

Basic Concepts



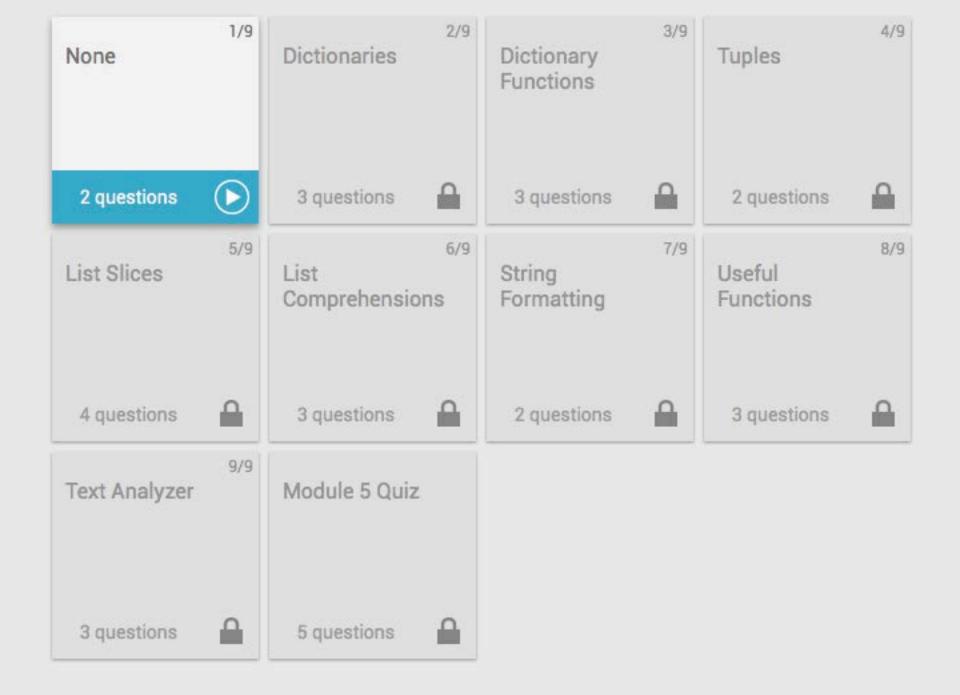




Exceptions & Files







None

The None object is used to represent the absence of a value.

It is similar to null in other programming languages.

Like other "empty" values, such as 0, [] and the empty string, it is False when converted to a Boolean variable.

When entered at the Python console, it is displayed as the empty string.

```
>>> None == None
True
>>> None
>>> print(None)
None
>>>
```

Try It Yourself





What is "None" often used to represent?

O Absence of a value
O An invalid value
O A false value







None

The None object is returned by any function that doesn't explicitly return anything else.

```
def some_func():
    print("Hi!")

var = some_func()
    print(var)
```

Try It Yourself

Result:

```
>>>
Hi!
None
>>>
```





```
What number does this code print?

foo = print()

if foo == None:
    print(1)

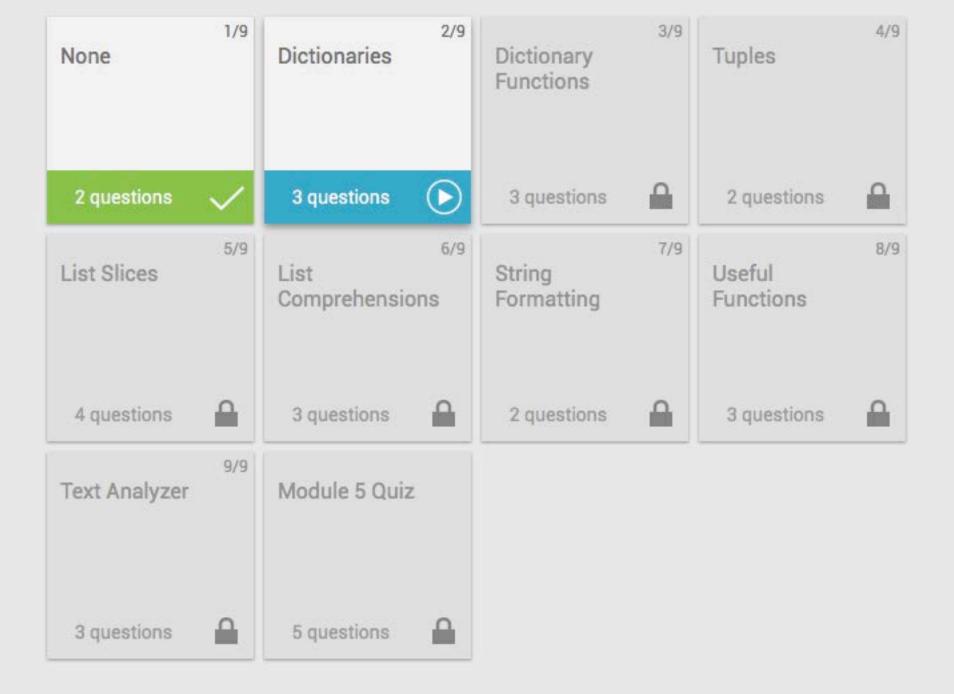
else:
    print(2)
```











Dictionaries are data structures used to map arbitrary keys to values.

Lists can be thought of as dictionaries with integer keys within a certain range.

Dictionaries can be indexed in the same way as lists, using square brackets containing keys.

Example:

```
ages = {"Dave": 24, "Mary": 42, "John": 58}
print(ages["Dave"])
print(ages["Mary"])
```

Try It Yourself

Result:

```
>>>
24
42
>>>
```

Each element in a dictionary is represented by a key.value pair.





