```python
# A function is a piece of code with three things:
# 1. A name;
# 2. A list of parameters;
# 3. A return value (optional) computed based on the parameters.

# The def keyword defines a function.
def absolute_value(num):
    # This defines a function named absolute_value. It takes one parameter
    # called num. Based on the name, we can surmise that it is supposed to
    # calculate the absolute value of a number.

    # Inside a def is a block like any other. We can declare variables,
    # use statements, and perform calculations.
    if num < 0:
        return -num

    # This function's purpose is to compute an answer. When our answer
    # is ready, we "return" it, sending it back to the person who wanted
    # the answer.
    return num


# Important: defining a function does not execute the code inside.
# We must "call" the function, as we do with print, input, int, etc.
number = int(input("Enter a number: "))
abs_number = absolute_value(-7)
print("The absolute value of {0} is {1}".format(number, abs_number))


# Adapt this into a main()

```

```python
import random

# Play a simple guessing game with the user
def random_password():
    return random.randrange(1, 11)

def get_guess():
    return int(input("Guess the password, from 1 to 10: "))

def main():
    password = random_password()
    number_of_tries = 1
    guess = get_guess()
    while guess != password:
        number_of_tries += 1
        guess = get_guess()

    # Give a message based on how many attempts it took
    print("It took you", number_of_tries, "guesses!")
    if number_of_tries == 1:
        print("Nice! You cheated, didn't you?")
    elif 2 <= number_of_tries <= 4:
        print("Hey, that's better than average!")
    elif number_of_tries == 5:
        print("That's the average number of guesses!")
    elif number_of_tries <= 9:
        print("You're a little unlucky.")
    elif number_of_tries == 10:
        print("You're REALLY unlucky.")
    else:
        print("You may need to rethink your guessing strategy...")

main()

```

```python
# is_factor
def is_factor(dividend, divisor):



# square
def square(num):



# get_grade
def get_grade(percent):

# is_even / is_odd
def is_even(x):


def is_odd(x):



# round_nearest
def round_nearest(x):



# gcd
def gcd(a, b):
    while b != 0:
        remainder = a % b
        a = b
        b = remainder
    return a

# max
def max(a, b):


def max3(a, b, c):




def median3(a, b, c):


```

```python
# Some good functions are good. Some are bad. We want to write good functions.

def absolute_value(value, is_negative):
    if is_negative:
        return -value
    return value

# Why is is_negative a parameter? Is it necessary?
# Good functions take only the parameters that are necessary
# to make their calculation or decision.


def power(base, exponent):
    return exponent ** base

# Good functions use give meaningful names to parameters,
# and use them in the way they are named.


def maximum(num1, num2):
    if num1 > num2:
        print("The max is", num1)
    else:
        print("The max is", num2)

# Good functions RETURN the values they calculate,
# they don't print them. Don't print in functions, except in rare
# circumstances:
#       - the function's purpose is to gather input from the user
#       - the function's purpose is to display information to the user
# Such functions will generally have names that reflect their purpose,
# like "get_gunpowder" or "print_menu".


def minimum(num1, num2):
    num1 = float(input("Please enter a positive number"))
    while num1 <= 0:
        num1 = float(input("Please enter a positive number"))

    print("OK, let's find the min of those numbers!")
    if num1 < num2:
        return num1
    return num2



```

```
1  # "Scope"
2
3  # Come back to this after looking at main()...
4  def some_function():
5      x = 20
6      print(x)
7
8  def another_function(x):
9      x = 30
10     print(x)
11
12 def main():
13     # Any time a variable is assigned a value for the very first time, that variable
14     # is created in the namespace with a scope bound to the function in which it was
15     # declared. If a variable is made outside of a function, we call it "global" --
16     # more on this later.
17
18     x = 10
19     # x did not exist before this line, so it is created. Its scope starts on the line
20     # it is declared, and terminates at the end of this main() function.
21     print(x)
22     # When we attempt to use a variable, we make sure it is still in scope. A semantic
23     # error if it is not.
24
25     if x != 10:
26         # this variable y has a scope that ends when main() ends.
27         y = 5
28     else:
29         y = 0
30     print(y)
31
32     # This can get confusing when calling other functions...
33     some_function()
34     print(x)
35     # The x in some_function is a DIFFERENT variable named x, because it was declared
36     # in a different namespace. Assigning 20 to x in some_function does *not* change the x
37     # declared here in main.
38
39     # This also applies to parameters. Even if the parameter has the same name
40     # as the argument sent to it, it is really a different variable.
41     another_function(x)
42     print(x)
43
44 main()
45
```