Python Code Challenges: Control Flow

Python Code Challenges involving Control Flow

This article will help you review Python functions by providing some code challenges about control flow.

Some of these challenges are difficult! Take some time to think about them before starting to code.

You might not get the solution correct on your first try — look at your output, try to find where you're going wrong, and iterate on your solution.

Finally, if you get stuck, use our solution code! If you "Check Answer" twice with an incorrect solution, you should see an option to get our solution code. However, truly investigate that solution — experiment and play with the solution code until you have a good grasp of how it is working. Good luck!

Function Syntax

As a refresher, function syntax looks like this:

```
def some_function(some_input1, some_input2):
  # … do something with the inputs …
  return output
```

For example, a function that returns the sum of the first and last elements of a given list might look like this:

```
def first_plus_last(lst):
  return lst[0] + lst[-1]
```

And this would produce output like:

```
>>> first_plus_last([1, 2, 3, 4])
5
>>> first_plus_last([8, 2, 5, -8])
0
>>> first_plus_last([-10, 2, 3, -4])
-14
```

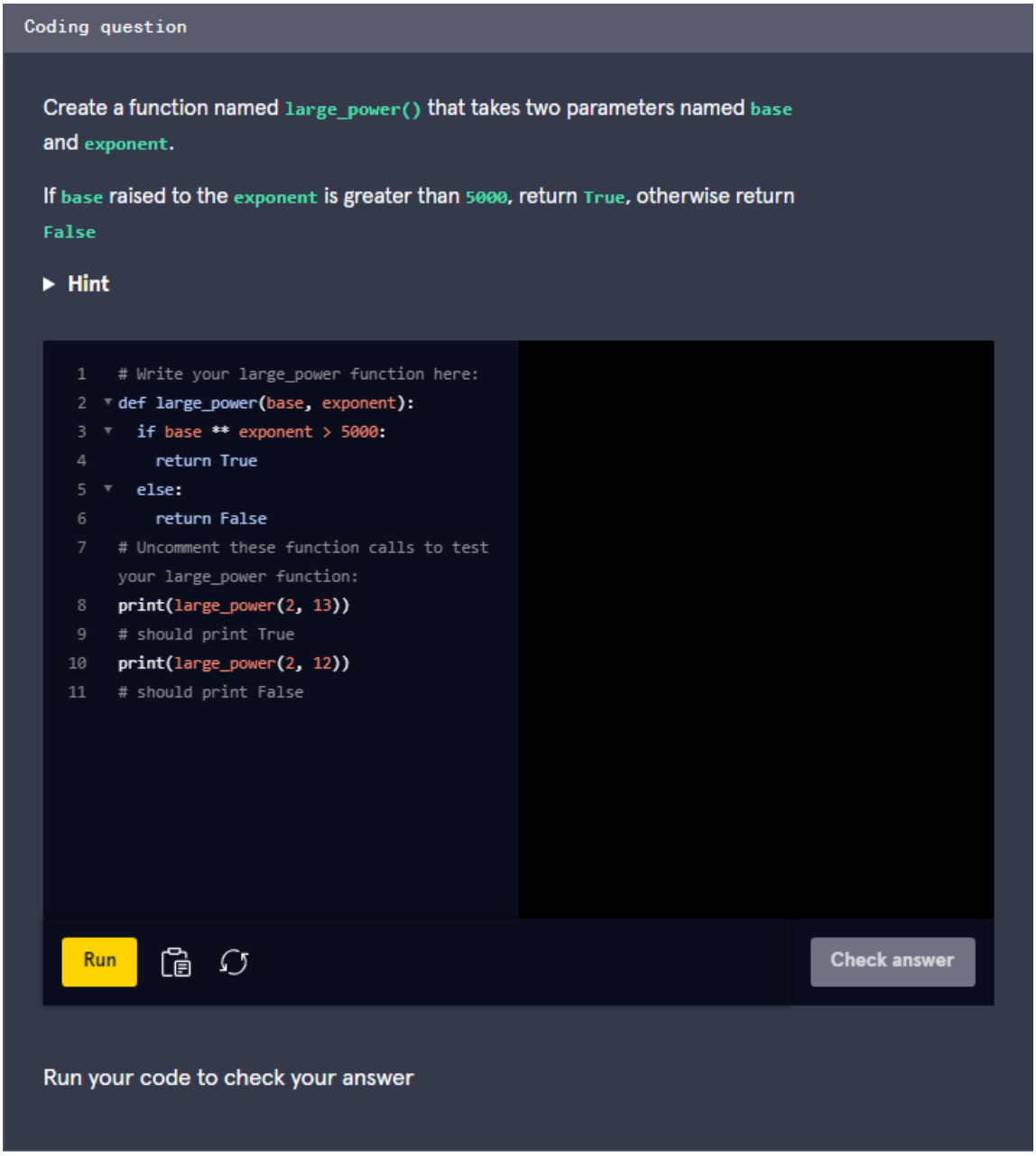
Challenges

We've included 5 challenges below. Try to answer all of them and polish up your problem-solving skills and your control flow expertise.

1. Large Power

For the first code challenge, we are going to create a method that tests whether the result of taking the power of one number to another number provides an answer which is greater than 5000. We will use a conditional statement to return `True` if the result is greater than 5000 or return `False` if it is not. In order to accomplish this, we will need the following steps:

1. Define the function to accept two input parameters called `base` and `exponent`
2. Calculate the result of `base` to the power of `exponent`
3. Use an `if` statement to test if the result is greater than 5000. If it is then return `True`. Otherwise, return `False`



This is how we solved it:

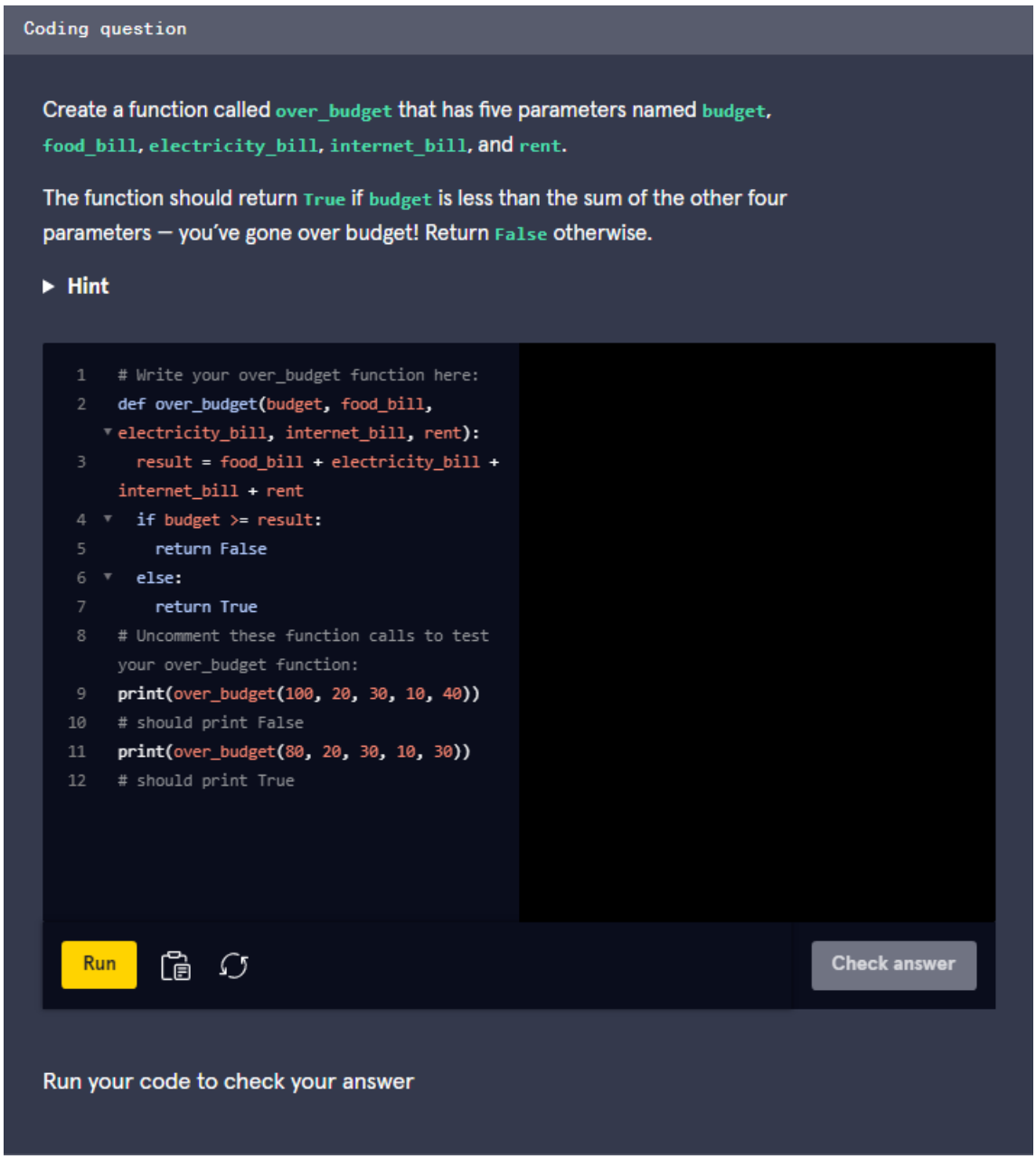
```
def large_power(base, exponent):
  if base ** exponent > 5000:
    return True
  else:
    return False
```

