Data Types & Variables

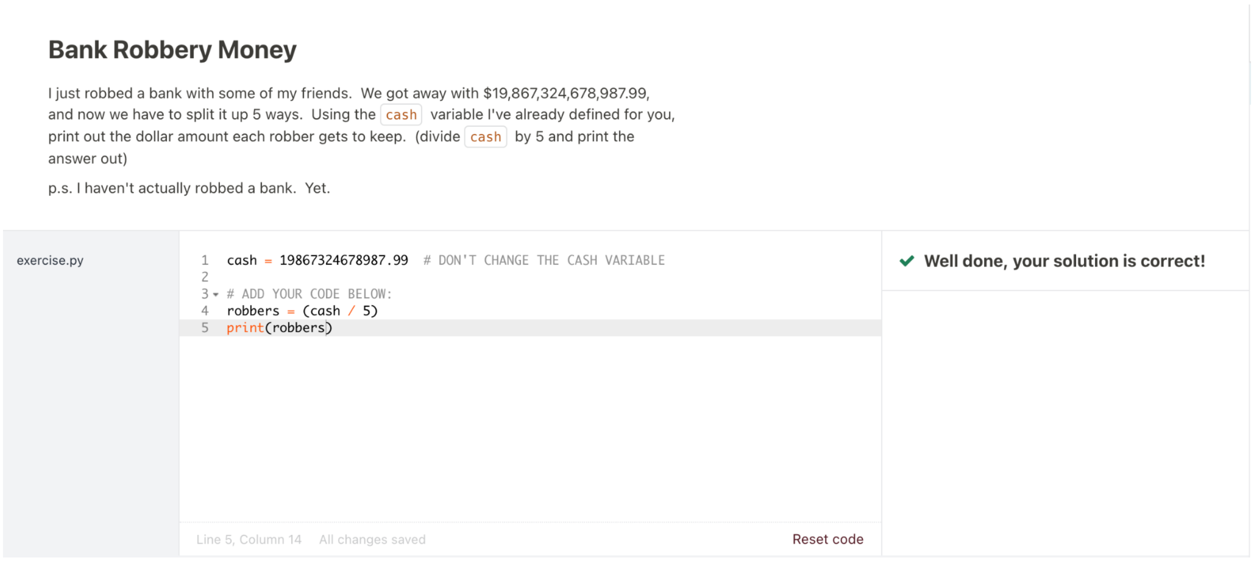
Objectives:

- Understand how to assign and use variables
- Learn Python naming restrictions and conventions
- Learn and use some of the different data types in Python
- Learn why Python is a dynamically typed language
- Understand how to convert data types
- Learn the ins and outs of strings
- Build a program that gets user input

Variable Assignment:

A variable in Python is similar to a variable in mathematics—it's a named symbol that holds a value. They are always assigned with the variable name on the left and the value on the right of the equal sign.

Variables must be assigned before they can be used.



Bank Robbery Money

I just robbed a bank with some of my friends. We got away with $19,867,324,678,987.99, and now we have to split it up 5 ways. Using the `cash` variable I've already defined for you, print out the dollar amount each robber gets to keep. (divide `cash` by 5 and print the answer out)

p.s. I haven't actually robbed a bank. Yet.

```python
cash = 19867324678987.99  # DON'T CHANGE THE CASH VARIABLE

# ADD YOUR CODE BELOW:
robbers = (cash / 5)
print(robbers)
```

✓ Well done, your solution is correct!

Naming restrictions:

- Variables must start with a letter or underscore
- The rest of the variable must consist of letters, numbers, or underscores
- Names are case-sensitive

Naming conventions:

Most Python programmers prefer to use standard style conventions when naming things.

- Most variables should be snake case (underscores between words)
- Most variables should be lower case, with some exceptions:
    - CAPITAL_SNAKE_CASE refers to constants (i.e. PI = 3.14)
    - UpperCamelCase usually refers to a class

- Variables that start and end with two underscores, aka dunder (**d**ouble **under**score) are supposed to be private or left alone

Data Types:
In any assignment, the assigned value must always be a valid data type. Those data types include

1. bool — or Boolean, True or False values
2. int — integer (1, 2, 3)
3. str — string, or a sequence of unicode characters
4. list — an ordered sequence of values of other data types
5. dict — or dictionary which contains a collection of keys: values

What is Dynamic Typing? Python is highly flexible about reassigning variables to different types.