Fill in the blanks to define a valid dictionary with two elements.









Trying to index a key that isn't part of the dictionary returns a **KeyError**. **Example:**

```
primary = {
    "red": [255, 0, 0],
    "green": [0, 255, 0],
    "blue": [0, 0, 255],
}

print(primary["red"])
print(primary["yellow"])
```

Try It Yourself

Result:

```
>>>
[255, 0, 0]

KeyError: 'yellow'
>>>
```

As you can see, a dictionary can store any types of data as values.

An empty dictionary is defined as {}.

What is the result of this code? test = {} print(test[0])

0	0
0	KeyError
0	None







Only immutable objects can be used as keys to dictionaries. Immutable objects are those that can't be changed. So far, the only mutable objects you've come across are lists and dictionaries. Trying to use a mutable object as a dictionary key causes a TypeError.

```
bad_dict = {
    [1, 2, 3]: "one two three",
}
```

Try It Yourself

Result:

```
>>>
TypeError: unhashable type: 'list'
>>>
```





Which of these values can't be used as a dictionary key?

O {2: 4, 3: 9, 4: 16,}

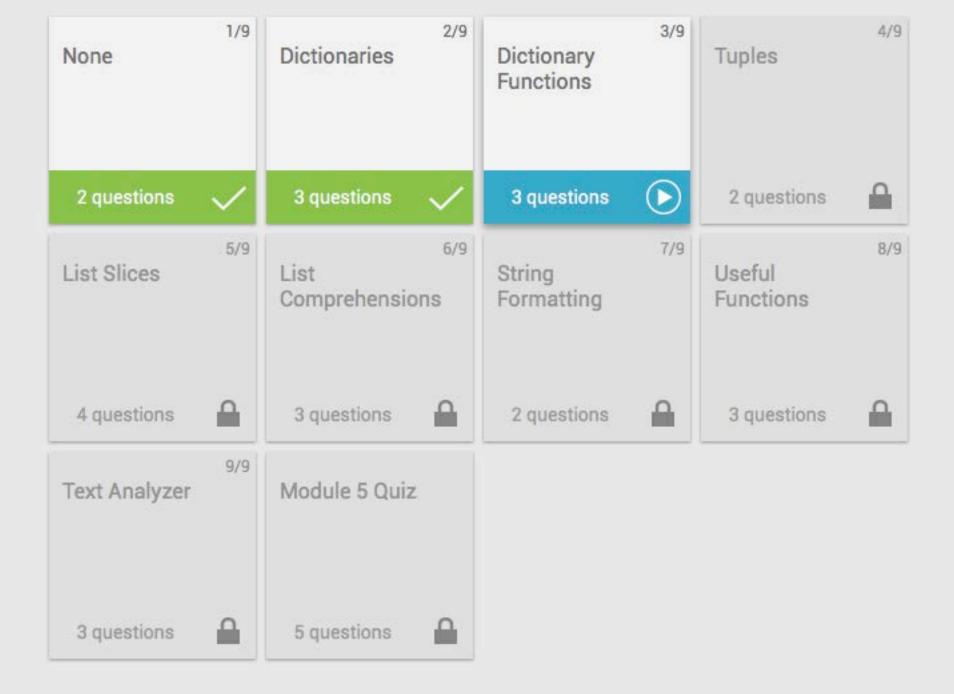
O "one two three"

O False









Just like lists, dictionary keys can be assigned to different values.

However, unlike lists, a new dictionary key can also be assigned a value, not just ones that already exist.

```
squares = {1: 1, 2: 4, 3: "error", 4: 16,}
squares[8] = 64
squares[3] = 9
print(squares)
```

Try It Yourself

Result:

```
{8: 64, 1: 1, 2: 4, 3: 9, 4: 16}
```





What is the result of this code? primes = {1: 2, 2: 3, 4: 7, 7:17} print(primes[primes[4]])









To determine whether a key is in a dictionary, you can use in and not in, just as you can for a list. Example:

```
nums = {
1: "one",
2: "two",
3: "three",
}
print(1 in nums)
print("three" in nums)
print(4 not in nums)
```

Try It Yourself

Result:

```
>>>
True
False
True
>>>
```





Fill in the blanks to print "Yes", if the key 112 is present in the dictionary named "pairs".

```
if _____:
print("Yes")
```

112 in pairs not if print







A useful dictionary method is **get**. It does the same thing as indexing, but if the key is not found in the dictionary it returns another specified value instead ('None', by default).

Example:

```
pairs = {1: "apple",
    "orange": [2, 3, 4],
    True: False,
    None: "True",
}

print(pairs.get("orange"))
print(pairs.get(7))
print(pairs.get(12345, "not in dictionary"))
```