In this solution, we have an example of how the operation can be performed in the condition of the `if` statement. This prevents us from needing to create an extra variable. If the condition is true, then the indented code is executed which returns `True`, otherwise the indented code in the `else` statement is executed.

2. Over Budget

Let's say we are trying to save some money and we are watching our budget. We need to make sure that the result of our spending is less than the total amount we have allocated for each of the categories. Our function will accept a parameter called `budget` which describes our spending limit. The next four parameters describe what we are spending our money on. We need to sum all of our spendings and compare it to the budget. If we have gone over budget, we will return `True`. Otherwise we return `False`. Here are the steps we need:

1. Define the function to accept five parameters starting with `budget` then `food_bill`, `electricity_bill`, `internet_bill`, and `rent`
2. Calculate the sum of the last four parameters
3. Use `if` and `else` statements to test if the budget is less than the sum of the calculated sum from the previous step.
4. If the condition is true, return `True` otherwise return `False`



This is one way to solve it:

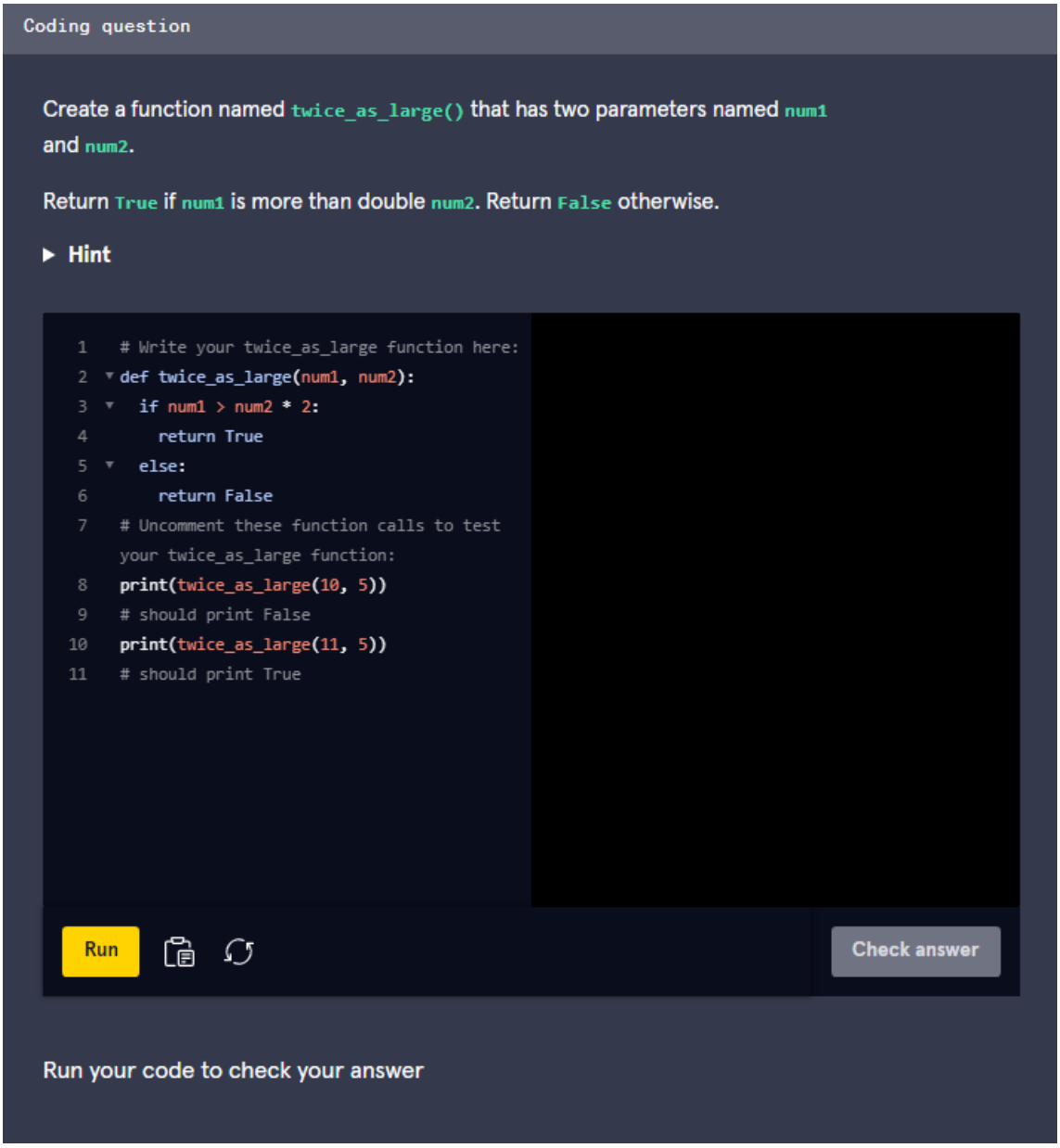
```
def over_budget(budget, food_bill, electricity_bill, internet_bill, rent):
  if (budget < food_bill + electricity_bill + internet_bill + rent):
    return True
  else:
    return False
```