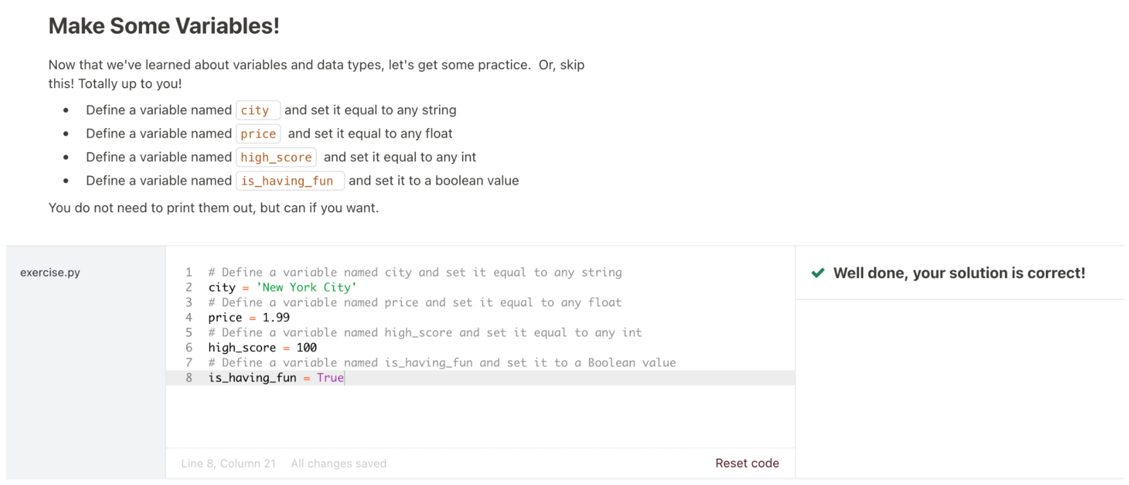
```
awesome = True
awesome = “a dog”
awesome = None
awesome = 22 / 7
```

Since the variables above can change types readily, this is dynamic typing.
Languages such as C++ are statically-typed because the variables are stuck with their originally assigned type.

More to read in this link here:
https://hackernoon.com/i-finally-understand-static-vs-dynamic-typing-and-you-will-too-ad0c2bd0acc7

The special value of None:
The keyword **None** is used to define a null value, or no value at all.

Declaring Strings:
String literals in Python can be declared with either single or double quotes. Either one is perfectly fine however, it is important to stick to the same convention throughout the same file.



Make Some Variables!

Now that we've learned about variables and data types, let's get some practice. Or, skip this! Totally up to you!

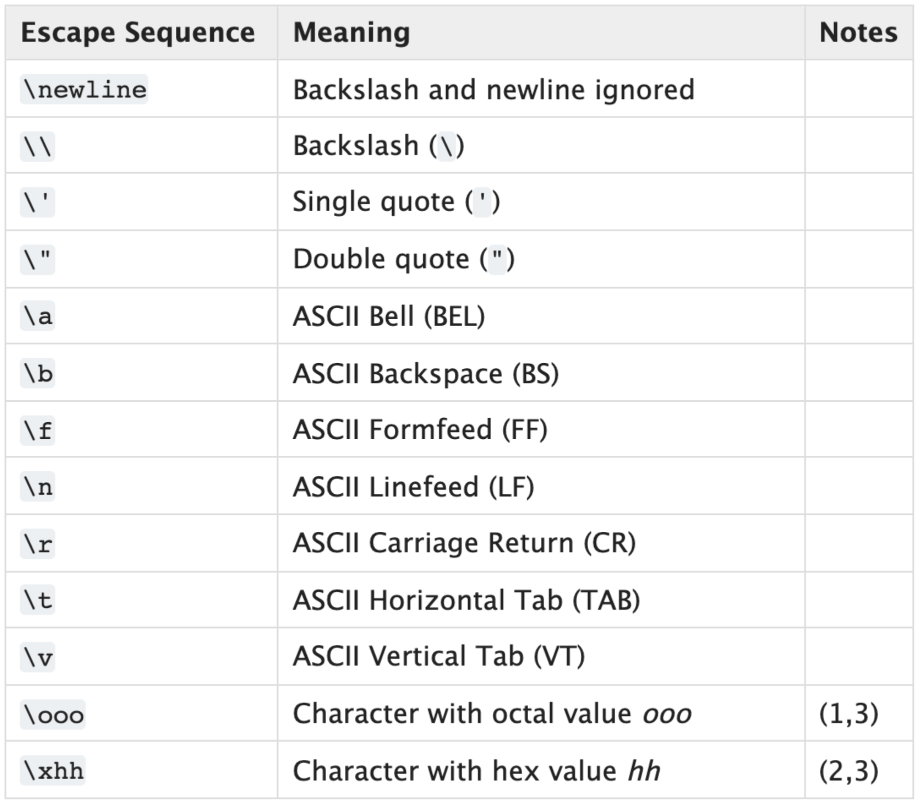
- Define a variable named `city` and set it equal to any string
- Define a variable named `price` and set it equal to any float
- Define a variable named `high_score` and set it equal to any int
- Define a variable named `is_having_fun` and set it to a boolean value

You do not need to print them out, but can if you want.

```
# Define a variable named city and set it equal to any string
city = 'New York City'
# Define a variable named price and set it equal to any float
price = 1.99
# Define a variable named high_score and set it equal to any int
high_score = 100
# Define a variable named is_having_fun and set it to a Boolean value
is_having_fun = True
```