Try It Yourself

Result:

```
>>>
[2, 3, 4]
None
not in dictionary
>>>
```

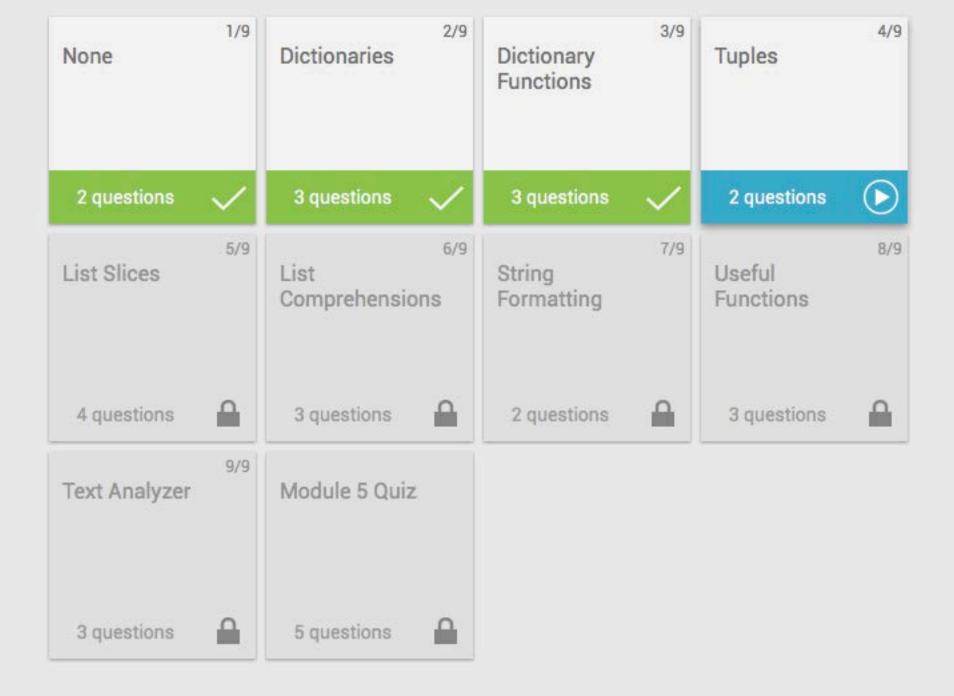
What is the result of this code? fib = {1: 1, 2: 1, 3: 2, 4: 3} print(fib.get(4, 0) + fib.get(7, 5))











Tuples

Tuples are very similar to lists, except that they are immutable (they cannot be changed).

Also, they are created using parentheses, rather than square brackets.

Example:

```
words = ("spam", "eggs", "sausages",)
```

You can access the values in the tuple with their index, just as you did with lists:

```
print(words[0])
```

Try It Yourself

Trying to reassign a value in a tuple causes a TypeError.

```
words[1] = "cheese"
```

Try It Yourself

Result:

>>>
TypeError: 'tuple' object does not support item assignment
>>>

You can access the values in the tuple with their index, just as you did with lists:

print(words[0])

Try It Yourself

Trying to reassign a value in a tuple causes a TypeError.

Try It Yourself

Result:

>>>

TypeError: 'tuple' object does not support item assignment

>>>

Like lists and dictionaries, tuples can be nested within each other.





Fill in the blanks to create a list, dictionary, and tuple:

```
# list
list = _ "one", "two" _
# dictionary
dict = _ 1:"one", 2:"two" _
# tuple
tp = _ "one", "two" _
```









Tuples

Tuples can be created without the parentheses, by just separating the values with commas. Example:

```
my_tuple = "one", "two", "three"
print(my_tuple[0])
```

Try It Yourself

Result:

```
>>> one >>>
```

An empty tuple is created using an empty parenthesis pair.

Tuples are faster than lists, but they cannot be changed.





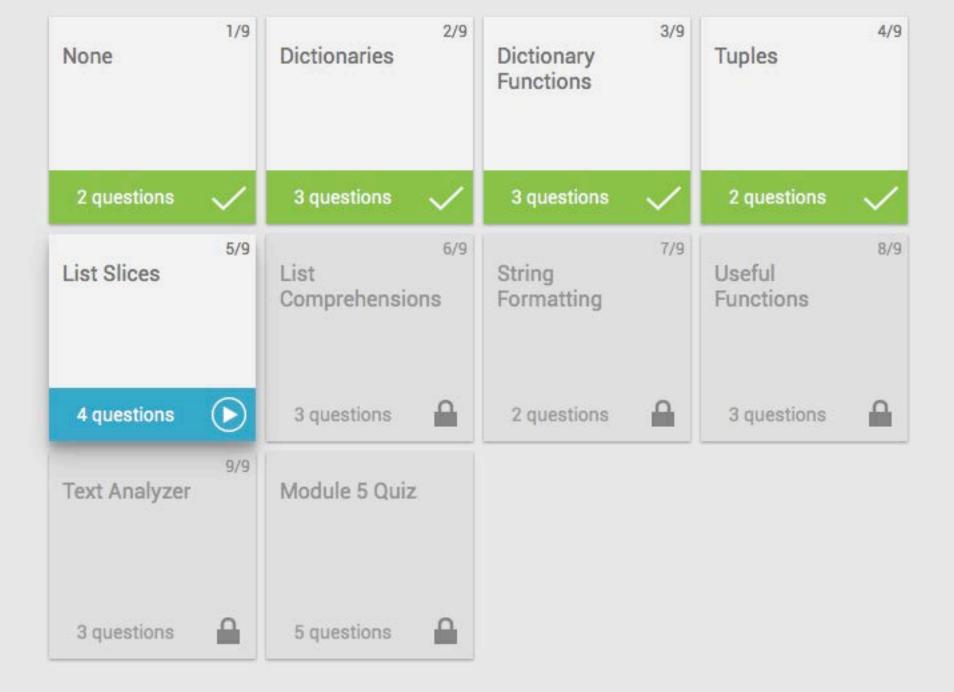
What is the result of this code? tuple = (1, (1, 2, 3)) print(tuple[1])

- O ((1, 2, 3))
- 0 1
- 0 (1, 2, 3)









List Slices

List slices provide a more advanced way of retrieving values from a list. Basic list slicing involves indexing a list with **two colon-separated integers**. This returns a new list containing all the values in the old list between the indices.

