.tech/projects/d5a61838-c8fd-43bb-8860-fad8d2623757?access_token=1107940D3D61C33285B180EDD27B8C5F&internal=false#left-sidebar)

3.1.4.1. Nested Loops

Ready to kick your control flow skills up a notch? In this mini-course, you'll learn some advanced control flow techniques.

You were introduced to for-loops in the first mini-course about control flow and used for-loops to iterate over list items. But what would you do if you ran into a list that looked like this:

companies = [['Google', 'Facebook', 'Apple'],  
            ['Warner Bros. Pictures', '20th Century Fox', 'Universal Pictures'],  
            ['Whole Foods', 'Starbucks', 'Walmart']]

In this version of the companies list, each item in the list is another list! Accessing the first element:

companies[0]

will return the whole list: ['Google', 'Facebook', 'Apple']

To access a particular item - let's say, 'Facebook' - you would type this:

`companies[0][1]`

Output: Facebook

What's going on here? The companies[0] is getting us the first element of the list companies, which is itself a list. Then we’re getting the second element of that list, so the element at index 1, i.e., the string ' Facebook'.

However, to iterate through these lists, you'll need to spice things up a bit.

Enter: **nested loops**. These are loops that can be used inside another loop!

These nested loops look like this:

for x in [1, 2, 3]:  
    for y in [4, 5, 6]:  
        print(x \* y)

This outputs:

4

5

6

8

10

12

12

15

18

Let's think a little about what the Python interpreter is actually doing when this is executed.

* First, it locks in on the first element of the outer list [1, 2, 3], namely, 1.
* It then gets the first element of the inner list [4, 5, 6], namely, 4.
* It then multiplies 1 by 4 to get 4, and prints it.
* Then the inner loop iterates to the next element: 5.
* The Python interpreter then multiplies 1 by 5 to get 5, and prints it.
* Then the inner loop iterates to its final element, 6.
* The Python interpreter then multiplies 1 by 6 to get 6, and prints it.
* Since all elements in the inner loop have been iterated through, the outer loop iterates to its next element: 2.
* It then gets the first element of the inner list, namely, 4.
* It then multiplies 2 by 4 to get 8, and prints it.
* Got it? Makes sense right?

In the tasks below, we're going to make a nested for loop with our companies list that's already in your text editor. To help you with this, let's build up some idea of what we're trying to do, step by step. Consider this code:

for x in range(len(companies)):  
    print(companies[x])

This goes through all the elements of the list companies, all of which are lists, and prints each element. So it outputs:

['Google', 'Facebook', 'Apple']

['Warner Bros. Pictures', '20th Century Fox', 'Universal Pictures']

['Whole foods', 'Starbucks', 'Walmart']

You can test this code out by first commenting out the lines from 6-8 inclusive in your editor (but don't delete them as you'll be filling them in when you do the tasks later).

What do you think the following code does? Try guessing first before checking out the output:

for x in range(len(companies)):  
    print(companies[x])  
    print(len(companies[x]))

The output of this is:

['Google', 'Facebook', 'Apple']

3

['Warner Bros. Pictures', '20th Century Fox', 'Universal Pictures']

3

['Whole foods', 'Starbucks', 'Walmart']

3

That's right: each iteration not only prints each list inside companies, but also gives the number of elements in each list.

From this position of clarity, we're now going to sneak a nested for loop in:

for x in range(len(companies)):    
    print(companies[x])  
    for y in range(len(companies[x])):  
        print(y)

This code outputs:

['Google', 'Facebook', 'Apple']

0

1

2

['Warner Bros. Pictures', '20th Century Fox', 'Universal Pictures']

0

1

2

['Whole foods', 'Starbucks', 'Walmart']

0

1

2

Can you see why? It prints each list, but also the numbers 0-2 inclusive, because the inner for loop is iterating from 0 to 2 inclusive. This seems like a dumb thing to try to print, but it's a stop gap, to make the next Python code more easy to understand. Now check this out:

for x in range(len(companies)):    
    print(companies[x])  
    for y in range(len(companies[x])):  
        print(companies[x][y])

This outputs:

['Google', 'Facebook', 'Apple']

Google

Facebook

Apple

['Warner Bros. Pictures', '20th Century Fox', 'Universal Pictures']

Warner Bros. Pictures

20th Century Fox

Universal Pictures

['Whole foods', 'Starbucks', 'Walmart']

Whole foods

Starbucks

Walmart

So here we're doing something sensible: we're printing each list contained in companies, and also all the members of each company.

It's only one small step to printing out the character-length of each company in each list in companies. Can you figure out what it is before looking at the code below?

for x in range(len(companies)):    
    print(companies[x])  
    for y in range(len(companies[x])):  
        print(len(companies[x][y]))

This outputs:

['Google', 'Facebook', 'Apple']

6

8

5

['Warner Bros. Pictures', '20th Century Fox', 'Universal Pictures']

21

16

18

['Whole foods', 'Starbucks', 'Walmart']

11

9

7

You've got everything you need to hit the challenge now, so uncomment lines 6-8 inclusive. Good luck!

Tasks

Complete the inner for-loop by providing a good in range argument.

Click the checkbox above to attempt this task.

Checks

Code PatternIncomplete

Complete inner for-loop

Attempt this task to view the check results.

Replace the underscore with the length of the particular company in the str part of the print statement.

Click the checkbox above to attempt this task.

Checks

Code PatternIncomplete

Complete print statement

Attempt this task to view the check results.

After filling the blanks in the code, run this check to see if your implementation works correctly.

Click the checkbox above to attempt this task.

Checks

Test CaseIncomplete

Checking the output from the nested loops

Attempt this task to view the check results.

string\_1 = "Data Science is future!"

string\_2 = "Everybody can learn programming!"

string\_3 = "You will learn how to program with Python"

#Do not change code above this line

"""print(len(string\_1))

print(len(string\_2))

string\_2 = "Python programming language"

print(len(string\_2))

print(string\_2.index('o'))

#print(string\_1.index('i))

print(string\_3.index('P'))

counter =0

string\_3 = "Being a Data Scientist has provided great opportunities for me!"

for char in string\_3:

    if (char == 'o'):

        counter += 1

print(counter)

string\_3 ='Being a Data Scientist has provided great opportunities for me!'

