Scientific Computing with Python

Python for Everybody

1. Introduction: Why Program?
2. Introduction: Hardware Architecture
3. Introduction: Python as a Language
4. Introduction: Elements of Python
5. Variables, Expressions, and Statements
6. Intermediate Expressions
7. Conditional Execution
8. More Conditional Structures
9. Python Functions
10. Build your own Functions
11. Loops and Iterations
12. Iterations: Definite Loops
13. Iterations: Loop Idioms
14. Iterations: More Patterns
15. Strings in Python
16. Intermediate Strings
17. Reading Files
18. Files as a Sequence
19. Python Lists
20. Working with Lists
21. Strings and Lists
22. Python Dictionaries
23. Dictionaries: Common Applications
24. Dictionaries and Loops
25. The Tuples Collection
26. Comparing and Sorting Tuples
27. Regular Expressions
28. Regular Expressions: Matching and Extracting Data
29. Regular Expressions: Practical Applications
30. Networking with Python
31. Networking Protocol
32. Networking: Write a Web Browser
33. Networking: Text Processing
34. Networking: Using urllib in Python
35. Networking: Web Scraping with Python
36. Using Web Services
37. Web Services: XML
38. Web Services: XML Schema
39. Web Services: JSON
40. Web Services: Service Oriented Approach
41. Web Services: APIs
42. Web Services: API Rate Limiting and Security
43. Python Objects
44. Objects: A Sample Class
45. Object Lifecycle
46. Objects: Inheritance
47. Relational Databases and SQLite
48. Make a Relational Database
49. Relational Database Design
50. Representing Relationships in a Relational Database
51. Relational Databases: Relationship Building
52. Relational Databases: Join Operation
53. Relational Databases: Many-to-many Relationships
54. Visualizing Data with Python
55. Data Visualization: Page Rank
56. Data Visualization: Mailing Lists

Projects

1. Arithmetic Formatter
2. Time Calculator
3. Budget App
4. Polygon Area Calculator
5. Probability Calculator

Here are some videos on the freeCodeCamp.org YouTube channel that will teach you everything you need to know to complete these projects:

- Python for Everybody Video Course (14 hours)
- Learn Python Video Course (4 hours)

1. Arithmetic Formatter

Create a function that receives a list of strings that are arithmetic problems and returns the problems arranged vertically and side-by-side.

Assignment

Students in primary school often arrange arithmetic problems vertically to make them easier to solve.

For example, "235 + 52" becomes:

```
  235
+  52
-----
```

Create a function that receives a list of strings that are arithmetic problems and returns the problems arranged vertically and side-by-side.

The function should optionally take a second argument. When the second argument is set to True, the answers should be displayed.

For example:

Function Call:

```
arithmetic_arranger(["32 + 698", "3801 - 2", "45 + 43", "123 + 49"])
```

Output:

```
   32      3801      45      123
+ 698    -    2    + 43    +  49
-----    ------    ----    -----
```

Function Call:

```
arithmetic_arranger(["32 + 8", "1 - 3801", "9999 + 9999", "523 - 49"], True)
```

Output:

```
  32         1      9999      523
+  8    - 3801    + 9999    -  49
----    ------    ------    -----
  40     -3800     19998      474
```

Rules

The function will return the correct conversion if the supplied problems are properly formatted, otherwise, it will return a string that describes an error that is meaningful to the user.

Situations that will return an error:

- If there are too many problems supplied to the function. The limit is five, anything more will return: Error: Too many problems.
- The appropriate operators the function will accept are addition and subtraction. Multiplication and division will return an error. Other operators not mentioned in this bullet point will not need to be tested. The error returned will be: Error: Operator must be '+' or '-'.
- Each number (operand) should only contain digits. Otherwise, the function will return: Error: Numbers must only contain digits.
- Each operand (aka number on each side of the operator) has a max of four digits in width. Otherwise, the error string returned will be: Error: Numbers cannot be more than four digits.