Example:

```
squares = [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

print(squares[2:6])

print(squares[3:8])

print(squares[0:1])
```

Try It Yourself

Result:

```
>>>
[4, 9, 16, 25]
[9, 16, 25, 36, 49]
[0]
>>>
```

Like the arguments to range, the first index provided in a slice is included in the result, but the second isn't.

What is the result of this code? sqs = [0, 1, 4, 9, 16, 25, 36, 49, 64] print(sqs[4:7])

- 0 [16, 25, 36]
- O [16, 25, 36, 49]
- O [25, 36, 49]







List Slices

If the first number in a slice is omitted, it is taken to be the start of the list. If the second number is omitted, it is taken to be the end.

Example:

```
squares = [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
print(squares[:7])
print(squares[7:])
```

Try It Yourself

Result:

```
>>>
[0, 1, 4, 9, 16, 25, 36]
[49, 64, 81]
>>>
```

Slicing can also be done on tuples.





Fill in the blanks to take the first two elements of the list:









List Slices

List slices can also have a third number, representing the step, to include only alternate values in the slice.

```
squares = [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
print(squares[::2])
print(squares[2:8:3])
```

Try It Yourself

Result:

```
>>>
[0, 4, 16, 36, 64]
[4, 25]
>>>
```

[2:8:3] will include elements starting from the 2nd index up to the 8th with a step of 3.





What is the output of this code? sqs = [0, 1, 4, 9, 16, 25, 36, 49, 64, 81] print(sqs[1::4])

- O [1, 25, 81]
- 0 [0, 1, 4]
- 0 [1, 25]
- O An error occurs







List Slices

Negative values can be used in list slicing (and normal list indexing). When negative values are used for the first and second values in a slice (or a normal index), they count from the end of the list.

```
squares = [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
print(squares[1:-1])
```

Try It Yourself

Result:

```
>>>
[1, 4, 9, 16, 25, 36, 49, 64]
>>>
```

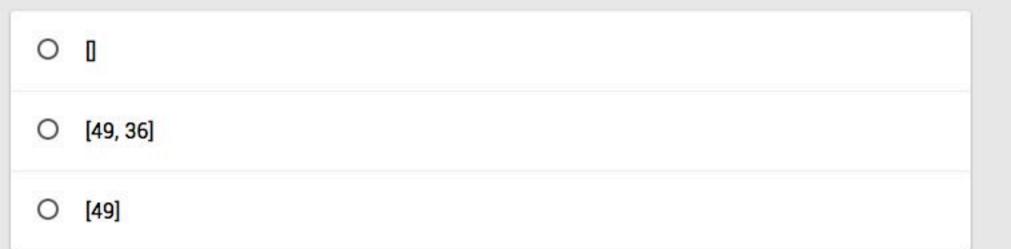
If a negative value is used for the step, the slice is done backwards.

Using [::-1] as a slice is a common and idiomatic way to reverse a list.





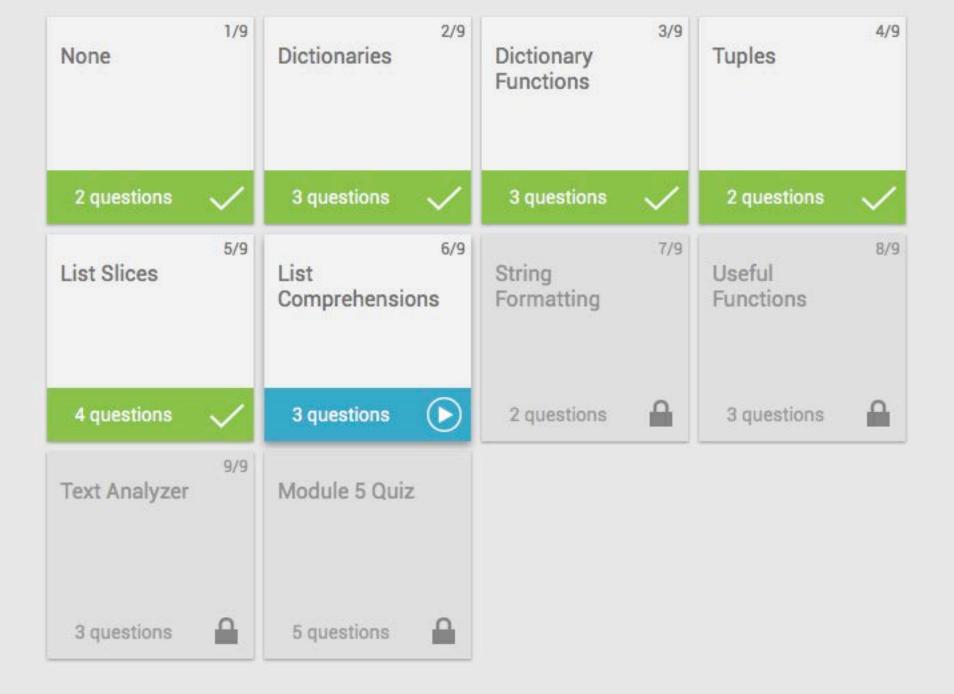
What is the output of this code? sqs = [0, 1, 4, 9, 16, 25, 36, 49, 64, 81] print(sqs[7:5:-1])













Basic Concepts





Exceptions & Files





Functional Programming

List Comprehensions

List comprehensions are a useful way of quickly creating lists whose contents obey a simple rule. For example, we can do the following:

```
# a list comprehension

cubes = [i**3 for i in range(5)]

print(cubes)
```

Try It Yourself

Result:

```
>>>
[0, 1, 8, 27, 64]
>>>
```

List comprehensions are inspired by set-builder notation in mathematics.