#print(string\_3.count('o'))

 #print the index within string\_3 of the first character of the substring 'ovi'

print(string\_3.index('ovi'))

# print a count of the occurrences of 'ovi' in string\_3

print(string\_3.count('ovi'))

print(string\_1.count('u'))

print(string\_2[14:19])

string\_4 = "Hello World!"

print(string\_4.upper())

print(string\_4.lower())

print(string\_1.lower())

print(string\_2.upper())

print(string\_1.split(" "))

print(string\_2.split(" "))

welcome = 'Hello Harry! How are you today Harry? Very nice to see you, Harry.'

welcome = welcome.replace('Harry', 'Jerry')

print(welcome)"""

print(string\_1.startswith('Data'))

print(string\_2.endswith('Python'))

famousQuotes = "Be not afraid of greatness. Some are born great, some achieve greatness, and some have greatness thrust upon 'em."

print(famousQuotes.replace('greatness', 'sausages'))

3.1.4.2 Break statement

There's an important difference between **iterating with a range() function**, and **iterating just with the elements of a list**: To illustrate this, suppose we have three pets, and we have three kinds of snack. We want to give one snack from each type to each of our pets. We can do this with code as follows:

guys = ['Giraffe', 'Monkey', 'Panda']  
snacks = ['Peanuts', 'Bananas', 'Jelly Babies']  
for guy in range(len(guys)):  
    for snack in range(len(snacks)):  
        print('Gave ' + snacks[snack] + ' to ' + guys[guy])

This code outputs:

Gave Peanuts to Giraffe

Gave Bananas to Giraffe

Gave Jelly Babies to Giraffe

Gave Peanuts to Monkey

Gave Bananas to Monkey

Gave Jelly Babies to Monkey

Gave Peanuts to Panda

Gave Bananas to Panda

Gave Jelly Babies to Panda

Which is exactly what we wanted! In the above, our dummy variables guy and snacks store numbers; in fact, integers. In the above, we are *iterating with the range() function*.

But now look at this. We can output exactly the same thing with this code:

guys = ['Giraffe', 'Monkey', 'Panda']  
snacks = ['Peanuts', 'Bananas', 'Jelly Babies']  
for guy in guys:  
    for snack in snacks:  
        print('Gave ' + snack + ' to ' + guy)

See the difference? Here, our dummy variables guy and snacks store elements of the lists, so strings! Here, we are *iterating just with the elements of lists*.

Of course, we might not want to give *every* snack to every pet. We can use the keywords break and continue to avoid this. Suppose our pet giraffe is violently allergic to bananas, so if our feeding distribution system is about to try to give bananas to her, we need to skip to the next snack. We can build that in like this:

guys = ['Giraffe', 'Monkey', 'Panda']  
snacks = ['Peanuts', 'Bananas', 'Jelly Babies']  
for guy in guys:  
    for snack in snacks:  
        if (guy == 'Giraffe' and snack == 'Bananas'):  
            continue  
        else:  
            print('Gave ' + snack + ' to ' + guy)

This code outputs:

Gave Peanuts to Giraffe

Gave Jelly Babies to Giraffe

Gave Peanuts to Monkey

Gave Bananas to Monkey

Gave Jelly Babies to Monkey

Gave Peanuts to Panda

Gave Bananas to Panda

Gave Jelly Babies to Panda

So when we get to Giraffe, we actually don't give her peanuts. But this doesn't stop all the other pets from getting all the snacks, nor does it stop Giraffe from getting Bananas and Jelly Babies.

Now imagine that our monkey got into the pantry an hour ago and absolutely stuffed his little face with peanuts. If we give him any more food he will explode. So we want our feeding distribution system to ignore him:

for guy in guys:  
    for snack in snacks:  
        if (guy == 'Giraffe' and snack == 'Bananas'):  
            continue  
        elif (guy == 'Monkey'):  
            break  
        else:  
            print('Gave ' + snack + ' to ' + guy)

This code outputs exactly what we want now:

Gave Peanuts to Giraffe

Gave Jelly Babies to Giraffe

Gave Peanuts to Panda

Gave Bananas to Panda

Gave Jelly Babies to Panda

We're ignoring Monkey, we're not giving Giraffe peanuts, and we're giving Panda everything.

What's nice about this example is the code is actually pretty complex structurally! We have a nested for loop, and an if/elif-else statement nested within that. But it's hopefully not too tricky to understand.

In sum, using a break statement terminates the loop that contains the statement. When you use the break statement, control of the program moves to the statement immediately after the body of the loop. If the break statement is inside a nested loop, the break will terminate the innermost loop. Using a continue statement just skips to the next iteration.

Why do break and continue commands help so much? Well, suppose for example that we're dividing a given number with a list of numbers using a range() function. We can skip the division where we're trying to divide that number by 0 (which would be mathematically impossible) by using continue.

Try it out for yourself!

**Important note:** When you're done with this exercise, please comment out the nested loop that you've added in to your code.

Tasks

Using the y- loop that you created in the previous step, add an if statement that checks if y is equal to one. If that is the case, break out the inner loop.

Click the checkbox above to attempt this task.

Checks

Code PatternIncomplete

Adding the break point in the if statement

Attempt this task to view the check results.

Code PatternIncomplete

Adding if statement to the innermost loop

Attempt this task to view the check results.

Change the break statement that you wrote in the previous step to continue and take a look at the output.

**Please note**: Leave y == 1, just as it was in the previous step.

Click the checkbox above to attempt this task.

Checks

Code PatternIncomplete

Add continue statement in y loop

Attempt this task to view the check results.

[Skip Navigation](https://next.tech/projects/d5a61838-c8fd-43bb-8860-fad8d2623757?access_token=FE7B0C464AB03ECA4731CA6F337DF84C&internal=false&step_id=a6b38154-f397-44d4-9c23-13130bd45366#left-sidebar)

3.1.4.3. While loops

Instead of iterating something with a for loop, you can use a while loop to iterate - as long as a specified condition is True. We've seen this a little in some examples already, but here we want to fully understand its syntax and get hands on with it.

To better understand this concept, take a look at this example. It features a program that asks the user to enter a specific word requested by the program:

word = 'snake'  
usr\_word = input('Type the word ' + word + ':')  
while usr\_word != word:  
    usr\_word = input('Try again!: ')  
print('Correct!')

This program uses a while loop to check that the word entered matches the requested word. If the word doesn't match, the program asks the user to try again until the words match up, at which point the loop ends.

Now, check out this number guessing game using a while loop:

secret\_number = 23  
guess = None #None is a special word in python that represents that something has no value.  
while guess != secret\_number:  
    guess = input('Enter a number: ')  
    guess = int(guess) #int() changes the data type to an integer. In this case the data is being changed from a string.  
    if guess > secret\_number:  
        print('Too high.')  
    if guess < secret\_number:  
        print('Too low.')  
print('You got it!')

This program will keep asking the user to guess a number until they guess the correct number. When they do, the while loop will exit (because guess will be equal to secret\_number) and the final print will be called.

**Important note:** If you added either of the example while loops above to your own code, make sure you remove them or comment them out before starting on the tasks for this exercise.

Sometimes, you'll run into a while-loop that runs forever and you'll need to stop the code with force. This can happen if we developers don't take care of our counters and conditions when defining the while-loop.

**First example:**

while True:

print("Hello World!")

The first example is pretty obvious. We explicitly said only True, so this loop will run **forever**.

**Second example:**

while 1:

print("Hello World!")