If the user supplied the correct format of problems, the conversion you return will follow these rules:

- There should be a single space between the operator and the longest of the two operands, the operator will be on the same line as the second operand, both operands will be in the same order as provided (the first will be the top one and the second will be the bottom.

- Numbers should be right-aligned.

- There should be four spaces between each problem.

- There should be dashes at the bottom of each problem. The dashes should run along the entire length of each problem individually. (The example above shows what this should look like.)

Development

- Write your code in `arithmetic_arranger.py`.
- For development, you can use `main.py` to test your `arithmetic_arranger()` function.
- Click the "run" button and `main.py` will run.

Testing

The unit tests for this project are in `test_module.py`. We imported the tests from `test_module.py` to `main.py` for your convenience. The tests will run automatically whenever you hit the "run" button.

2. Time Calculator

Write a function named "add_time" that can add a duration to a start time and return the result.

Assignment

Write a function named add_time that takes in two required parameters and one optional parameter:

- a start time in the 12-hour clock format (ending in AM or PM)
- a duration time that indicates the number of hours and minutes
- (optional) a starting day of the week, case insensitive

The function should add the duration time to the start time and return the result.

If the result will be the next day, it should show (next day) after the time. If the result will be more than one day later, it should show (n days later) after the time, where "n" is the number of days later.

If the function is given the optional starting day of the week parameter, then the output should display the day of the week of the result. The day of the week in the output should appear after the time and before the number of days later.

Below are some examples of different cases the function should handle. Pay close attention to the spacing and punctuation of the results.

Examples

```
add_time("3:00 PM", "3:10")
# Returns: 6:10 PM

add_time("11:30 AM", "2:32", "Monday")
# Returns: 2:02 PM, Monday

add_time("11:43 AM", "00:20")
# Returns: 12:03 PM

add_time("10:10 PM", "3:30")
# Returns: 1:40 AM (next day)

add_time("11:43 PM", "24:20", "tueSday")
# Returns: 12:03 AM, Thursday (2 days later)

add_time("6:30 PM", "205:12")
# Returns: 7:42 AM (9 days later)
```

Do not import any Python libraries. Assume that the start times are valid times. The minutes in the duration time will be a whole number less than 60, but the hour can be any whole number.

Development

Write your code in time_calculator.py. For development, you can use main.py to test your time_calculator() function. Click the "run" button and main.py will run.

Testing

The unit tests for this project are in test_module.py. We imported the tests from test_module.py to main.py for your convenience. The tests will run automatically whenever you hit the "run" button.

3. Budget App

Create a "`Category`" class that can be used to create different budget categories.

Assignment

Complete the `Category` class in `budget.py`. It should be able to instantiate objects based on different budget categories like *food*, *clothing*, and *entertainment*. When objects are created, they are passed in the name of the category. The class should have an instance variable called `ledger` that is a list. The class should also contain the following methods:

- A `deposit` method that accepts an amount and description. If no description is given, it should default to an empty string. The method should append an object to the ledger list in the form of `{"amount": amount, "description": description}`.

- A `withdraw` method that is similar to the `deposit` method, but the amount passed in should be stored in the ledger as a negative number. If there are not enough funds, nothing should be added to the ledger. This method should return `True` if the withdrawal took place, and `False` otherwise.

- A `get_balance` method that returns the current balance of the budget category based on the deposits and withdrawals that have occurred.

- A `transfer` method that accepts an amount and another budget category as arguments. The method should add a withdrawal with the amount and the description "*Transfer to [Destination Budget Category]*". The method should then add a deposit to the other budget category with the amount and the description "*Transfer from [Source Budget Category]*". If there are not enough funds, nothing should be added to either ledgers. This method should return `True` if the transfer took place, and `False` otherwise.