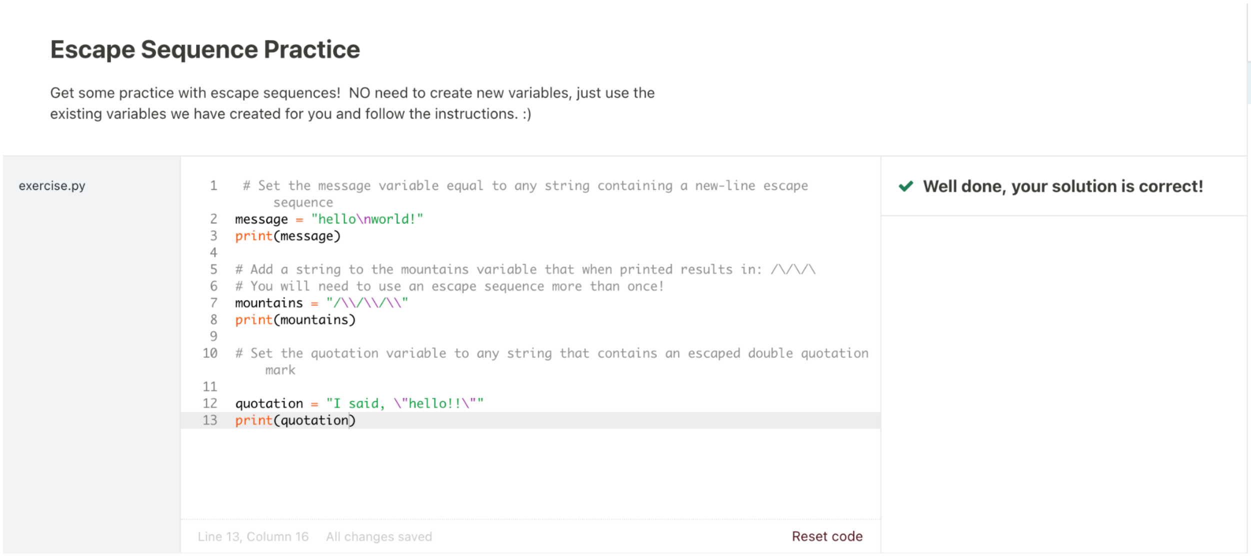
✓ Well done, your solution is correct!

In Python, there are also ‘escape characters’, which are ‘metacharacters’ that get interpreted by Python to do something special. Here are some examples:



Escape Sequence	Meaning	Notes
`\newline`	Backslash and newline ignored	
`\\`	Backslash (`\`)	
`\'`	Single quote (`'`)	
`\"`	Double quote (`"`)	
`\a`	ASCII Bell (BEL)	
`\b`	ASCII Backspace (BS)	
`\f`	ASCII Formfeed (FF)	
`\n`	ASCII Linefeed (LF)	
`\r`	ASCII Carriage Return (CR)	
`\t`	ASCII Horizontal Tab (TAB)	
`\v`	ASCII Vertical Tab (VT)	
`\ooo`	Character with octal value *ooo*	(1,3)
`\xhh`	Character with hex value *hh*	(2,3)

Here is the link to read more about literals:
https://docs.python.org/3/reference/lexical_analysis.html



Escape Sequence Practice

Get some practice with escape sequences! NO need to create new variables, just use the existing variables we have created for you and follow the instructions. :)

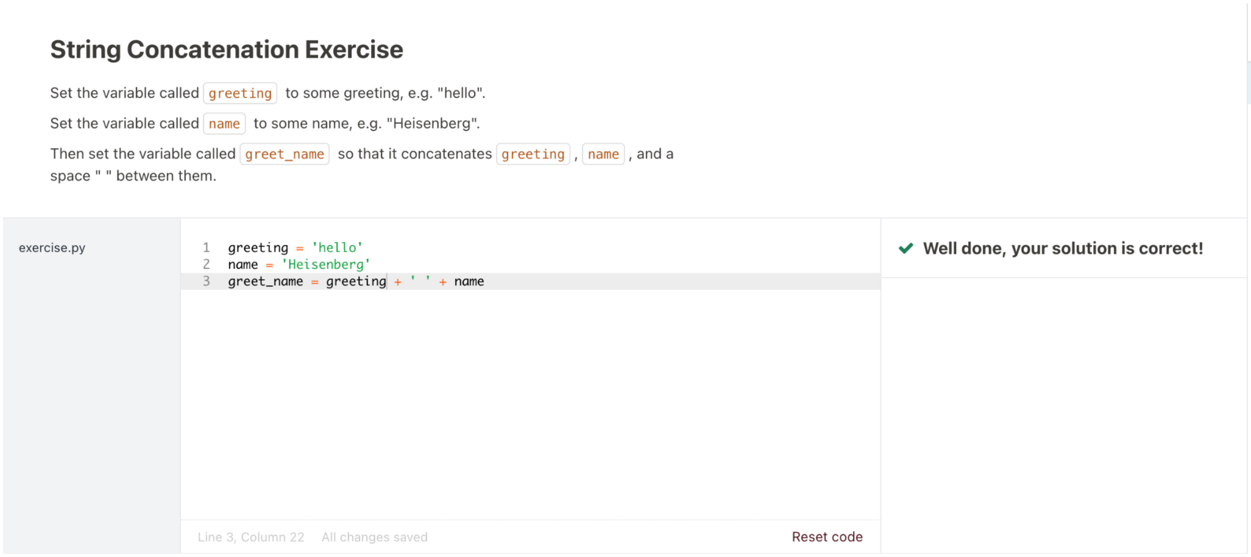
exercise.py

```python
 # Set the message variable equal to any string containing a new-line escape
     sequence
message = "hello\nworld!"
print(message)

# Add a string to the mountains variable that when printed results in: /\/\/\
# You will need to use an escape sequence more than once!
mountains = "/\\/\\/\\"
print(mountains)

# Set the quotation variable to any string that contains an escaped double quotation
    mark

quotation = "I said, \"hello!!\""
print(quotation)
```

Line 13, Column 16   All changes saved   Reset code

✓ Well done, your solution is correct!