The second example isn't as obvious. We only said **1**, and in Python **1** means True as well, so this loop will run **forever**.

**Third example**:

counter = 1

while counter < 10:

print("Hello World!")

The third case is the subtlest of all! While this is a correct implication, we missed one crucial part — we need **to increase the value of the counter inside the while loop!** In this example, our **counter** always stays at the value 1, so it will be less than 10 **forever**.

The last case is the most common use ofwhile loops, so be sure to be extra careful when using and defining while loops.

NOTE: If you want to increase a value of variable x, there are two ways to accomplish that:

1. x = x + some\_number
2. x += some\_number

Tasks

Define two necessary variables for our while loop; start\_number and end\_number.

1. start\_number variable should be set to value of 0
2. end\_number variable should be set to value of 10

Click the checkbox above to attempt this task.

Checks

Code PatternIncomplete

Define the variable end\_number and set its value to 10

Attempt this task to view the check results.

Code PatternIncomplete

Define the variable start\_number and set its value to 0

Attempt this task to view the check results.

Create a while loop that will run as long as start\_number does not equal end\_number.

Click the checkbox above to attempt this task.

Increment start\_number by 1 (this means add 1 to the start\_number). We need to increment it every time we loop.

Click the checkbox above to attempt this task.

Checks

Code PatternIncomplete

Increase start\_number by one

Attempt this task to view the check results.

Print the current\_value of start\_number inside the while loop.

Click the checkbox above to attempt this task.

Checks

Test CaseIncomplete

Prints 1 through 9

Attempt this task to view the check results.

Code PatternIncomplete

Prints numbers from 1 to 9

Attempt this task to view the check results.

NOTE: If you want to increase a value of variable x, there are two ways to accomplish that:

1. x = x + some\_number
2. x += some\_number

Tasks

Define two necessary variables for our while loop; start\_number and end\_number.

1. start\_number variable should be set to value of 0
2. end\_number variable should be set to value of 10

Awesome work!

Checks

Code PatternComplete

Define the variable end\_number and set its value to 10

Description

Searched your code for a specific pattern:

end\_number\s\*=\s\*10

You can learn more about regular expressions [here](https://ruby-doc.org/core-2.1.1/Regexp.html).

Code PatternComplete

Define the variable start\_number and set its value to 0

Description

Searched your code for a specific pattern:

start\_number\s\*=\s\*0

You can learn more about regular expressions [here](https://ruby-doc.org/core-2.1.1/Regexp.html).

Create a while loop that will run as long as start\_number does not equal end\_number.

Well done!

Increment start\_number by 1 (this means add 1 to the start\_number). We need to increment it every time we loop.

Way to go!

Checks

Code PatternComplete

Increase start\_number by one

Description

Searched your code for a specific pattern:

(start\_number\s\*=\s\*start\_number\s\*\+\s\*1|start\_number\s\*\+=\s\*1)

You can learn more about regular expressions [here](https://ruby-doc.org/core-2.1.1/Regexp.html).

Print the current\_value of start\_number inside the while loop.

Great work!

Checks

Test CaseComplete

Prints 1 through 9

Input

Output

1

2

3

4

5

6

7

8

9

10

Results

1

2

3

4

5

6

7

8

9

Expected Output

1

2

3

4

5

6

7

8

9

0

Code PatternComplete

Prints numbers from 1 to 9

Description

Searched your code for a specific pattern:

print\(\s\*start\_number\s\*\)

You can learn more about regular expressions [here](https://ruby-doc.org/core-2.1.1/Regexp.html).

3.1.5.1. Python functions (1)

We've talked a little about the difference between functions and methods in this course, for example in the Computational Thinking resource. Let's recap this and make sure the distinction is absolutely clear.

Thinking in an Object-Oriented way makes programming a great deal easier. Objects in programming are very similar to objects in the actual world.

Take this **lamp** for example. The lamp is composed of various colors, has dimensions and weight, and features various other characteristics you could use to describe it. There's also what the lamp can **do**- for example, turn on and off. These actions that can be performed by or with a lamp (in this case, turning on and turning off) are **methods**.

In programming languages, **every object is an instance of a class, and that class determines that object's abilities (methods) and features (variables)**. Let's look at an example using *strings*. Almost everything in the Python language is an object, from integers, to strings, to custom-made objects. All of these objects have methods and variables assigned to them.

When you worked with strings, you used a variety of *methods*, including **lower()**, **upper()**, **startswith()**, etc. Any ability of an object is a method of it.

HINT: This exercise features a method you haven't seen before: capitalize() You can learn more about this method [here](https://www.programiz.com/python-programming/methods/string/capitalize).

Tasks

Use the capitalize() method on the given string ('sample') inside the print statement.

This method takes a string and capitalizes its first character.

Feedback

Click the checkbox above to attempt this task.

Checks

Code PatternIncomplete

Checking if a resulting string is hard-coded.

Attempt this task to view the check results.

Test CaseIncomplete

Checking if a string is capitalized

Attempt this task to view the check results.

Code PatternIncomplete

Using the .capitalize() method

Attempt this task to view the check results.

Use the replace() method on the given string inside the print statement.

The replace() method takes two arguments. The first is a part of a string that should be changed and the second is a string that will place an original piece of the code.

Your task is to change fox inside the original string to box.

Feedback

Click the checkbox above to attempt this task.

Checks

Code PatternIncomplete

Checking if the string is hard-coded or the method replace is used.

Attempt this task to view the check results.

Test CaseIncomplete

Checking if the word fox has been changed to box in the string

Attempt this task to view the check results.

Code PatternIncomplete

Changing fox into a box

Attempt this task to view the check results.

thod [here](https://www.programiz.com/python-programming/methods/string/capitalize).

Tasks

Use the capitalize() method on the given string ('sample') inside the print statement.

This method takes a string and capitalizes its first character.

Feedback

Fantastic job!

Checks

Code PatternComplete

Checking if a resulting string is hard-coded.

Description

Searched your code for a specific pattern:

Red fox is a fire fox

You can learn more about regular expressions [here](https://ruby-doc.org/core-2.1.1/Regexp.html).

Test CaseComplete

Checking if a string is capitalized

Input

Output

Red fox is a fire fox

Results

Red fox is a fire fox

Expected Output

Red fox is a fire fox

0

Code PatternComplete

Using the .capitalize() method

Description

Searched your code for a specific pattern:

sample\.capitalize\s\*\(\s\*\)

You can learn more about regular expressions [here](https://ruby-doc.org/core-2.1.1/Regexp.html).

Use the replace() method on the given string inside the print statement.

The replace() method takes two arguments. The first is a part of a string that should be changed and the second is a string that will place an original piece of the code.

Your task is to change fox inside the original string to box.

Feedback

Excellent work!

Checks

Code PatternComplete

Checking if the string is hard-coded or the method replace is used.