- A `check_funds` method that accepts an amount as an argument. It returns `False` if the amount is greater than the balance of the budget category and returns `True` otherwise. This method should be used by both the `withdraw` method and `transfer` method.

When the budget object is printed it should display:

- A title line of 30 characters where the name of the category is centered in a line of `*` characters.
- A list of the items in the ledger. Each line should show the description and amount. The first 23 characters of the description should be displayed, then the amount. The amount should be right aligned, contain two decimal places, and display a maximum of 7 characters.
- A line displaying the category total.

Here is an example of the output:

```
*************Food*************
initial deposit        1000.00
groceries               -10.15
restaurant and more foo -15.89
Transfer to Clothing    -50.00
Total: 923.96
```

Besides the Category class, create a function (ouside of the class) called create_spend_chart that takes a list of categories as an argument. It should return a string that is a bar chart.

The chart should show the percentage spent in each category passed in to the function. The percentage spent should be calculated only with withdrawals and not with deposits. Down the left side of the chart should be labels 0 - 100. The "bars" in the bar chart should be made out of the "o" character. The height of each bar should be rounded down to the nearest 10. The horizontal line below the bars should go two spaces past the final bar. Each category name should be vertically below the bar. There should be a title at the top that says "Percentage spent by category”.

This function will be tested with up to four categories.

Look at the example output below very closely and make sure the spacing of the output matches the example exactly.

```
Percentage spent by category
100|
 90|
 80|
 70|
 60| o
 50| o
 40| o
 30| o
 20| o  o
 10| o  o  o
  0| o  o  o
    ----------
     F  C  A
     o  l  u
     o  o  t
     d  t  o
        h
        i
        n
        g
```

The unit tests for this project are in test_module.py.

Development

Write your code in budget.py. For development, you can use main.py to test your Category class. Click the "run" button and main.py will run.

Testing

We imported the tests from test_module.py to main.py for your convenience. The tests will run automatically whenever you hit the "run" button.

4. Polygon Area Calculator

In this project you will use object oriented programming to create a Rectangle class and a Square class. The Square class should be a subclass of Rectangle and inherit methods and attributes.

Assignment

In this project you will use object oriented programming to create a Rectangle class and a Square class. The Square class should be a subclass of Rectangle and inherit methods and attributes.

Rectangle class

When a Rectangle object is created, it should be initialized with `width` and `height` attributes. The class should also contain the following methods:

- `set_width`
- `set_height`
- `get_area`: Returns area (`width * height`)
- `get_perimeter`: Returns perimeter (`2 * width + 2 * height`)
- `get_diagonal`: Returns diagonal (`(width ** 2 + height ** 2) ** .5`)
- `get_picture`: Returns a string that represents the shape using lines of "*". The number of lines should be equal to the height and the number of "*" in each line should be equal to the width. There should be a new line (`\n`) at the end of each line. If the width or height is larger than 50, this should return the string: "Too big for picture.”.
- `get_amount_inside`: Takes another shape (square or rectangle) as an argument. Returns the number of times the passed in shape could fit inside the shape (with no rotations). For instance, a rectangle with a width of 4 and a height of 8 could fit in two squares with sides of 4.

Additionally, if an instance of a Rectangle is represented as a string, it should look like: `Rectangle(width=5, height=10)`

Square class

The Square class should be a subclass of Rectangle. When a Square object is created, a single side length is passed in. The `__init__` method should store the side length in both the `width` and `height` attributes from the Rectangle class.

The Square class should be able to access the Rectangle class methods but should also contain a `set_side` method. If an instance of a Square is represented as a string, it should look like: `Square(side=9)`

Additionally, the `set_width` and `set_height` methods on the Square class should set both the width and height.