String Concatenation — is combining multiple strings together. In Python, you can simply do this with the '+' operator.



String Concatenation Exercise

Set the variable called `greeting` to some greeting, e.g. "hello".

Set the variable called `name` to some name, e.g. "Heisenberg".

Then set the variable called `greet_name` so that it concatenates `greeting`, `name`, and a space " " between them.

exercise.py

```python
greeting = 'hello'
name = 'Heisenberg'
greet_name = greeting + ' ' + name
```

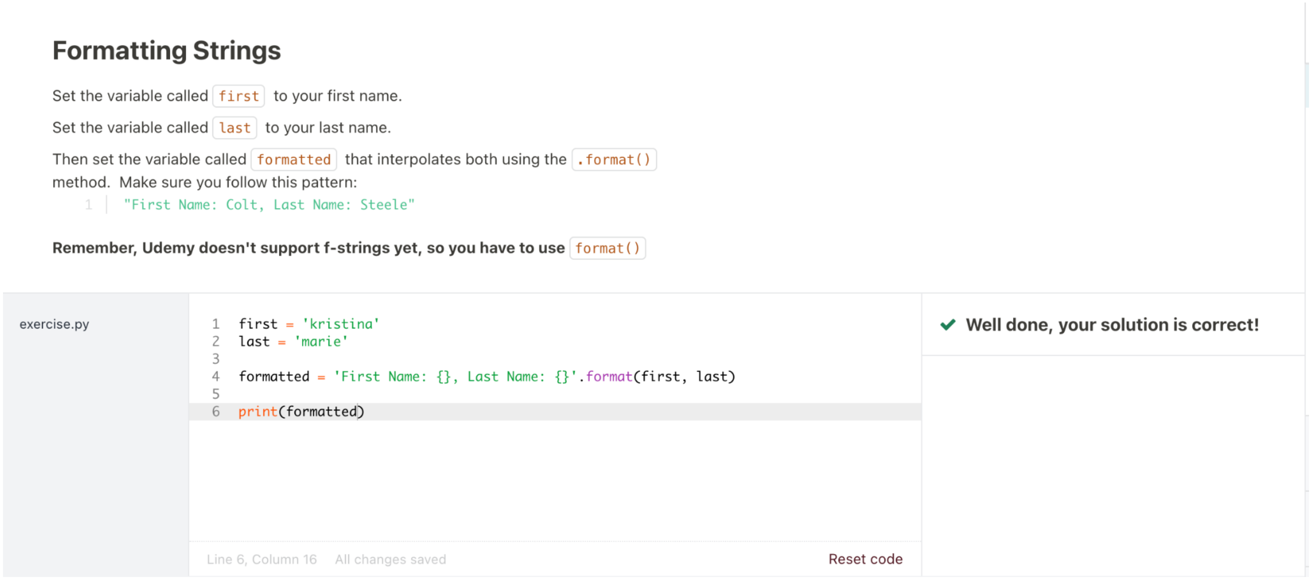
Line 3, Column 22   All changes saved   Reset code

✓ Well done, your solution is correct!

Formatting Strings:

There are also several ways to format strings in Python to **interpolate** variables.
The new way in Python3.6+ is by using f-strings.

i.e.
```
x = 10
formatted = f"I\'ve told you {x} times already!"
```



Formatting Strings

Set the variable called `first` to your first name.

Set the variable called `last` to your last name.

Then set the variable called `formatted` that interpolates both using the `.format()` method. Make sure you follow this pattern:

```
"First Name: Colt, Last Name: Steele"
```

Remember, Udemy doesn't support f-strings yet, so you have to use `format()`

exercise.py

```
first = 'kristina'
last = 'marie'

formatted = 'First Name: {}, Last Name: {}'.format(first, last)

print(formatted)
```

Line 6, Column 16   All changes saved   Reset code

✓ Well done, your solution is correct!

Converting Data Types:

In string interpolation, data types are implicitly converted into string form. You can also explicitly convert variables by using the name of the builtin type as a function.

Boolean and Conditional Logic

Objectives:

- Learn how to get user input in Python
- Learn about "Truthiness"
- Learn how to use comparison operators to make a basic program
    - How do we make decisions in our programs using comparison operators and boolean logic?

User Input:

There is a built-in function within Python called 'input' that will prompt the user and store the result to a variable.