Description

Searched your code for a specific pattern:

red box is a fire box

You can learn more about regular expressions [here](https://ruby-doc.org/core-2.1.1/Regexp.html).

Test CaseComplete

Checking if the word fox has been changed to box in the string

Input

Output

Red fox is a fire fox red box is a fire box

Results

red box is a fire box

Code PatternComplete

Changing fox into a box

Description

Searched your code for a specific pattern:

sample\.replace\(\s\*["']fox["']\s\*,\s\*["']box["']\s\*\)

You can learn more about regular expressions [here](https://ruby-doc.org/core-2.1.1/Regexp.html).

3.1.5.2. (2) Python functions

Let's cement our knowledge of functions. In the programming world, a **function** is a sequence of commands that execute in a certain order. They let us implement reusable code.

Functions are independent of objects. To explain this statement, let's take another look at the lamp from the previous exercise. It had some methods connected to it (specifically turning on and turning off) but we couldn't use a method to move the lamp from the living room to the bedroom.

Instead, to perform this operation, we'd need to create a function called move(). This function would look something like this:

move(lamp):  
    take the lamp  
    move to bedroom  
    put the lamp down

Note of course that this is [*Pseudocode*](https://en.wikipedia.org/wiki/Pseudocode) not executable code.

Just like when you learned about methods in the last exercise, let's also look at how functions interact with strings in the programming world.

If you want to check to see if a fifth character in a string is equal to "D", you'd need to create a function to perform that check.

Why don't you do just that by working through the tasks listed below?

Tasks

Define a function called check\_fifth\_char that takes one argument, which is called string.

Feedback

Click the checkbox above to attempt this task.

Checks

Code PatternIncomplete

Define function called check\_fifth\_char

Attempt this task to view the check results.

Inside the function check\_fifth\_char check if the given string's fifth character is an uppercase "D". If the string's fifth character is an uppcase D, return True, otherwise return False.

**Helpful hint:** Remember, the first character/element has an index of zero (0).

Feedback

Click the checkbox above to attempt this task.

Checks

Code PatternIncomplete

Check if a fifth char is equal to upper case D

When you are done implementing the check\_fifth\_char function, run this check. This will automatically check your function for both cases.

Feedback

Click the checkbox above to attempt this task.

Checks

Unit TestIncomplete

Testing the check\_fifth\_char function when the string has D at the index 4.

Attempt this task to view the check results.

Unit TestIncomplete

Test if the function check\_fiftt\_char returns False when the input string doesn't have D at the index 4.

Attempt this task to view the check results.

So the correct implementation of our function was:

def check\_fifth\_char(string):  
    if string[4] == 'D':  
        return True  
    else:  
        return False

But suppose we came across a different function, like this one:

def custom(string):  
    string2 = string[5:]  
    if string2[2] == 't':  
        return True  
    else:  
        return False

And we call this function with the following string:

print(custom('qwertyfgtcva'))

Now this code outputs: False, but unless we're hot on our Python string manipulation, it can be hard to know why.

What we want to emphasize here is **how to debug functions**. The best way of doing this is printing out the values of variables within those functions. So if we write:

def custom(string):  
    string2 = string[5:]  
        print(string2[2])  
    if string2[2] == 't':  
        return True  
    else:  
        return False

We get the output:

yfgtcva

g

False

We now know what both string[5:] and string2[2] have given us: yfgtcva and g respectively. It's now no surprise at all why this function has returned False here.

3.1.6.1. Creating a text file with Python

In section 2.3, we saw how to get user inputs with Python. In this lesson, we'll look at creating files with Python and interacting with them.

First off, let's take a look at some functions and how they relate to files.

open() is a built-in function that's used to (you guessed it!) open a file.

When you use the open function, it will return a file object.

File objects have methods and attributes, like any other object that you've worked with before; these methods and attributes can be used to manipulate the file you're working with.

When you use the open function, you'll need to provide two arguments. The first argument is the name of the file that's about to be opened; the second argument is a mode. There are a bunch of mode variations you can use, but the modes that are important to this course include:

‘r’ – **Read mode** is used when you're only reading the file

'w’ – **Write mode** is used to edit and write new information to the file (any existing files with the same name will be erased when this mode is activated)

‘a’ – **Appending mode** is used to add new data to the end of the file. This new information is automatically amended to the end

‘r+’ – **Special read and write mode** is used to handle both reading and writing actions when working with a file

These are all *keywords*, with inherent functionality, and can't be changed. Save [this resource](https://www.programiz.com/python-programming/methods/built-in/open) for later on the open() function.

So, how can use all this information to create a totally new file? You're about to find out!

Creating a new, empty file in Python is a breeze. Simply open a new file using the **write** mode like so:

file = open('new\_file.txt', 'w')

The above line will create a file with the name new\_file, which will have a .txt extension.

When working with files, it's very important to **close** the file when you're done working with it. Here's how you do this:

file = open('new\_file.txt','w')

file.close()

To close a file simply write .close().

In the above examples, we used file as the name of the file object but you can name your file objects whatever you want.

Now, try to create your own file! Note: the Next Tech interface supports a file system with files listed or tabs showing each of the files.

Tasks

Using the open() function, create your first file, which should be called test\_file.txt.

Note: Please store your file in a variable called file

Click the checkbox above to attempt this task.

Checks

Code PatternIncomplete

Create first file

Attempt this task to view the check results.

Using the close() method, close the opened file that you named file\_test.txt

Click the checkbox above to attempt this task.

Checks

Code PatternIncomplete

Close file

Attempt this task to view the check results.

"""

sample = 'red fox is a fire fox'

#Do not change code above this line.

capitalized\_string = sample.capitalize()

replace\_string = sample.replace('fox', 'box')

capital\_replace\_string = sample.capitalize().replace('fox', 'box')

print(capitalized\_string)

print(replace\_string)

print(capital\_replace\_string)

def check\_fifth\_char(string):

    if string[4] == 'D':

        return True

    else:

        return False

print(check\_fifth\_char('bandD'))

def custom(string):

    string2 = string[5:]

    if string2[2] == 't':

        return True

    else:

        return False

print(custom('qwertyfgtcva'))"""

def custom(string):

    string2 = string[5:]

    print(string2[2])

    if string2[2] == 't':

        return True

    else:

        return False

    print(custom('qwertygtcva'))

A program is of little use if it can't interact with the rest of the world. A useful program needs to accept inputs and produce outputs. The easiest way for a program to interact with the world is through a terminal. "Standard input" refers to accepting input from the terminal, whereas "standard output" refers to printing output to the terminal. In this lesson, you'll learn some basic Python functions in order to read inputs and write outputs.

3.1.6.1. Creating a text file with Python

In section 2.3, we saw how to get user inputs with Python. In this lesson, we'll look at creating files with Python and interacting with them.

First off, let's take a look at some functions and how they relate to files.

open() is a built-in function that's used to (you guessed it!) open a file.

