

---

# Python Functions

---

Python functions can take in data , and return data . So far we have been *using* many functions in python that take in information and possibly return information such as

```
# takes in a prompt, returns user response
text = input("prompt? ")

# takes in a number, returns the square root of that number
y = math.sqrt(25)

# takes in text to display to the user
print("Hello, world")
```

In this chapter we are going to learn how to *define* functions which can take in information and return information!

## Anatomy of a Function

To understand how to defined functions which can take in data and can return data, let's dissect a simple example of the definition of an average function. Average takes in two pieces of data and returns the mathematical average.

Definition: The definition of average specifies that it needs two pieces of information by writing two variable names, separated by a comma `,` between the parenthesis in the definition line. In the body of the average function it can assume that the function call provided two pieces of information inside the function body average can refer to the first piece of information as variable `a` and the second as variable `b`. The definition of average specifies that it gives information back to the caller via the line that starts with `return`.

Call: Average is called in the main function. Information is provided for the function call to average between parentheses in the call: `average(5.0, 12.0)`. The value that is returned from the function can be saved in a variable, as in this example: `mid = average(5.0, 12.0)`.

You can step through these slides (click on the slides) to see some of the terminology:

# Anatomy of a function

```
def main(): function "call"  
    mid = average(5.0, 10.2)  
    print(mid)
```

function caller



function "definition"

```
def average(a, b):  
    sum = a + b  
    return sum / 2
```

function author



To better understand this, let's look at a few examples

## No Parameters, No Return Example

```
def print_intro():  
    """  
    Functions do not need to take in information and do not need  
    to return information. Here is a simple example:  
    """  
    print("Welcome to class")  
    print("It's the best part of my day")  
  
def main():  
    print_intro()
```

## Parameter Example

```
def print_opinion(num):  
    """  
    This function requires a single piece of information!  
    Inside the definition we will refer to the information provided  
    as num.  
    """  
    if num == 5:  
        print("I love 5!")  
    else:  
        print("Whatever")  
  
def main():  
    # when calling the function, we must provide the information  
    print_opinion(5)  
  
    # which can be as a literal number or as a variable
```

```
user_num = int(input("enter a num: "))
print_opinion(user_num)
```

## Parameter and Return Example

```
def meters_to_cm(meters):
    """
    This simple function expects information in: a number of meters
    and provides information back: the corresponding number of cm.
    It uses the return keyword followed by the information to be returned.
    """
    return 100 * meters

def main():
    # when you call this function, it gives you information back
    # a common practice is to save the result in a variable
    num_cm = meters_to_cm(5.1)
    print(num_cm)

    # you can use that information immediately
    print(meters_to_cm(9.5))
```

## Contrasting case: print vs. return:

```
# how is this function
def meters_to_cm_case1(meters):
    return 100 * meters
```

```
# Different than this function?
def meters_to_cm_case2(meters)
    print(100 * meters)
```

Returning is not the same as printing. Return gives a number back to the part of the program that called your function. Print puts the value on the console.

## Multiple Return Statements

```
def max(num1, num2):
    """
    this function has multiple return statements. When the function executes,
    if it hits a return statement it immediately halts and gives back the specified
    """
    if num1 >= num2:
        # this return statement only happens if the condition in the if-statement is 1
        return num1
    return num2

def main():
    larger = max(5,1)
```

## Big Example

Here is a big example of a medium sized program which was originally written in one large main file. By using functions, we can decompose the program into smaller, separate pieces. The code "using functions" is much better style. It is easier to read the main function. The

flow of information is well controlled. It is longer, but mostly because of the extra comments, another important piece of writing programs with good style:

## Using Functions

```
def main():
    print("Welcome to the CodeInPlace G
    print("Pick a door and win a prize"
    print("-----

    door = get_door()
    prize = compute_prize(door)
    print('You win ' + str(prize) + ' t

def get_door():
    """
    Get door, asks the user to enter a
    Reprompts the user until they enter
    Returns the valid door choice as an
    """

    door = int(input("Door: "))
    # while the input is invalid
    while door < 1 or door > 3 :
        # tell the user the input was i
        print("Invalid door!")
        # ask for a new input
        door = int(input("Door: "))
    return door
```

## Without Functions

```
def main():
    print("Welcome to the CodeInPlace G
    print("Pick a door and win a prize"
    print("-----

    # Part 1: get the door number form
    door = int(input("Door: "))
    # while the input is invalid
    while door < 1 or door > 3 :
        # tell the user the input was i
        print("Invalid door!")
        # ask for a new input
        door = int(input("Door: "))

    # Part 2: compute the prize
    prize = 4
    if door == 1:
        prize = 2 + 9 // 10 * 100
    elif door == 2:
        locked = prize % 2 != 0
        if not locked:
            prize += 6
    elif door == 3 :
        for i in range(door):
```

```
def compute_prize(door):  
    """  
    Compute prize, based off the door w  
    given as a parameter. Returns the p  
    Uses many non-constant magic-number  
    make the game more fun to play in c  
    """  
    prize = 4  
    if door == 1:  
        # what happens in door 1?  
        prize = 2 + 9 // 10 * 100  
    elif door == 2:  
        # what happens in door 2?  
        locked = prize % 2 != 0  
        if not locked:  
            prize += 6  
    elif door == 3:  
        # what happens in door 3?  
        for i in range(door):  
            prize += i  
    return prize
```

```
prize += i
```

```
# Part 3: report the prize  
print('You win ' + str(prize) + ' t
```