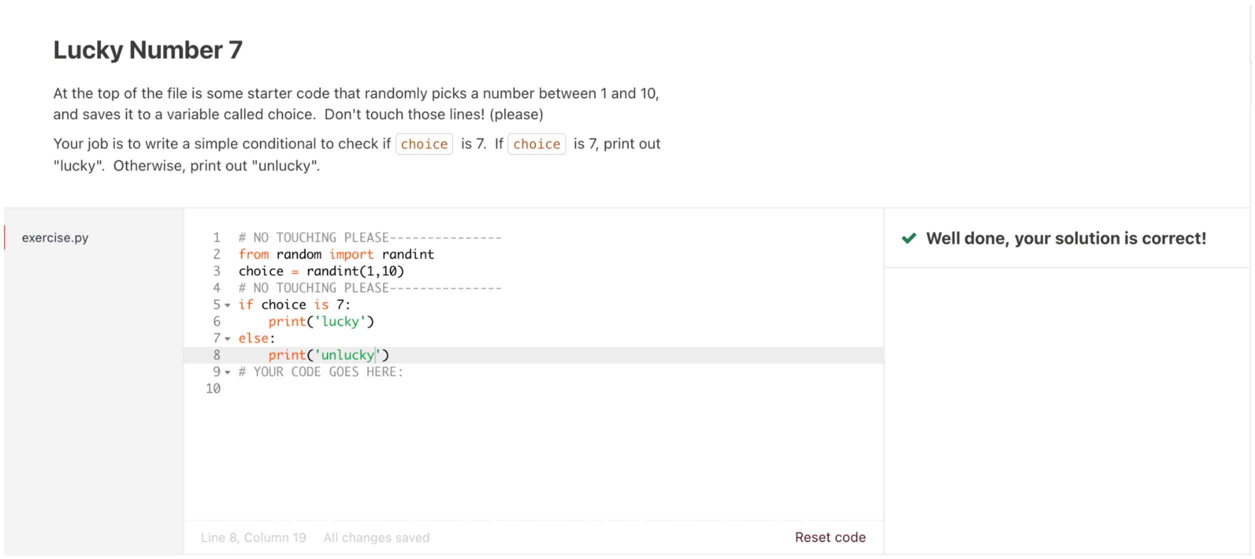
i.e. name = input("Enter your name here: ")   # prompts the user to input name

Boolean Expressions:

Conditional Statements — are conditional logic using *if* statements to represent different paths a program can take based on some type of comparison input.

This pseudocode was written to represent a conditional statement:

```
if CONDITION_1 is True:
    do something
elif CONDITION_2 is True:
    do something
else:
    do something
```



Lucky Number 7

At the top of the file is some starter code that randomly picks a number between 1 and 10, and saves it to a variable called choice. Don't touch those lines! (please)

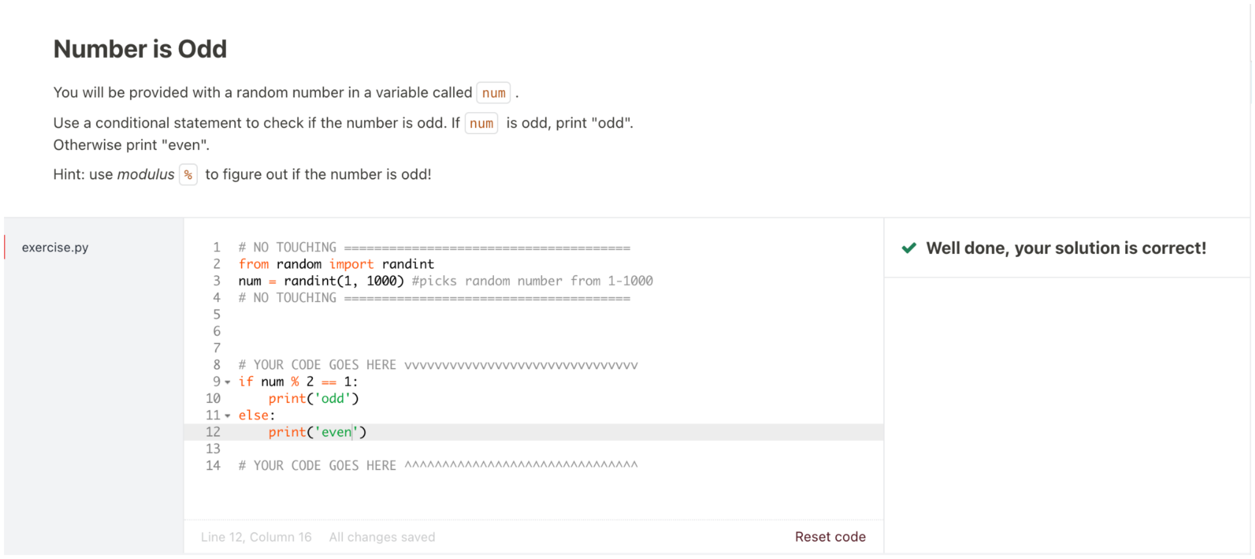
Your job is to write a simple conditional to check if `choice` is 7. If `choice` is 7, print out "lucky". Otherwise, print out "unlucky".

exercise.py

```python
# NO TOUCHING PLEASE---------------
from random import randint
choice = randint(1,10)
# NO TOUCHING PLEASE---------------
if choice is 7:
    print('lucky')
else:
    print('unlucky')
# YOUR CODE GOES HERE:

```

Line 8, Column 19   All changes saved   Reset code

✓ Well done, your solution is correct!



Number is Odd

You will be provided with a random number in a variable called `num`.

Use a conditional statement to check if the number is odd. If `num` is odd, print "odd". Otherwise print "even".