When you use the open function, it will return a file object.

File objects have methods and attributes, like any other object that you've worked with before; these methods and attributes can be used to manipulate the file you're working with.

When you use the open function, you'll need to provide two arguments. The first argument is the name of the file that's about to be opened; the second argument is a mode. There are a bunch of mode variations you can use, but the modes that are important to this course include:

‘r’ – **Read mode** is used when you're only reading the file

'w’ – **Write mode** is used to edit and write new information to the file (any existing files with the same name will be erased when this mode is activated)

‘a’ – **Appending mode** is used to add new data to the end of the file. This new information is automatically amended to the end

‘r+’ – **Special read and write mode** is used to handle both reading and writing actions when working with a file

These are all *keywords*, with inherent functionality, and can't be changed. Save [this resource](https://www.programiz.com/python-programming/methods/built-in/open) for later on the open() function.

So, how can use all this information to create a totally new file? You're about to find out!

Creating a new, empty file in Python is a breeze. Simply open a new file using the **write** mode like so:

file = open('new\_file.txt', 'w')

The above line will create a file with the name new\_file, which will have a .txt extension.

When working with files, it's very important to **close** the file when you're done working with it. Here's how you do this:

file = open('new\_file.txt','w')

file.close()

To close a file simply write .close().

In the above examples, we used file as the name of the file object but you can name your file objects whatever you want.

Now, try to create your own file! Note: the Next Tech interface supports a file system with files listed or tabs showing each of the files.

Tasks

Using the open() function, create your first file, which should be called test\_file.txt.

Note: Please store your file in a variable called file

Click the checkbox above to attempt this task.

Checks

Code PatternIncomplete

Create first file

Attempt this task to view the check results.

Using the close() method, close the opened file that you named file\_test.txt

Click the checkbox above to attempt this task.

Checks

Code PatternIncomplete

Close file

Attempt this task to view the check results.

[Skip Navigation](https://next.tech/projects/6acf331c-dbf3-445a-a1c2-661436c468f6?access_token=B891777B16DAD92A572DB771B06FE622&internal=false&step_id=a2234826-4f4a-4c52-845c-fb87a10e688a#left-sidebar)

3.1.6.2. Writing to a file

In the previous exercise, you learned how to create a new file with Python. Let's build on what you've learned by taking a look at how you can write something to your file.

To write something to a file, you'll need to use the .write() method and add what you want to write to the file as its argument.

Take a look:

file = open('new\_file.txt', 'w')

file.write('I love Python')

file.close()

In the above example, we opened a file, wrote to it by using the .write() method and included a string inside the method's brackets (in this case, the string was I love Python.) When we were done with all that, we closed the file (remember, this last step is super important!)

Your turn!

Task

Using the write() method, write anything you want to the test\_file.txt that you created in the previous step.

Click the checkbox above to attempt this task.

Checks

Code PatternIncomplete

Write to the file

Attempt this task to view the check results.

3.1.6.3. Reading from a file with read() method

Once you have a file with data in it, you can read directly from it directly to the Python console. (Yes, we realize this sentence sounds grammatically weird, but in the world of Python it makes sense.)

To read from a file, the first thing you'll have to do is change modes from write to read.

After you open the file you want to read in the read mode, you can approach the actual reading of the data in several different ways. The most basic of these approaches involves using the read() method.

The output of the read() method will display **all** the text inside the file.

Take a look at this example:

file = open('new\_file.txt', 'r')

print(file.read())

file.close()

Output: I love Python

Another way to use the read() method is to specify the number of characters you want to have read!

Check out this way of using the read() method here:

file = open('new\_file.txt', 'r')

print(file.read(6))

file.close()

Output: I love

Ready to use the read() method on your own code? Let's get to it.

Tasks

Open the test\_file.txt in reading mode and store it in a variable called file

Click the checkbox above to attempt this task.

Checks

Code PatternIncomplete

Open file in the reading mode

Attempt this task to view the check results.

Using the read() method, read everything that you have wrote in test\_file.txt and print that to the console using the print() method.

Click the checkbox above to attempt this task.

Checks

Code PatternIncomplete

Reading from files - part 1

Attempt this task to view the check results.

Using the close() method, close the opened file that you named file\_test.txt

Well done!

Checks

Code PatternComplete

Close file

Description

Searched your code for a specific pattern:

file.close\s\*\(\s\*\)

[Skip Navigation](https://next.tech/projects/6acf331c-dbf3-445a-a1c2-661436c468f6?access_token=B891777B16DAD92A572DB771B06FE622&internal=false&step_id=cd8a345f-6b52-480a-ae2f-549e218c197c#left-sidebar)

3.1.6.4. Reading from a file using readline() method

While reading the entire text of a file is all well and good, sometimes you'll want to read a file line-by-line. When these instances arise, you'll use the readline() method.

The readline() method will return a string of characters that contain a single line of text or information from the provided file.

Take a look:

file = open('new\_file.txt', 'r')

print(file.readline())

file.close()

Output: I love Python

The output is the same as the read method because our file only contains one line. However, if you were working with a file that had more than one line, you could use the readline method to specify the exact line that you to have read, too.

Try this method for yourself!

Task

Print out the first line from the test\_file.txt using the .readline() method.

Click the checkbox above to attempt this task.

Checks

Code PatternIncomplete

Using the readline() method

Attempt this task to view the check results.

[Skip Navigation](https://next.tech/projects/6acf331c-dbf3-445a-a1c2-661436c468f6?access_token=B891777B16DAD92A572DB771B06FE622&internal=false&step_id=be76305c-1729-4f93-bdb6-e7964caa1ca4#left-sidebar)

3.1.6.5. For loop and file reading

Alright — you've learned how to read a whole file, and you've learned how to read certain lines from a file. There's only one more reading method to go — the readlines() method. Note the s at the end.

The readlines() method will return all the text in a file as a list, where each item will be written out as its own line.

Take a look:

file = open('new\_file.txt', 'r')

lines = file.readlines()

print(lines)

file.close()

Output: ["I love Python", ]

Now that we have a list of lines, it'll be easy to iterate through it with a for-loop if the project you're working on calls for it. You can see an example of this here:

file = open('new\_file.txt', 'r')

lines = file.readlines()

for line in lines:

print(line)

file.close()

Output: I love Python

Try using a for-loop to read from your own file.

Tasks

Using the open() function, open the test.txt file in reading mode and store it in the variable called file\_2

**Please note:** We've created this file for you so there's text that already exists inside it.

Click the checkbox above to attempt this task.

Checks

Code PatternIncomplete

Open test.txt

Attempt this task to view the check results.

Using the readlines() method, get all the lines from the test.txt file.

Assign this to a variable.

Click the checkbox above to attempt this task.

Checks

Code PatternIncomplete

Geting all lines in the file

Attempt this task to view the check results.

Using a for - loop, iterate through the lines and print each of them to the terminal.