)?) ?) ?) ?) ?

What does this list comprehension create? nums = [i*2 for i in range(10)]

- O A list of even numbers between 0 and 10
- A list of even numbers between 0 and 18
- A list of all numbers between 0 and 10







List Comprehensions

A list comprehension can also contain an if statement to enforce a condition on values in the list. Example:

```
evens=[i**2 for i in range(10) if i**2 % 2 == 0]
print(evens)
```

Try It Yourself

Result:

```
>>>
[0, 4, 16, 36, 64]
>>>
```





Create a list of multiples of 3 from 0 to 20.









List Comprehensions

Trying to create a list in a very extensive range will result in a **MemoryError**. This code shows an example where the list comprehension runs out of memory.

```
even = [2*i for i in range(10**100)]
```

Try It Yourself

Result:

>>> MemoryError >>>

This issue is solved by generators, which are covered in the next module.





Fill in the blanks to create a list of numbers multiplied by 10 in the range of 5 to 9.

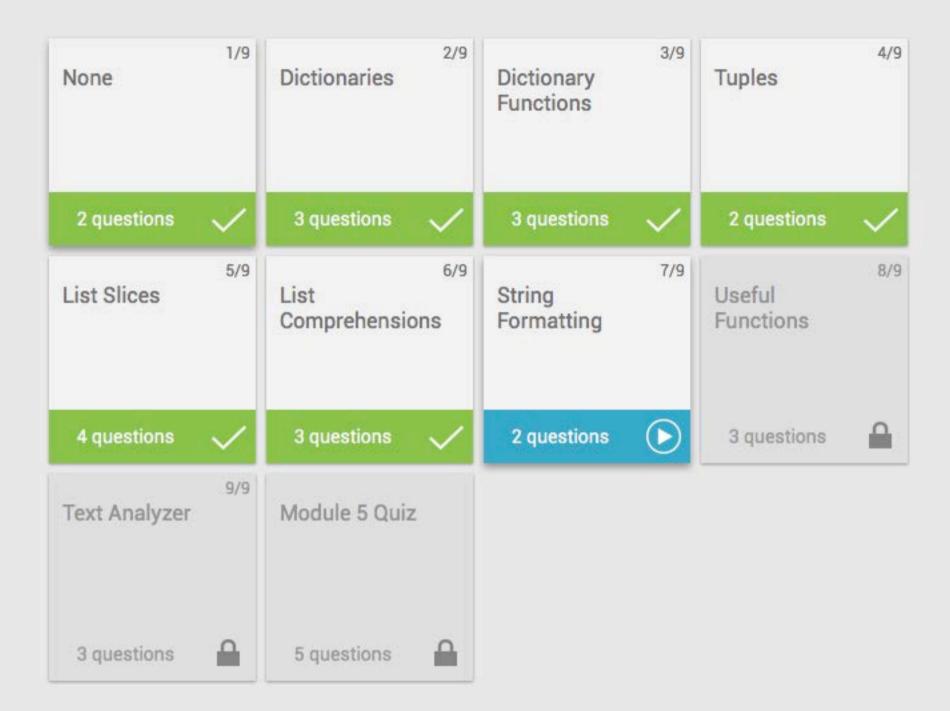
$$a = [x*10 \text{ for } x \text{ in range}(5,9)]$$











String Formatting

So far, to combine strings and non-strings, you've converted the non-strings to strings and added them.

String formatting provides a more powerful way to embed non-strings within strings. String formatting uses a string's format method to substitute a number of arguments in the string. **Example:**

```
# string formatting
nums = [4, 5, 6]
msg = "Numbers: {0} {1} {2}". format(nums[0], nums[1], nums[2])
print(msg)
```

Try It Yourself

Result:

>>>

Numbers: 456

>>>

Each <u>argument</u> of the format <u>function</u> is placed in the <u>string</u> at the corresponding position, which is determined using the curly braces { }.



▶) ?) ▶) ?

What is the result of this code? print("{0}{1}{0}".format("abra", "cad"))

abracadabra









String Formatting

String formatting can also be done with named arguments. **Example:**

```
a = "{x}, {y}".format(x=5, y=12)
print(a)
```

Try It Yourself

Result:

>>> 5, 12 >>>





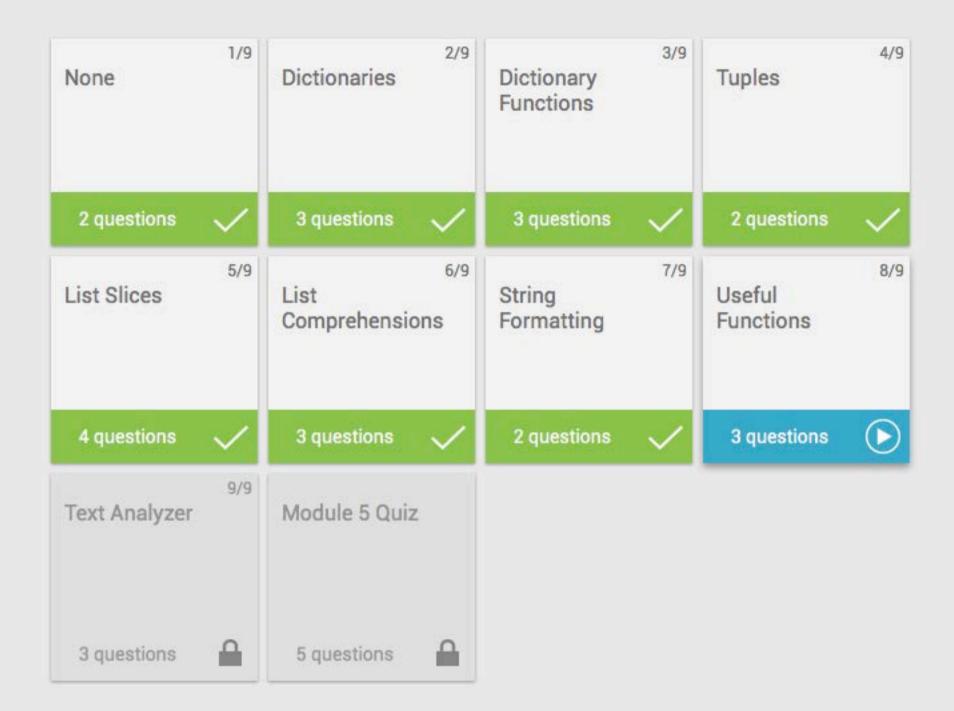
What is the result of this code? str="{c}, {b}, {a}".format(a=5, b=9, c=7) print(str)

- 0 9,7,5
- 0 7,9,5
- 0 5, 9, 7









String Functions

Python contains many useful built-in functions and methods to accomplish common tasks.

join - joins a list of strings with another string as a separator.

replace - replaces one substring in a string with another.

startswith and endswith - determine if there is a substring at the start and end of a string, respectively.

To change the case of a string, you can use lower and upper.

The method split is the opposite of join, turning a string with a certain separator into a list. Some examples:

```
print(", ".join(["spam", "eggs", "ham"]))
#prints "spam, eggs, ham"
print("Hello ME".replace("ME", "world"))
#prints "Hello world"
print("This is a sentence.".startswith("This"))
# prints "True"
print("This is a sentence.".endswith("sentence."))
# prints "True"
print("This is a sentence.".upper())
# prints "THIS IS A SENTENCE."
print("AN ALL CAPS SENTENCE".lower())
#prints "an all caps sentence"
print("spam, eggs, ham".split(", "))
#prints "['spam', 'eggs', 'ham']"
```

respectively.

To change the case of a string, you can use lower and upper.

The method split is the opposite of join, turning a string with a certain separator into a list. Some examples:

```
print(", ".join(["spam", "eggs", "ham"]))
#prints "spam, eggs, ham"
print("Hello ME".replace("ME", "world"))
#prints "Hello world"
print("This is a sentence.".startswith("This"))
# prints "True"
print("This is a sentence.".endswith("sentence."))
# prints "True"
print("This is a sentence.".upper())
# prints "THIS IS A SENTENCE."
print("AN ALL CAPS SENTENCE".lower())
#prints "an all caps sentence"
print("spam, eggs, ham".split(", "))
#prints "['spam', 'eggs', 'ham']"
```

Try It Yourself





Fill in the blanks to turn the string uppercase.









Numeric Functions

To find the maximum or minimum of some numbers or a list, you can use max or min.

To find the distance of a number from zero (its absolute value), use abs.

To round a number to a certain number of decimal places, use round.

To find the total of a list, use sum.

Some examples:

```
print(min(1, 2, 3, 4, 0, 2, 1))
print(max([1, 4, 9, 2, 5, 6, 8]))
print(abs(-99))
print(abs(42))
print(sum([1, 2, 3, 4, 5]))
```

Try It Yourself

Result:

```
>>>
0
9
99
42
15
>>>
```

What is the result of this code? a=min([sum([11, 22]), max(abs(-30), 2)]) print(a)









List Functions

Often used in conditional statements, **all** and **any** take a list as an **argument**, and return **True** if all or any (respectively) of their arguments evaluate to **True** (and **False** otherwise). The **function enumerate** can be used to iterate through the values and indices of a list simultaneously.

Example:

```
nums = [55, 44, 33, 22, 11]

if all([i > 5 for i in nums]):
   print("All larger than 5")

if any([i % 2 == 0 for i in nums]):
   print("At least one is even")

for v in enumerate(nums):
   print(v)
```

Try It Yourself

Result:

```
>>>
All larger than 5
At least one is even
(0, 55)
(1, 44)
(2, 33)
(3, 22)
```

```
nums = [55, 44, 33, 22, 11]

if all([i > 5 for i in nums]):
   print("All larger than 5")

if any([i % 2 == 0 for i in nums]):
   print("At least one is even")

for v in enumerate(nums):
   print(v)
```