Similarly to the last problem, we can perform the operations within the condition of the `if` statement to prevent us from creating an extra variable. We calculate the sum and compare it to `budget` at the same time and return `True` if the condition is met, otherwise we return `False`.

3. Twice As Large

In this challenge, we will determine if one number is twice as large as another number. To do this, we will compare the first number with two times the second number. Here are the steps:

1. Define our function with two inputs `num1` and `num2`
2. Multiply the second input by 2
3. Use an `if` statement to compare the result of the last calculation with the first input
4. If num1 is greater then return `True` otherwise return `False`



Here is this solution:

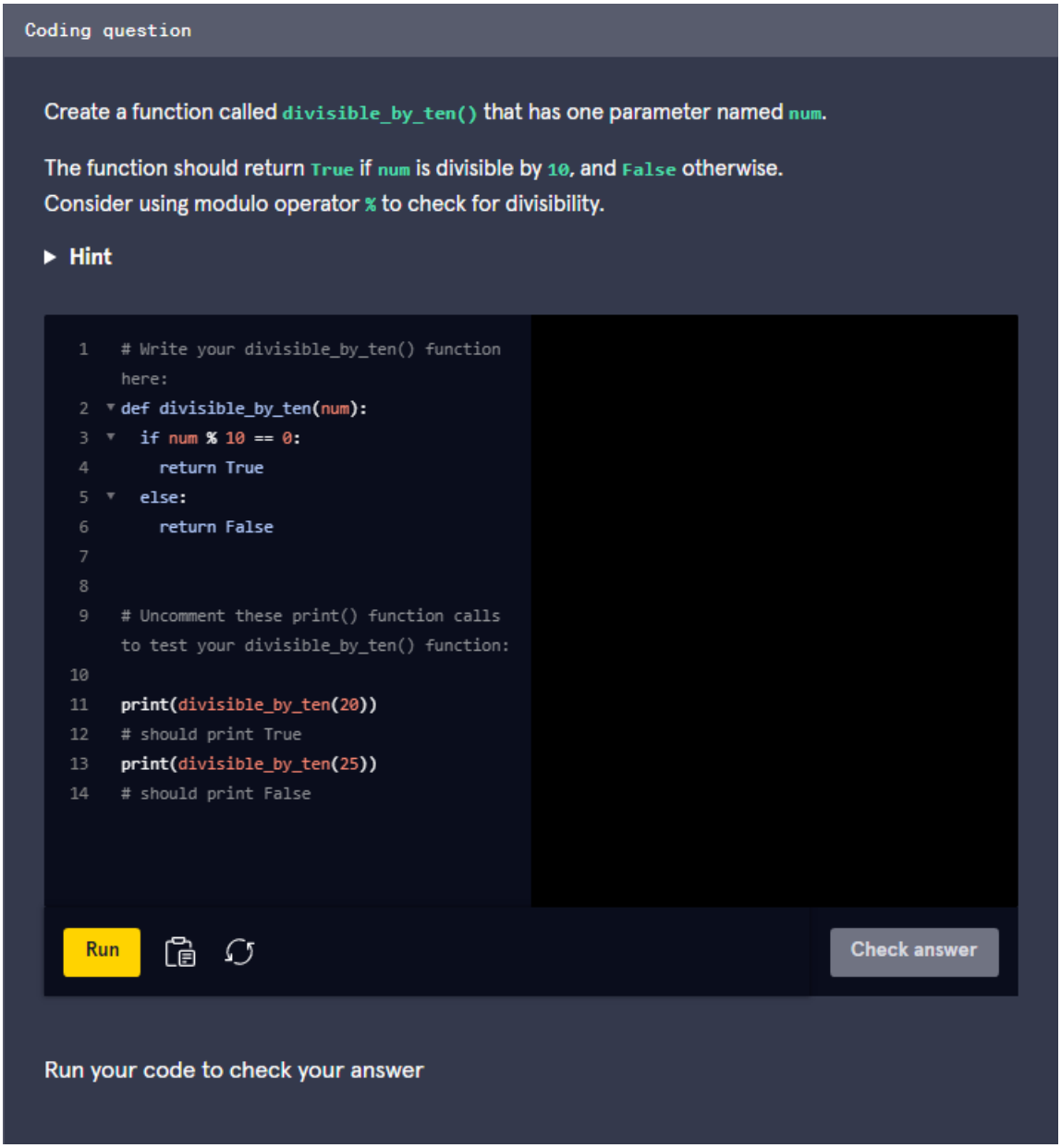
```
def twice_as_large(num1, num2):
  if num1 > 2 * num2:
    return True
  else:
    return False
```

In this function, we also performed the operation within the condition of the `if` statement. The second input is multiplied by 2 and then compared to the first input on the same line.

4. Divisible By Ten

To make things a bit more challenging, we are going to create a function that determines whether or not a number is divisible by ten. A number is divisible by ten if the remainder of the number divided by 10 is 0. Using this, we can complete this function in a few steps:

1. Define the function header to accept one input `num`
2. Calculate the remainder of the input divided by 10 (use modulus)
3. Use an `if` statement to check if the remainder was 0. If the remainder was 0, return `True`, otherwise, return `False`



Here's one solution:

```
def divisible_by_ten(num):
  if (num % 10 == 0):
    return True