**Note:** Please name the counter in the for - loop : line

Click the checkbox above to attempt this task.

Checks

Code PatternIncomplete

Read line by line

Attempt this task to view the check results.

Using the close() method, close the file\_2 file.

Click the checkbox above to attempt this task.

Checks

Code PatternIncomplete

Closing file\_2

Attempt this task to view the check results.

3.1.6.6. Appending to a file

Once you've filled a file, you can append it in order to change what's in the file. Appending generally means *adding* to the end of something; for example when we appended elements to a list, we added them to the end of the list.

Appending to a file is really similar to writing in the file - you just use a different mode to open the file.

If you want to append to a file, you need to open it in a+ mode, rather than the w mode.

Take a look:

file = open('new\_file.txt', 'a+')

file.write('Appended line 1')

file.write('Appended line 2')

file.close()

Try appending your own file after you read the note below!

**Please note:**

If you try to perform **writing/reading** or even **appending** to a file that has been closed, Python will break out and stop the execution of your code. The error you'll see if this happens is: I/O operation on closed file Check out this example:

file = open('new\_file.txt', 'a+')

file.close()

file.write('Appended line 2')

Output: ValueError: I/O operation on closed file.

Luckily, this error can be easily fixed — look around for the line where you closed your file ( look forfile.close()) and move it **below** the line where you try to read/write or append to that file.

Tasks

Using the a+ flag, open your file test\_file.txt in appending mode.

Note: Please store the open file in a variable called file

Click the checkbox above to attempt this task.

Checks

Code PatternIncomplete

Open file for appending

Attempt this task to view the check results.

Using the write() method, append a line at the end of test\_file.txt.

Click the checkbox above to attempt this task.

Checks

Code PatternIncomplete

Appending to the test\_file.txt

Attempt this task to view the check results.

Using the close() method, close the opened file that you named file\_test.txt

Well done!

Checks

Code PatternComplete

Close file

Description

Searched your code for a specific pattern:

file.close\s\*\(\s\*\)

You can learn more about regular expressions [here](https://ruby-doc.org/core-2.1.1/Regexp.html).

A program is of little use if it can't interact with the rest of the world. A useful program needs to accept inputs and produce outputs. The easiest way for a program to interact with the world is through a terminal. "Standard input" refers to accepting input from the terminal, whereas "standard output" refers to printing output to the terminal. In this lesson, you'll learn some basic Python functions in order to read inputs and write outputs

"""

file = open('test\_file.txt', 'w')

file  = open(file\_test.txt, 'w')

file.close()

file = open('test\_file.txt', 'w')

file.write('Welcome to Springboard School')

file.close()

file = open('new\_file.txt', 'w')

file.write('I love Python')

file.close()

file = open('new\_file.txt', 'r')

print(file.read())

file.close()

file = open('new\_file.txt', 'r')

print(file.read(6))

file.close()

file = open('test\_file.txt', 'r')

file.close()

file = open('test\_file.txt', 'w')

file.write('Hello World')

file.close()

file = open('test\_file.txt', 'r')

print(file.read())

file.close()

file = open('new\_file.txt', 'r')

print(file.readline())

file.close()

file = open('test\_file.txt', 'r')

print(file.readline())

file.close()

file = open('new\_file.txt', 'r')

lines = file.readlines()

print(lines)

file.close()

file = open('new\_file.txt', 'r')

lines = file.readlines()

for line in lines:

    print(line)

file.close()

file\_2 = open('test.txt', 'r')

lines = file\_2.readlines()

for line in  lines:

    print(line)

file\_2.close()"""

file = open('test\_file.txt', 'a+')

file.write('Appended line 1')

file.close()

Ready to dive into your next hands-on project? This project takes things up a notch and asks you to apply your intermediate Python skills. As a reminder, these hands-on projects are designed with the Technical Skills Survey in mind. Be sure to reach out to your mentor or your online community if you have any questions as you work through the project steps.

**Submitting Your Work**

Once you've completed your work, submit a link to your mentor. To do this, follow these steps:

1. Click the settings gear in the lower left corner of the Next.Tech page
2. Click "Share"
3. Copy the URL in the popup that appears
4. Share the URL with your mentor.

DSC Prep 3. Intermediate Python Project. 1. Intro

***Birds of Paradise: a Pythonic Photography Adventure***

This project will test:

* dictionaries (and manipulation),
* strings (and manipulation),
* nested loops,
* functions vs methods,
* reading and writing to a file.
* **Note**: This challenge makes use of *binary code*. If you’re not confident on what this is, check out [this resource](https://www.springboard.com/archeio/download/a6e707f1b1e946c3aefec01ce8641263/).

You’re on safari in the most dense, remote part of the South American rainforest as a zoologist.

You’re looking for a number of birds of paradise whose rarity, elusiveness and ferociousness means they’ve never before been photographed.

In order to get good photographs of these birds, you have a photographer with you; she’s carrying a very large and expensive camera. It’s your job to creep ahead of the photographer, using your binoculars intermittently, and, once you see one of the birds, remain perfectly still, and communicate to your photographer: a. Which bird it is, and b. Where the bird is, so she can photograph it.

Your method of communication needs to be fast and perfectly silent. These birds are conditioned to pick out the voices of potential predators easily, and will disappear if they hear you speak. Are you ready? Go to the next page to check out the birds we're looking for.

[Skip Navigation](https://next.tech/projects/e2ec562b-0a0e-4538-830c-95f306a782c4?access_token=B508F9F06CF4A9BB8A3540CE37EBECBC&internal=false&step_id=2eb36d12-cb8a-4de3-8456-aa92fa08f90d#left-sidebar)

DSC Prep 3. Intermediate Python Project. 2. The compendium

Here are the magical birds we want photographs of:

| **Name** | **Height (m)** | **Weight (kg)** | **Colour** | **Endangered** | **Aggressive** |
| --- | --- | --- | --- | --- | --- |
| Gold-crested Toucan | 1.1 | 35 | Gold | True | True |
| Pearlescent Kingfisher | 0.25 | 0.5 | White | False | False |
| Four-metre Hummingbird | 0.6 | 0.5 | Blue | True | False |
| Giant Eagle | 1.5 | 52 | Black and White | True | True |
| Ancient Vulture | 2.1 | 70 | Brown | False | False |

The first task below is to make a dictionary called rarebirds whose keys are the names of the birds above, and whose values are *themselves dictionaries* with the keys: Height (m), Weight (kg), Colour, Endangered, Aggressive. We'll get you started on how this would look, and you can complete it:

rarebirds = {  
    'Gold-crested Toucan': {  
        'Height (m)': 1.1,  
        'Weight (kg)': 35,  
        'Color': 'Gold',  
        'Endangered': True,  
        'Aggressive': True},  
'Pearlescent Kingfisher': {  
        .  
        .  
        .  
    }  
}

Task

Task 1:

Make a dictionary called rarebirds that summarises our data as demonstrated above.