Hint: use *modulus* `%` to figure out if the number is odd!

exercise.py

```python
# NO TOUCHING ======================================
from random import randint
num = randint(1, 1000) #picks random number from 1-1000
# NO TOUCHING ======================================



# YOUR CODE GOES HERE vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
if num % 2 == 1:
    print('odd')
else:
    print('even')

# YOUR CODE GOES HERE ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

Line 12, Column 16   All changes saved   Reset code

✓ Well done, your solution is correct!

Truthiness:
In Python, all conditional checks resolve to **True** or **False**.

We can call values that resolve to True “truthy”, or values that resolve to False “falsey”.

Besides False conditional checks, other things that are naturally falsey include: empty objects, empty strings, *None*, and zero.

Operation	What it does	Example
==	Truthy if **a** has the exact same value as **b**	a **==** b
!=	Truthy if **a** does **NOT** have the same value as **b**	a **!=** b
> <	Truthy if **a** is greater than **b** Truthy if **a** is less than **b**	a **>** b a **<** b
>= <=	Truthy if **a** is greater than or equal to **b** Truthy if **a** is less than or equal to **b**	a **>=** b a **<=** b

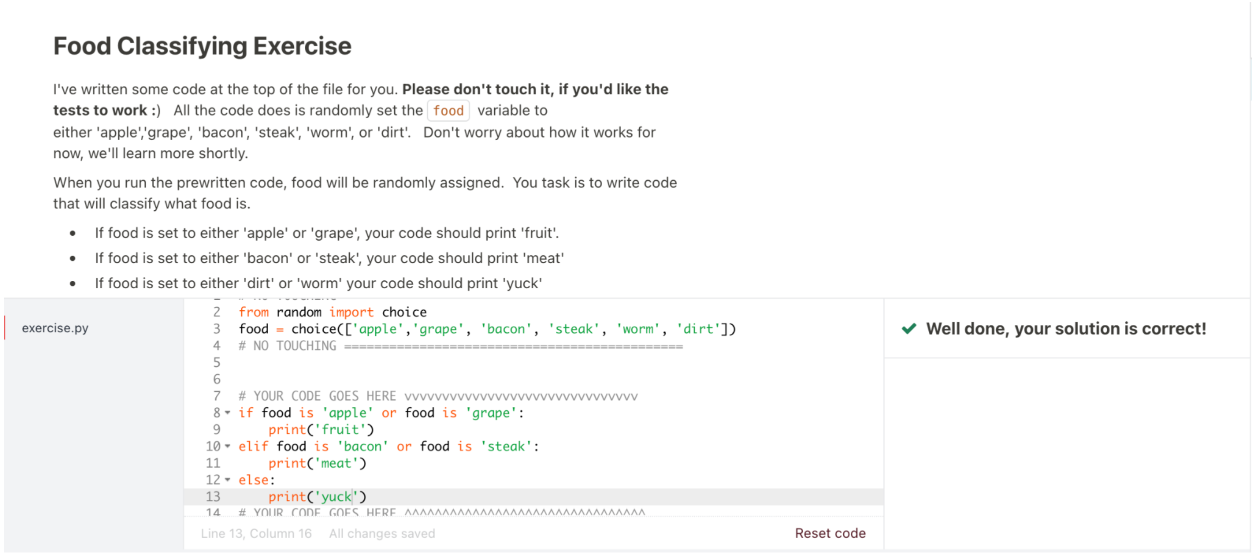
Logical Operators:
In Python, the following operators can be used to make Boolean logical comparisons or statements:

Operation	What it does	Example

and	Truthy if both **a** AND **b** are true (logical conjunction)	if a **and** b:
		    print(c)

or	Truthy if either **a** OR **b** are true (logical disjunction)	if a **or** b:
		    print(c)

not	Truthy if opposite of **a** is true (logical negation)	if **not** a:
		    print(b)



Food Classifying Exercise

I've written some code at the top of the file for you. **Please don't touch it, if you'd like the tests to work :)** All the code does is randomly set the `food` variable to either 'apple','grape', 'bacon', 'steak', 'worm', or 'dirt'. Don't worry about how it works for now, we'll learn more shortly.

When you run the prewritten code, food will be randomly assigned. You task is to write code that will classify what food is.

- If food is set to either 'apple' or 'grape', your code should print 'fruit'.
- If food is set to either 'bacon' or 'steak', your code should print 'meat'
- If food is set to either 'dirt' or 'worm' your code should print 'yuck'

exercise.py

```
from random import choice
food = choice(['apple','grape', 'bacon', 'steak', 'worm', 'dirt'])
# NO TOUCHING ==============================================


# YOUR CODE GOES HERE vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
if food is 'apple' or food is 'grape':
    print('fruit')
elif food is 'bacon' or food is 'steak':
    print('meat')
else:
    print('yuck')
# YOUR CODE GOES HERE ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

