

- (d) This produces an error.
- (e) None of the above.

15. The following code is executed.

```
def inc_by_two(x):
    x = x + 2
    return x

x = 10
inc_by_two(x)
print("x = ", x)
```

What is the output produced by the `print()` statement?

ANSWERS: 1) b; 2) True; 3) d, `q` is not defined outside of the function; 4) d, the function is a void function and hence `q` is None; 5) b; 6) e, the output is None since this is a void function; 7) c; 8) a, `def` is a keyword and we cannot assign a value to it; 9) d, the variable `x` is not defined in `main()`; 10) c; 11) d, `fl()` is a void function; 12) b; 13) b; 14) e, `z` would be assigned 5.0; 15) `x = 10`.

4.11 Exercises

- Write a function called `convert_to_days()` that takes no parameters. Have your function prompt the user to input numbers of hours, minutes, and seconds. Write a helper function called `get_days()` that uses these values and converts them to days in float form (fractions of a day are allowed). `get_days()` should return the number of days. Use this helper function within the `convert_to_days()` function to display the numbers of days to the user. The built-in function `round()` takes two arguments: a number and an integer indicating the desired precision (i.e., the desired number of digits beyond the decimal point). Use this function to round the number of days four digits after the decimal point.

The following demonstrates the proper behavior of `convert_to_days()`:

```
1 >>> convert_to_days()
2 Enter number of hours: 97
3 Enter number of minutes: 54
4 Enter number of seconds: 45
5
6 The number of days is: 4.0797
```

- Write a function called `calc_weight_on_planet()` that calculates your equivalent weight on another planet. This function takes two arguments: your weight on Earth in pounds and the surface gravity of the planet of interest with units m/s^2 . Make the second argument optional and supply a default value of 23.1 m/s^2 which is the approximate surface gravity of Jupiter (Earth's surface gravity is approximately 9.8 m/s^2). To perform the conversion, use

the equation: weight is equal to mass times surface gravity. Since your weight on Earth is given and you know the Earth's surface gravity, have your function use this information to calculate your mass (it is fine if, at this point, the units of mass are a mix of Imperial and the MKS system). Then, use your mass and the given surface gravity to calculate your effective weight on the other planet.

The following demonstrates the proper behavior of this function:

```
1 >>> calc_weight_on_planet(120, 9.8)
2 120.0
3 >>> calc_weight_on_planet(120)
4 282.85714285714283
5 >>> calc_weight_on_planet(120, 23.1)
6 282.85714285714283
```

3. Write a function called `num_atoms()` that calculates how many atoms are in n grams of an element given its atomic weight. This function should take two parameters: the amount of the element in grams and an optional argument representing the atomic weight of the element. The atomic weight of any particular element can be found on a periodic table but make the default value for the optional argument the atomic weight of gold (Au) 196.97 with units in grams/mole. A mole is a unit of measurement that is commonly used in chemistry to express an amount of a substance. Avogadro's number is a constant, 6.022×10^{23} atoms/mole, that can be used to find the number of atoms in a given sample. Use Avogadro's number, the atomic weight, and the amount of the element in grams to find the number of atoms present in the sample. Your function should return this value.

The following demonstrates the proper behavior of this function using 10 grams and the atomic weight of gold (default), carbon, and hydrogen:

```
1 >>> num_atoms(10)
2 3.0573183733563486e+22
3 >>> num_atoms(10, 12.001)
4 5.017915173735522e+23
5 >>> num_atoms(10, 1.008)
6 5.97420634920635e+24
```

4. The aspect ratio of an image describes the relationship between the width and height. Aspect ratios are usually expressed as two numbers separated by a colon that represent width and height respectively. Common aspect ratios in photography are 4:3, 3:2, and 16:9. An image that has an aspect ratio of $x : y$ means that for every x inches of width you will have y inches of height no matter the size of the image (and, of course, you can use any unit of length, not just inches, and even abstract units such as pixels). Suppose you are writing a blog and you have an image that is m units wide and n units high but your blog only has space for an image that is z units wide (where z is less than m). Write a function called `calc_new_height()` that returns the height the image must be in order to preserve the aspect ratio (i.e., a height that will not distort the image). This function takes no arguments and prompts the user for the current width, the current height, and the desired new width. In addition to *returning* the

new height, this function also prints the value. The new height is a `float` regardless of the types of the values the user entered.

The following demonstrates the proper behavior of this function:

```

1 >>> calc_new_height()
2 Enter the current width: 800
3 Enter the current height: 560
4 Enter the desired width: 370
5
6 The corresponding height is: 259.0
7 259.0

```

5. Write a function called `convert_temp()` that takes no arguments. This function obtains a temperature in Fahrenheit from the user and uses two helper functions to convert this temperature to Celsius and Kelvin. Write a helper function called `convert_to_celsius()` that takes a single argument in Fahrenheit and returns the temperature in Celsius using the formula

$$T_c = \frac{5}{9}(T_f - 32).$$

where T_c is the temperature in Celsius and T_f is the temperature in Fahrenheit. Write another helper function called `convert_to_kelvin()` that takes a single argument in degrees Celsius and returns degrees Kelvin using the formula

$$T_k = T_c + 273.15.$$

where T_k is the temperature in Kelvin. Use these two functions within your `convert_temp()` function to display (i.e., print) the temperatures for the user. The `convert_temp()` does not return anything.

The following demonstrates the proper behavior of this function:

```

1 >>> convert_temp()
2 Enter a temperature in Fahrenheit: 32
3
4 The temperature in Fahrenheit is: 32
5 The temperature in Celsius is: 0.0
6 The temperature in Kelvin is: 273.15
7 >>> convert_temp()
8 Enter a temperature in Fahrenheit: 80
9
10 The temperature in Fahrenheit is: 80
11 The temperature in Celsius is: 26.666666666666668
12 The temperature in Kelvin is: 299.81666666666666

```