Feedback

Click the checkbox above to complete this task.

DSC Prep 3. Intermediate Python Project. 3. Preparation

We need to make some preparations. Communication with our photographer needs to be lightning fast and silent. All these birds are either very aggressive, endangered, or easily scared and will quickly disappear.

We need to make:

* A list of possible locations
* Codes for those locations that can be communicated silently to our photographer
* A list of possible actions.

Tasks

Task 2:

If you spot a bird, you need a very simple way of telling your photographer where it is.

Make a list called birdlocation with the following 7 elements (made into strings):

* In the canopy directly above our heads.
* Between my 6 and 9 o’clock above.
* Between my 9 and 12 o’clock above.
* Between my 12 and 3 o’clock above.
* Between my 3 and 6 o’clock above.
* In a nest on the ground.
* Right behind you.

Note: provided you use commas in the right places, you can declare a list over multiple lines, which makes its elements easier to read.

The last location is necessary because, for reasons no zoologist yet understands, the terrifying, ancient vulture sometimes stalks potential prey from behind with a fearsome confidence.

Feedback

Click the checkbox above to complete this task.

Task 3:

Because you won’t be able to communicate these descriptions verbally, you need to encode them with **binary code**.

You’ll be using either coin flips, flashlight flashes, or simply Morse code tapped out onto your own hand that your photographer colleague reads. Choose your favourite binary system for this situation: it makes no difference.

Write a dictionary called codes whose keys are the binary codes (of length 3) - written either ‘HHH’ or ‘111’, depending on the system you’ve chosen - and whose values are the members of birdlocation. If this isn't making sense, check out the [binary resource](https://www.springboard.com/archeio/download/a6e707f1b1e946c3aefec01ce8641263/) we've prepared for you.

Note: there are 8 possible binary codes of length 3:

111

110

101

100

011

010

001

000

so you won’t be using one of them. It’s up to you which one you don’t use (but perhaps it’s harder to convey ‘000’ than ‘111’, so ‘000’ is a good candidate for being dropped)!

Task 4:

If we see one of these birds, it'll be easy to panic. Some of them have only been ever spoken about in myths or legends. It's not known, for example, whether the enormous, dinosaur-like Gold-crested Toucan really exists.

As a result, we need a list of actions that will reduce our options for us in moments of panic.

Make a list called actions, containing the following strings: 'Back Away’, 'Cover our Heads','‘Take a Photograph'.

Feedback

Click the checkbox above to complete this task.

You're all set, your equipment is packed. Go to the next page to embark on the journey...

DSC Prep 3. Intermediate Python Project. 4. Entering the jungle

You're in thick, impenetrable jungle; the air is thick and humid, the noise of the insects constant, and the canopy overhead an intoxicating labyrinth of fecund colours.

Your photographer colleague wipes sweat from her brow. She wants to ask you a question.

Tasks

Task 5:

Your photographer has heard that the Giant Eagle has killed adult gorillas before. She wants to know if it's aggressive.

Using your rarebirds dictionary, print out the Giant Eagle’s value for the "Aggressive" key within the nested dictionary (hint: you should be printing out the Boolean True).

Feedback

Click the checkbox above to complete this task.

Task 6:

Your colleague is now nervous, and you’re even more so. She wants the low-down on the rest of the birds you may encounter.

Using a for loop that goes through the keys and the values of your rarebirds dictionary:

* Print out a string specifying the names of all the birds we’re looking for.
* If a bird is aggressive, print out a string advising us to cover our heads. Make use of the actions list in this print statement.
* If a bird is endangered, print out a string advising us to back away. Make use of the actions list in this print statement.

Feedback

Click the checkbox above to complete this task.

Task 7:

Your photographer wants a reminder of what the binary codes that you've come up with mean.

Using a for loop, iterate through the keys and the values of the codes dictionary.

Print out a statement that specifies what each code means.

Feedback

Click the checkbox above to complete this task.

DSC Prep 3. Intermediate Python Project. 5. Midday

An hour has passed. The sun’s risen to its highest point and it’s become swelteringly hot in the jungle. You’re now deep into the habitat of these rare birds and you haven’t seen anything.

Task

Task 8:

Using a for loop, add an extra bird attribute (to go with with height, weight, etc) called 'seen', and set its value to False for all birds.

Feedback

Very well done!

DSC Prep 3. Intermediate Python Project. 6. The Encounter, part 1

A flash of unearthly colour. A sound you've never heard before. Something moving that seems otherworldly, strange, like an alien or an angel. Something’s happening.

The user of your program, ultimately, will get to decide what happens.

This page will get a lot of things happening: each task will be simple, but there'll be quite a few of them. Keep at it.

Tasks

Task 9:

Make a new variable called encounter, and set its value to True.

Feedback

Excellent work!

Task 10:

Ask the user for an input, and give them the prompt: ‘What do you see?’. Store this input in a variable called sighting.

Make this input all lowercase with a built-in function.

Feedback

Well done.

Task 11:

Make a list called rarebirdsList of the names of the rare birds, using your dictionary rarebirds and the built-in method keys().

Feedback

Keep it up! :)

Task 12:

Write some code that checks whether the sighting is in rarebirdsList. If it isn’t, print out the statement: “that’s not one of the birds we’re looking for”.

If it is, print out the statement: “this is one of the birds we’re looking for!”

Feedback

Great work!

[Skip Navigation](https://next.tech/projects/e2ec562b-0a0e-4538-830c-95f306a782c4?access_token=37C2AD67566F90422AABADF665A60D43&internal=false&step_id=5a310f87-78bd-45f3-bcc6-76505d860672#left-sidebar)

DSC Prep 3. Intermediate Python Project. 7. The Encounter, part 2

You're bristling with excitement. You need to find out the location of the sighting from the user, and call upon the code system you came up with earlier to communicate to your photographer where the sighting is.

Tasks

Task 13:

Ask the user for another input, with the prompt: ‘Where do you see it? Input the correct code.’, and store this input in a variable called code.

Feedback

Well done! Keep going...

Task 14:

Make a new variable called location that gets the correct location from the dictionary codes using the key code.

You can do this in a single assignment, looking like:

location = codes[code]

Feedback

Well done!

Task 15:

Make a printout summarizing which bird has been seen and its location, making use of the variables sighting and location.

This could look like:

print("So you've seen a", sighting, location, "My goodness.")

Feedback

Excellent, keep plugging away!

[Skip Navigation](https://next.tech/projects/e2ec562b-0a0e-4538-830c-95f306a782c4?access_token=37C2AD67566F90422AABADF665A60D43&internal=false&step_id=aea639c3-8283-40e0-85ac-7943d21898f2#left-sidebar)

DSC Prep 3. Intermediate Python Project. 8. The Aftermath

You're almost there - you've done fantastically so far. Just a couple more tasks and you'll be out of the jungle and home to your family.

