# A0 - Square Root

You now understand that Python statements can be algebraic equations. Write a function to return the square root of a positive number. Use an algebraic equation to compute the square root. Use the template to write your program in.

After you are satisfied that the program runs correctly, save the contents of the cell in a new file named *square_root.py*. gedit is a basic text editor in Linux. You could copy and paste the contents of the cell inside it. You've created your first Python program.

Submit the *square_root.py* program through Autolab. Login to Autolab through IRIS. Autolab will run your program against a few inputs (shown in a Table below) and check if the outputs generated are as expected. Go through the Autolab output and try to make sense

**Square root Template (square_root.py)**

```
'''
CS101. A0 - Square root program
'''

def square_root(n):
      """Calculate and return the square root of a positive integer

      :param n: int positive number.
      :return: square root of n.

      Function that takes in a positive integer, calculates its square root
      using a simple algebraic equation. Returns the result.
      """

      pass
```

**Test Cases**

| **Function** | **Inputs** | **Output** | **Remarks** |
|---|---|---|---|
| square_root | 100 | 10.0 | |
| square_root | 1 | 1.0 | |
| square_root | 2 | 1.4142135623730951 | |
| square_root | 17 | 4.123105625617661 | |
| square_root | 1024 | 32.0 | |
| square_root | 6241 | 79.0 | |

*****

A1 - Cookbook

You're going to write some code to help you cook a sweet recipe from your favorite cookbook. You have five tasks, all related to cooking your recipe.

1. Define expected cooking time in minutes
2. Calculate remaining cooking time in minutes
3. Calculate preparation time in minutes
4. Calculate total cooking time (preparation + cooking time) in minutes
5. Update the recipe with notes. Go back through the recipe, adding notes and documentation

**1. Define expected cooking time in minutes**
Define an EXPECTED_COOK_TIME constant that returns how many minutes the ingredients should spend in the cooking vessel. According to your cookbook, the recipe needs around 40 minutes cooking time
>>> EXPECTED_COOK_TIME
40

**2. Calculate remaining cooking time in minutes**
Implement the cooking_time_remaining() function that takes the actual minutes passed since start of cooking as an argument and returns how many minutes the ingredients still need to cook based on the EXPECTED_COOK_TIME.
>>> cooking_time_remaining(30)
10

**3. Calculate preparation time in minutes**
Preparation time is the time spent before the cooking on fire begins. Implement the preparation_time_in_minutes() function that takes the number of pieces you want to make as an argument. This function returns how many minutes you would spend preparing the ingredients for making those many finished sweets. Assume that a batch of 10 takes 5 minutes to prepare.
Assume: number of pieces is a perfect multiple of 10 always
>>> preparation_time_required_in_minutes(20)
10

**4. Calculate the number of minutes remaining**
Implement the remaining_time_in_minutes() function that has two parameters: number_of_pieces (the number of sweet pieces being prepared) and elapsed_time (the number of minutes elapsed till now). This function returns the number of minutes remaining till the sweets are fully done.

>>> remaining_time_in_minutes(20, 25)
25

**5. Update the recipe with notes. Go back through the recipe, adding notes and documentation.**

```
def remaining_time_in_minutes(number_of_pieces, elapsed_time):
    """
    Return remaining cooking time.
    This function takes two input parameters: number of sweet pieces and
    time elapsed till now.
    The function returns the remaining time for the sweets to be ready
    """
```

**6. Error Checking.**
num_pieces cannot be 0, or negative

If num_pieces is input as 0 or negative, raise the following error
if num_pieces < 1:
    raise ValueError("Number of pieces should be greater than Zero.")

**Cookbook Template (cookbook.py)**

```
'''
Advanced Python. Numbers.
The Sweet Recipe Question
'''

# insert the value
EXPECTED_COOK_TIME=

def cooking_time_remaining(elapsed_cooking_time):
    """Calculate the cooking time remaining.

    :param elapsed_cooking_time: int cooking time already elapsed.
    :return: int remaining cooking time derived from 'EXPECTED_COOK_TIME'.

    Function that takes the actual minutes the sweet has been in the cooking pan
    on fire as an argument and returns how many minutes the sweet still needs to cook
    based on the `EXPECTED_COOK_TIME`.
    """

    pass

def preparation_time_required_in_minutes(num_pieces):
    """
    Calculate preparation time for the given number of pieces
    Preparation time is (num_pieces // BATCH_SIZE) * PREPARATION_TIME

    Assumption: num_pieces % 10 == 0 always
    """

    pass

def remaining_time_in_minutes(num_pieces, elapsed_time):
    """
    Two parameters:
    num_pieces: number of sweet pieces being prepared
    elapsed_cooking_time: the number of minutes elapsed from start till now

    Returns the number of minutes remaining till all the
    Sweets are fully done.
    """
    pass
```

**Test Cases**

| Function | Inputs | Output | Remarks |
|---|---|---|---|
| EXPECTED_COOK_TIME | | 40 | Constant declared or not? |
| cooking_time_remaining | 25 | 15 | |
| cooking_time_remaining | 0 | 40 | |
| preparation_time_required_in_minutes | 40 | 20 | |
| preparation_time_required_in_minutes | 0 | ValueError | Zero pieces not allowed |
| remaining_time_in_minutes | 20, 17 | 33 | |
| remaining_time_in_minutes | 20, 100 | 0 | No waiting if elapsed time is larger than cooking time |

CS101. A2 - Grains on a Chessboard

Calculate the number of grains of wheat on a chessboard given that the number on each square doubles. There are 64 squares on a chessboard (where square 1 has one grain, square 2 has two grains, and so on).

Write 2 functions.

1. how many grains were on a given square
    a. def square(number)
    b. Input: the number of the square. Valid square numbers are 1 to 64
    c. Check if invalid square number is given - less than 1 or greater than 64. If so raise this ValueError: `raise ValueError("Square number out of range")`
2. the total number of grains on the chessboard
    a. def total() # sum of all the grains on the chessboard

The Test Cases table lists more sample inputs and outputs.

**Code Template (grains.py)**

```
"""
Grains on a Chess Board
"""
def square(number):
     pass

def total():
     pass
```

**Sample output**

```
>>> print(square(1))
1
>>> print(square(4))
8
```

**Test Cases**

| Function | Inputs | Output | Remarks |
|---|---|---|---|
| square | 8 | 128 | |
| square | 13 | 4096 | |
| square | 60 | 576460752303423488 | |
| square | 0 | `raise ValueError("Square number out of range")` | |
| square | 65 | `raise ValueError("Square number out of range")` | |
| total | | 18446744073709551615 | |

********