Line 13, Column 16 All changes saved Reset code

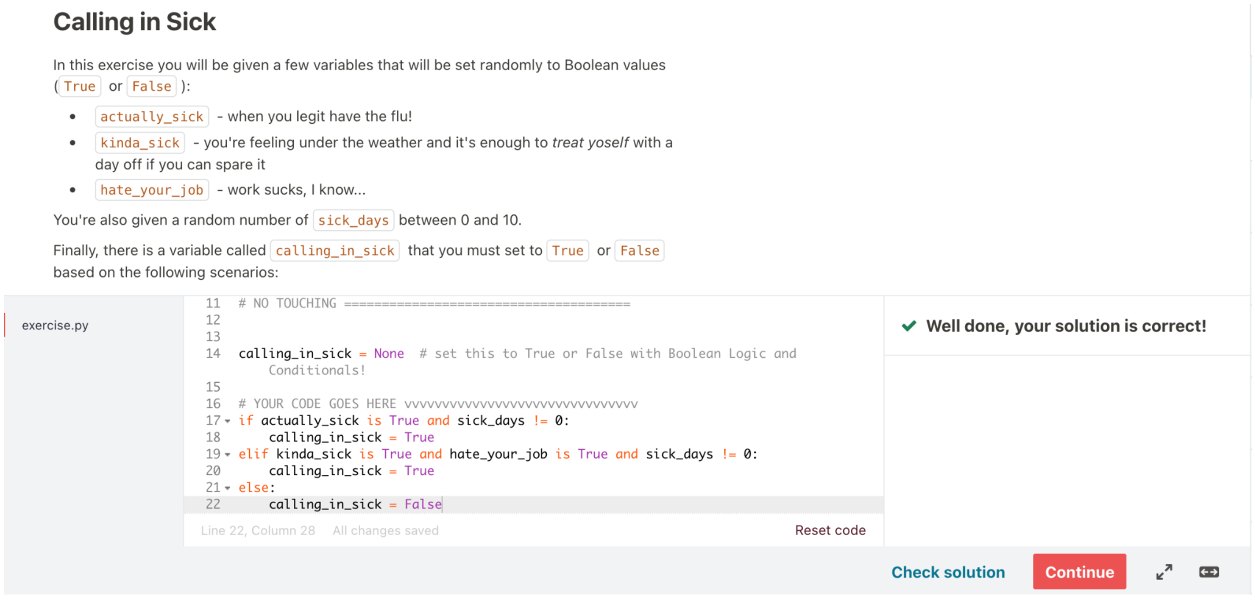
✔ **Well done, your solution is correct!**

<u>is vs ==:</u>

"It depends upon what the meaning of the word '*is*' is." - Bill Clinton

In Python, both *is* and == are very similar comparators; however, they are not the same.

```
i.e.
a = [1, 2, 3]
b = [1, 2, 3]
a == b        # returns True (checks if values are the same)
a is b        # returns False (checks if stored in the same place in memory)
```



Calling in Sick

In this exercise you will be given a few variables that will be set randomly to Boolean values (`True` or `False`):

- `actually_sick` - when you legit have the flu!
- `kinda_sick` - you're feeling under the weather and it's enough to *treat yoself* with a day off if you can spare it
- `hate_your_job` - work sucks, I know...

You're also given a random number of `sick_days` between 0 and 10.

Finally, there is a variable called `calling_in_sick` that you must set to `True` or `False` based on the following scenarios:

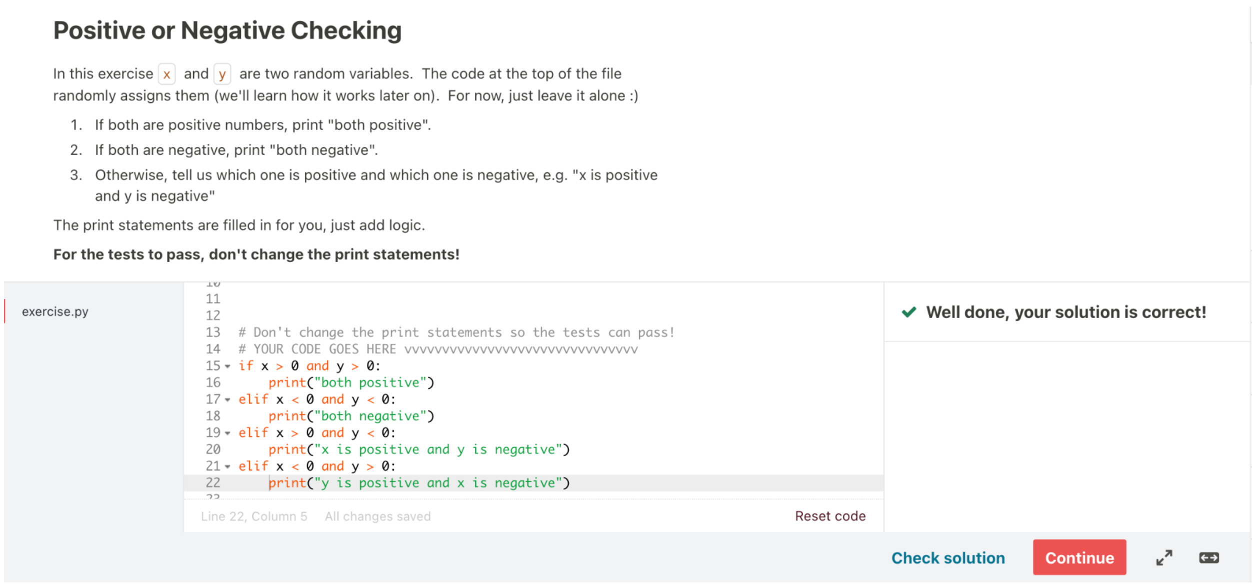
exercise.py

```
# NO TOUCHING ======================================


calling_in_sick = None  # set this to True or False with Boolean Logic and
    Conditionals!

# YOUR CODE GOES HERE vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
if actually_sick is True and sick_days != 0:
    calling_in_sick = True
elif kinda_sick is True and hate_your_job is True and sick_days != 0:
    calling_in_sick = True
else:
    calling_in_sick = False
```

Line 22, Column 28 All changes saved Reset code

✔ **Well done, your solution is correct!**

Check solution Continue



Positive or Negative Checking

In this exercise `x` and `y` are two random variables. The code at the top of the file randomly assigns them (we'll learn how it works later on). For now, just leave it alone :)

1. If both are positive numbers, print "both positive".
2. If both are negative, print "both negative".
3. Otherwise, tell us which one is positive and which one is negative, e.g. "x is positive and y is negative"

The print statements are filled in for you, just add logic.