Tasks

Task 16:

Let's do some more logic. Make an if, elif, and else statement.

In the if-statement, check whether the sighting is aggressive. If it is, print out that it’s aggressive, and that we need to back away and cover our heads. In addition, print out that we need to photograph the sighting at its location.

In the elif-statement, check whether the sighting is endangered. If it is, print out that it’s endangered, and that we need to back away. Also, print out that we need to photograph the sighting at its location.

In the else statement (i.e, the sighting is neither aggressive nor endangered) print out that we need to photograph the ultra rare sighting at its location.

In all of these blocks, make use of the actions list and the variables sighting and location in your printouts.

Feedback

Excellent work!

Task 17:

Using your encounter variable within a big while-loop, modify the above code so that input is repeatedly requested from the user until they input one of the birds in our rarebirds dictionary.

Hint: start the encounter by making encounter’s value True. Make encounter’s value False within the nested if, elif and else statements.

Feedback

Excellent work.

rarebirds = {

    'Gold-crested Toucan': {

        'Height (m)': 1.1,

        'Weight (kg)': 35,

        'Color': 'Gold',

        'Endangered': True,

        'Aggressive': True},

    'Pearlescent Kingfisher': {

        'Height (m)':0.25,

        'Weight (kg)': 35,

        'Color': 'white',

        'Endangered': False,

        'Aggressive': False},

    'Four-metre Hummingbird': {

        'Height (m)': 0.6,

        'Weight (kg)': 0.5,

        'Color': 'Blue',

        'Endangered': True,

        'Aggressive': False},

    'Giant Eagle': {

        'Height (m)': 1.5,

        'Weight (kg)': 52,

        'Color': 'Black with White',

        'Endangered': True,

        'Aggressive': True},

    'Ancient Vulture': {

        'Height (m)': 2.1,

        'Weight (kg)': 70,

        'Color': 'Brown',

        'Endangered': False,

        'Aggressive': False}

}

birdlocation = [("a","In the canopy above our heads."),

                ("b", "between my 6 and 9 o'clock above."),

                ("c", "between my 9 and 12 o'clock above."),

                ("d", "between my 12 and 3 o'clock above."),

                ("d", "between my 3 and 6 o'clock above."),

                ("e", "in a nest on the ground."),

                ("f", "right behind you.")]

actions = ["Back Away",

            "Cover our Heads",

            "Take a Photograph"]

codes= {

    "HHH": "Back Away",

    "HHT": "Cover our Heads",

    "HTH": "Take a Photograph",

    "HTT": "",

    "THH": "",

    "THT": "",

    "TTH": "",

    "TTT": ""

}

print(rarebirds["Giant Eagle"]["Aggressive"])

for i in rarebirds.keys():

    print(i)

    if rarebirds[i]["Aggressive"] == True:

        print(actions[1])

    if rarebirds[i]["Endangered"] == True:

        print(actions[0])

for c, v in codes.items():

    print(c, v)

for i in rarebirds.keys():

    rarebirds[i]["seen"] = False

print(rarebirds)

encounter = False

while encounter != True:

    sighting = (input("What do  you see? ").lower())

    print(sighting)

    rarebirdsList = list(rarebirds.keys())

    print(rarebirdsList)

    for i in rarebirdsList:

        if i.lower() != sighting:

            print("that's not one of the birds we're looking for.")

        else:

            encounter = True

            print("this is one of the birds we're looking for.")

code = (input("Where do you see it? "))

print(code)

location = dict(birdlocation)[code]

print(location)

print("So you've seen a" , sighting, location)

for i in rarebirds.keys():

    if sighting == i.lower():

        if rarebirds[i]["Aggressive"] == True:

            print("It's aggressive, we need to " + codes["HHH"] + " and " + codes["HHT"])

            print("We need to  photograph the " + sighting + " at it's " + location)

        elif rarebirds[i]["Endangered"] == True:

            print("It's endangered, we need to " + codes["HHH"])

            print("We need to photograph the " + sighting + " at it's " + location)

        else:

            print("The " + sighting + "is neither aggressive nor endangered.")

            print("We need to photograph the ultra rare " + sighting + " and " + location

workspace $ python3 main.py

True

Gold-crested Toucan

Cover our Heads

Back Away

Pearlescent Kingfisher

Four-metre Hummingbird

Back Away

Giant Eagle

Cover our Heads

Back Away

Ancient Vulture

HHH Back Away

HHT Cover our Heads

HTH Take a Photograph

HTT

THH

THT

TTH

TTT

{'Gold-crested Toucan': {'Height (m)': 1.1, 'Weight (kg)': 35, 'Color': 'Gold', 'Endangered': True, 'Aggressive': True, 'seen': False}, 'Pearlescent Kingfisher': {'Height (m)': 0.25, 'Weight (kg)': 35, 'Color': 'white', 'Endangered': False, 'Aggressive': False, 'seen': False}, 'Four-metre Hummingbird': {'Height (m)': 0.6, 'Weight (kg)': 0.5, 'Color': 'Blue', 'Endangered': True, 'Aggressive': False, 'seen': False}, 'Giant Eagle': {'Height (m)': 1.5, 'Weight (kg)': 52, 'Color': 'Black with White', 'Endangered': True, 'Aggressive': True, 'seen': False}, 'Ancient Vulture': {'Height (m)': 2.1, 'Weight (kg)': 70, 'Color': 'Brown', 'Endangered': False, 'Aggressive': False, 'seen': False}}

What do you see? bluebird

bluebird

['Gold-crested Toucan', 'Pearlescent Kingfisher', 'Four-metre Hummingbird', 'Giant Eagle', 'Ancient Vulture']

that's not one of the birds we're looking for.

that's not one of the birds we're looking for.

that's not one of the birds we're looking for.

that's not one of the birds we're looking for.

that's not one of the birds we're looking for.

What do you see? egret

egret

['Gold-crested Toucan', 'Pearlescent Kingfisher', 'Four-metre Hummingbird', 'Giant Eagle', 'Ancient Vulture']

that's not one of the birds we're looking for.

that's not one of the birds we're looking for.

that's not one of the birds we're looking for.

that's not one of the birds we're looking for.

that's not one of the birds we're looking for.

What do you see? giant eagle

giant eagle

['Gold-crested Toucan', 'Pearlescent Kingfisher', 'Four-metre Hummingbird', 'Giant Eagle', 'Ancient Vulture']

that's not one of the birds we're looking for.

that's not one of the birds we're looking for.

that's not one of the birds we're looking for.

this is one of the birds we're looking for.

that's not one of the birds we're looking for.

Where do you see it? d

d

between my 3 and 6 o'clock above.

So you've seen a giant eagle between my 3 and 6 o'clock above.

It's aggressive, we need to Back Away and Cover our Heads

We need to photograph the giant eagle at it's between my 3 and 6 o'clock above.