Try It Yourself

Result:

```
>>>
All larger than 5
At least one is even
(0, 55)
(1, 44)
(2, 33)
(3, 22)
(4, 11)
>>>
```





```
What is the result of this code?

nums = [-1, 2, -3, 4, -5]

if all([abs(i) < 3 for i in nums]):

print(1)

else:

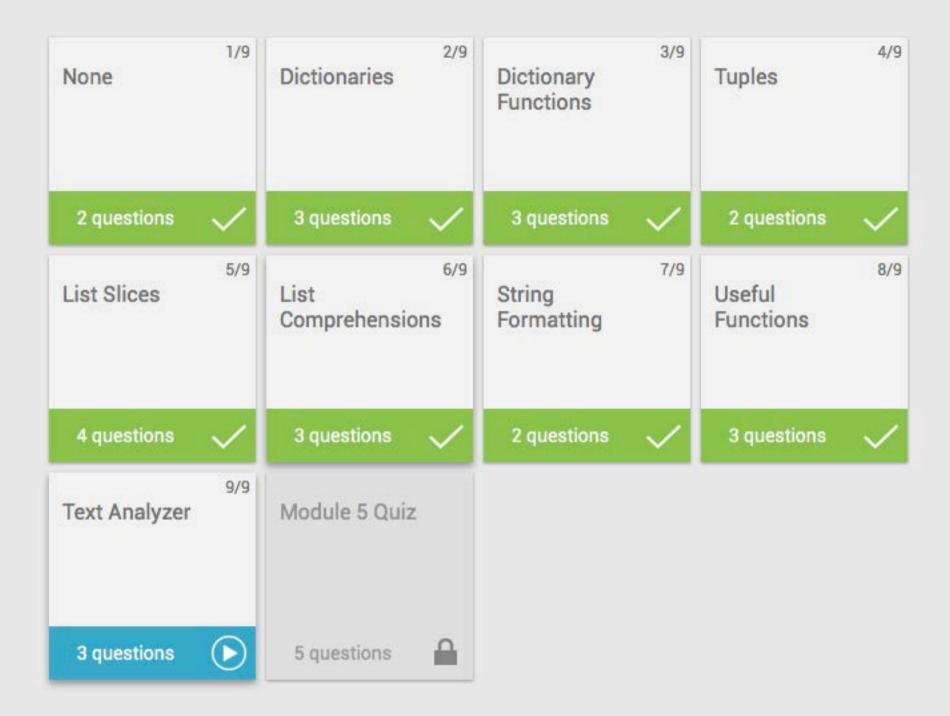
print(2)
```

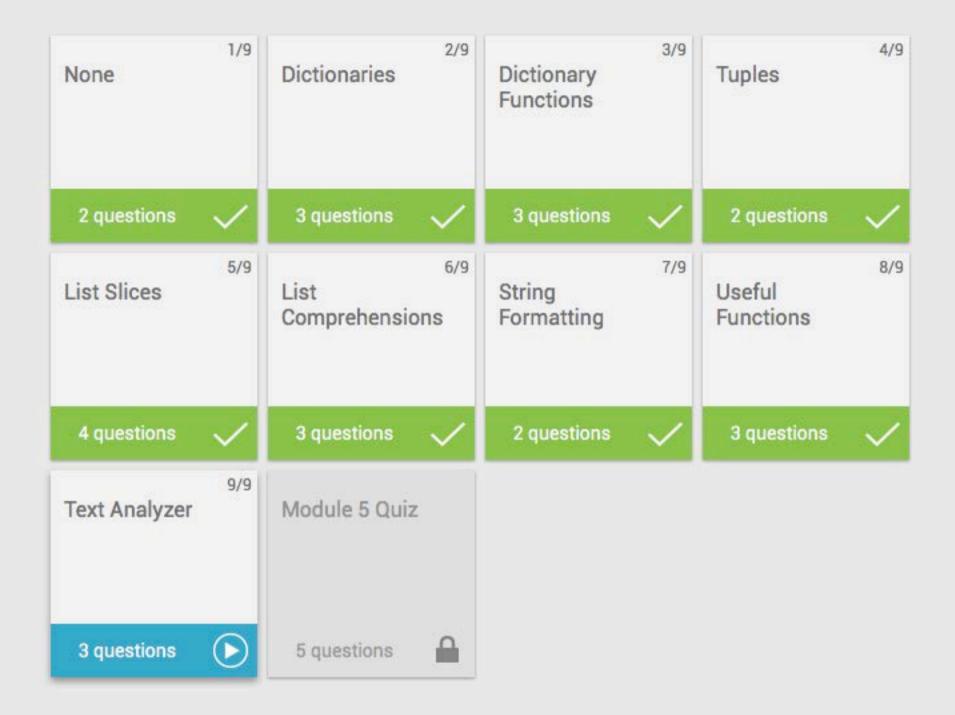












Text Analyzer

This is an example project, showing a program that analyzes a sample file to find what percentage of the text each character occupies.

This section shows how a file could be open and read.

```
filename = input("Enter a filename: ")

with open(filename) as f:
   text = f.read()

print(text)
```

Result:

>>>

Enter a filename: test.txt

Ornhgvshy vf orggre guna htyl.

Rkcyvpvg vf orggre guna vzcyvpvg.

Fvzcyr vf orggre guna pbzcyvpngrq.

Syng vf orggre guna arfgrq.

Fcenfr fv orggre guna qrafr.

Ernqnovyvgl pbhagf.

Fcrpvny pnfrf nera'g fcrpvny rabthu gb oernx gur ehyrf.

Nygubhtu cenpgvpnyvgl orgnf chevgl.

Reebef fubhyq arire cnff fvyragyl.

Hayrff rkcyvpvgyl fvyraprq.

Va gur snpr bs nzovthvgl, ershfr gur grzcgngvba bg thrff.

Gurer fubhyq or bar-- naq cersrenoylbayl bar --boivbhf jnl gb qb vg.

Nygubhtu gung jnl znl abg or boivbhf ng svefg hayrff lbh'er Qhgpu.

Abj vf orggre guna arrire.

Nygubhtu arire vf bsgra orggre guna *evtug* abj.