For the tests to pass, don't change the print statements!

exercise.py

```python
# Don't change the print statements so the tests can pass!
# YOUR CODE GOES HERE vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
if x > 0 and y > 0:
    print("both positive")
elif x < 0 and y < 0:
    print("both negative")
elif x > 0 and y < 0:
    print("x is positive and y is negative")
elif x < 0 and y > 0:
    print("y is positive and x is negative")
```

Line 22, Column 5 All changes saved

✓ Well done, your solution is correct!

Looping in Python

Objectives:

- Understand what loops are and how they are useful
- Learn what an "iterable object" is
- Use ***for*** and ***while*** loops to iterate over ranges and strings
- Learn how to control exiting a loop

For Loops:

In Python, ***for*** loops are written like this:

```
for item in iterable_object:
        do something with item
```

An **iterable object** is some kind of collection of items, for instance: a list of numbers, a string of characters, a range, etc.

Item is a new variable that can be called whatever the user chooses. It references the current position of our **iterator** within the *iterable*. It will iterate over (run through) every item of the collection and then go away when it has visited all items.

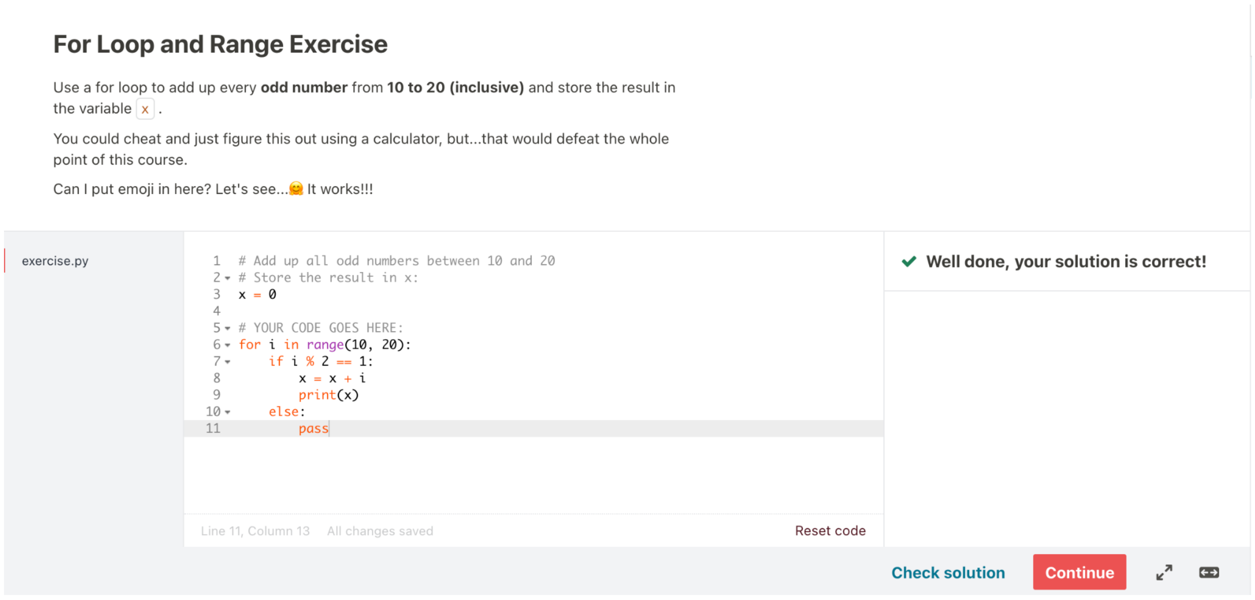
What is a range? A range is an immutable sequence of numbers and is commonly used for looping through an iterable object a specific number of times, i.e.

```
for number in range(1, 8):
        print(number)
```

More to read on ranges here:
https://docs.python.org/3/library/stdtypes.html#typesseq-range

Python ranges come in multiple forms:

- **range(7)** gives integers from 0 through 6 — *Count starts at 0 and is exclusive*
- **range(1, 8)** will give you integers from 1 to 7 — *Two parameters are (start, end)*
- **range(1, 10, 2)** will give you odds from 1 to 10 — *Third param indicates how many steps to skip. Also, which way to count, up + or down -*
- **range(7, 0, -1)** will give you integers from 7 to 1



For Loop and Range Exercise

Use a for loop to add up every **odd number** from **10 to 20 (inclusive)** and store the result in the variable `x` .

You could cheat and just figure this out using a calculator, but...that would defeat the whole point of this course.

Can I put emoji in here? Let's see...🤗 It works!!!

exercise.py

```python
# Add up all odd numbers between 10 and 20
# Store the result in x:
x = 0

# YOUR CODE GOES HERE:
for i in range(10, 20):
    if i % 2 == 1:
        x = x + i
        print(x)
    else:
        pass
```

Line 11, Column 13   All changes saved   Reset code

✔ Well done, your solution is correct!