Usage example

```
rect = shape_calculator.Rectangle(10, 5)
print(rect.get_area())
rect.set_height(3)
print(rect.get_perimeter())
print(rect)
print(rect.get_picture())

sq = shape_calculator.Square(9)
print(sq.get_area())
sq.set_side(4)
print(sq.get_diagonal())
print(sq)
print(sq.get_picture())

rect.set_height(8)
rect.set_width(16)
print(rect.get_amount_inside(sq))
```

That code should return:

```
50
26
Rectangle(width=10, height=3)
**********
**********
**********

81
5.656854249492381
Square(side=4)
****
****
****
****

8
```

The unit tests for this project are in test_module.py.

Development

Write your code in shape_calculator.py. For development, you can use main.py to test your shape_calculator() function. Click the "run" button and main.py will run.

Testing

We imported the tests from test_module.py to main.py for your convenience. The tests will run automatically whenever you hit the "run" button.

5. Probability Calculator

Write a program to determine the approximate probability of drawing certain balls randomly from a hat.

Assignment

Suppose there is a hat containing 5 blue balls, 4 red balls, and 2 green balls. What is the probability that a random draw of 4 balls will contain at least 1 red ball and 2 green balls? While it would be possible to calculate the probability using advanced mathematics, an easier way is to write a program to perform a large number of experiments to estimate an approximate probability.

For this project, you will write a program to determine the approximate probability of drawing certain balls randomly from a hat.

First, create a Hat class in prob_calculator.py. The class should take a variable number of arguments that specify the number of balls of each color that are in the hat. For example, a class object could be created in any of these ways:

```
hat1 = Hat(yellow=3, blue=2, green=6)
hat2 = Hat(red=5, orange=4)
hat3 = Hat(red=5, orange=4, black=1, blue=0, pink=2, striped=9)
```

A hat will always be created with at least one ball. The arguments passed into the hat object upon creation should be converted to a contents instance variable. contents should be a list of strings containing one item for each ball in the hat. Each item in the list should be a color name representing a single ball of that color. For example, if your hat is {"red": 2, "blue": 1}, contents should be ["red", "red", “blue"].

The Hat class should have a draw method that accepts an argument indicating the number of balls to draw from the hat. This method should remove balls at random from contents and return those balls as a list of strings. The balls should not go back into the hat during the draw, similar to an urn experiment without replacement. If the number of balls to draw exceeds the available quantity, return all the balls.

Next, create an experiment function in prob_calculator.py (not inside the Hat class). This function should accept the following arguments:

- hat: A hat object containing balls that should be copied inside the function.
- expected_balls: An object indicating the exact group of balls to attempt to draw from the hat for the experiment. For example, to determine the probability of drawing 2 blue balls and 1 red ball from the hat, set expected_balls to {"blue":2, “red":1}.
- num_balls_drawn: The number of balls to draw out of the hat in each experiment.
- num_experiments: The number of experiments to perform. (The more experiments performed, the more accurate the approximate probability will be.)

The experiment function should return a probability.

For example, let's say that you want to determine the probability of getting at least 2 red balls and 1 green ball when you draw 5 balls from a hat containing 6 black, 4 red, and 3 green. To do this, we perform `N` experiments, count how many times `M` we get at least 2 red balls and 1 green ball, and estimate the probability as `M/N`. Each experiment consists of starting with a hat containing the specified balls, drawing a number of balls, and checking if we got the balls we were attempting to draw.

Here is how you would call the `experiment` function based on the example above with 2000 experiments:

```
hat = Hat(black=6, red=4, green=3)
probability = experiment(hat=hat,
                  expected_balls={"red":2,"green":1},
                  num_balls_drawn=5,
                  num_experiments=2000)
```

Since this is based on random draws, the probability will be slightly different each time the code is run.

Hint
Consider using the modules that are already imported at the top of `prob_calculator.py`.

Development

Write your code in `prob_calculator.py`. For development, you can use `main.py` to test your code. Click the "run" button and `main.py` will run.

Testing

The unit tests for this project are in `test_module.py`. We imported the tests from `test_module.py` to `main.py` for your convenience. The tests will run automatically whenever you hit the "run" button.