```

```
  else:
    return False
```

In this solution, we perform the modulus operation within the condition of the if statement. We test if the result is equal to 0 and if it is, then we return `True` otherwise we return `False`.

5. Not Sum To Ten

Finally, we are going to check if the summation of two inputs does not equal ten. Our function will accept two inputs and add them together. If the two numbers added together are not equal to ten, then we will return `True`, otherwise, we will return `False`. Here is what we need to do:

1. Define the function to accept two parameters, `num1` and `num2`
2. Add the two parameters together
3. Test if the result is not equal to 10
4. If the sum is not equal, return `True`, otherwise, return `False`

Coding question

Create a function named `not_sum_to_ten()` that has two parameters named `num1` and `num2`.

Return `True` if `num1` and `num2` do not sum to `10`. Return `False` otherwise.

▶ Hint

```
# Write your not_sum_to_ten function here:
def not_sum_to_ten(num1, num2):
  if (num1 + num2) != 10:
    return True
  else:
    return False
# Uncomment these function calls to test your not_sum_to_ten function:
print(not_sum_to_ten(9, -1))
# should print True
print(not_sum_to_ten(9, 1))
# should print False
print(not_sum_to_ten(5,5))
# should print False
```

Run

Check answer

Run your code to check your answer

Here is how we solved this one:

```
def not_sum_to_ten(num1, num2):
  if (num1 + num2 != 10):
    return True
  else:
    return False
```

We begin by adding our parameters within the condition of the `if` statement. Next, we use `!=` to compare the sum to 10. If they are not equal then our function returns `True`, otherwise it returns `False`.