Vs gur vzcyrzragngyba vf uneg gb rkcynya, vg'f n ong vgrn.

text = f.read()
print(text)

Result:

>>>

Enter a filename: test.txt

Ornhgvshy vf orggre guna htyl.

Rkcyvpvg vf orggre guna vzcyvpvg.

Fvzcyr vf orggre guna pbzcyvpngrq.

Syng vf orggre guna arfgrq.

Fcenfr fv orggre guna qrafr.

Ernqnovyvgl pbhagf.

Fcrpvny pnfrf nera'g fcrpvny rabthu gb oernx gur ehyrf.

Nygubhtu cenpgvpnyvgl orgnf chevgl.

Reebef fubhyq arire cnff fvyragyl.

Hayrff rkcyvpvgyl fvyraprq.

Va gur snpr bs nzovthvgl, ershfr gur grzcgngvba bg thrff.

Gurer fubhyq or bar-- naq cersrenoylbayl bar --boivbhf jnl gb qb vg.

Nygubhtu gung jnl znl abg or boivbhf ng svefg hayrff lbh'er Qhgpu.

Abj vf orggre guna arrire.

Nygubhtu arire vf bsgra orggre guna *evtug* abj.

Vs gur vzcyrzragngvba vf uneq gb rkcynva, vg'f n onq vqrn.

Vs gur vzcyrzragngvba vf rnfl gb rkcynva, vg znl or n tbbq vqrn.

Anzrfcnprf ner bar ubaxvat terng vgrn -- yrg'f qb zber bs gubfr!

This is sample content for demonstration purposes only.





Fill in the blanks to read the contents of a file using the "with" statement.









Text Analyzer

This part of the program shows a function that counts how many times a character occurs in a string.

```
def count_char(text, char):
    count = 0
    for c in text:
    if c == char:
        count += 1
    return count
```

This function takes as its arguments the text of the file and one character, returning the number of times that character appears in the text.

Now we can call it for our file.

```
filename = input("Enter a filename: ")
with open(filename) as f:
text = f.read()

print(count_char(text, "r"))
```

Result:

```
>>>
Enter a filename: test.txt
83
>>>
```

```
for c in text:
if c == char:
count += 1
return count
```

This function takes as its arguments the text of the file and one character, returning the number of times that character appears in the text.

Now we can call it for our file.

```
filename = input("Enter a filename: ")
with open(filename) as f:
text = f.read()

print(count_char(text, "r"))
```

Result:

```
>>>
Enter a filename: test.txt
83
>>>
```

The character "r" appears 83 times in the file.





▶ ? ? > ▶ ? ? > ▶ ?

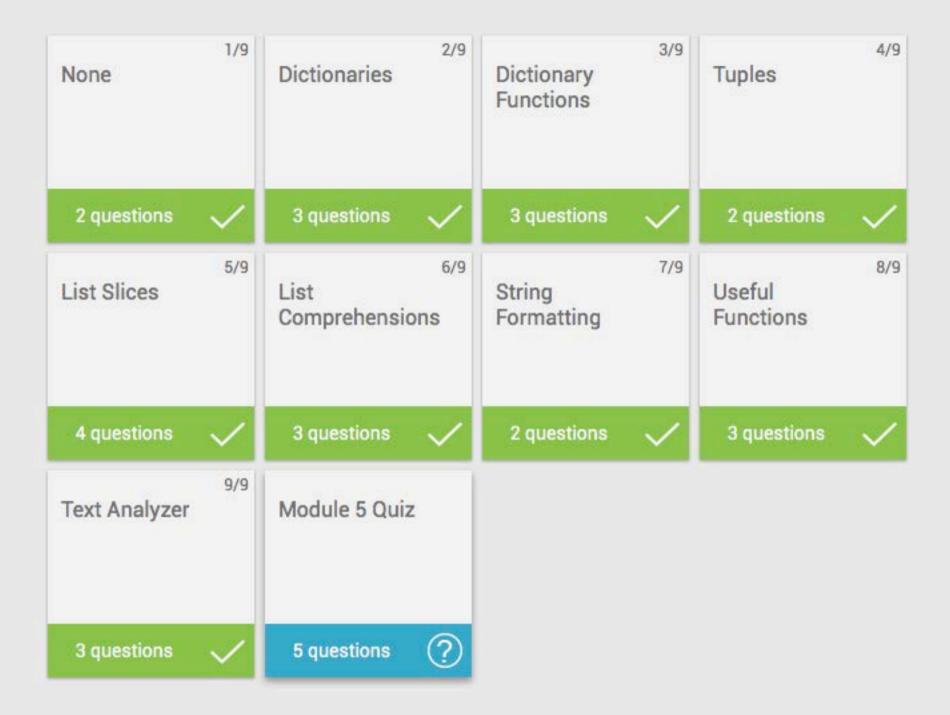
What is the purpose of the round function in the code?

- O To make it more precise
 - O To reduce the number of digits printed
 - O To save memory









? > ? > ? > ? > ?

Can you slice a tuple?

O No

O Yes







? ? ? ? ? ? ? ?

Which list slicing reverses the list 'numbers'?

0	numbers[::-1]
0	numbers[::]
0	numbers[-1::]







? > ? > ? > ? > ?

What could be described as an immutable list?

0	A t	lau	e
_	,,,,,	up.	•

- O A number
- A dictionary







? > ? > ? > ? > ?

What is returned by functions that don't have a return statement?

0	0		
0	False		
0	None		







? ? ? ? ?

What is the result of this code? nums = (55, 44, 33, 22) print(max(min(nums[:2]), abs(-42)))











Basic Concepts









