# Functions Part 2

can code communities

# Making Functions More Versatile

We can add multiple parameters to a function.

If we want a value to be changeable by user, but have a default value, we can set the parameter equal to the default value within the function definition.

If some parameters have default values, the user can choose to input some arguments but not others. So the number of arguments might be less than the number of parameters.

The number of **arguments** will always be **less than or equal to** the number of **parameters**, never greater than.

# Default Values

If the user calls the function with an argument, that overrides the default value, so the default is ignored.

If the user calls the function without an argument, the default is selected.

What is the output of this code?

```
def my_function(country = "Norway"):
  print("I am from " + country)

my_function("Sweden")
my_function("India")
my_function()
my_function("Brazil")
```

(Example from w3schools)

# Default Values

What is the output of this code?

```
def power(num, exp_num=1, exp_den=1):
    num **= (exp_num / exp_den)
    return num

pow1 = power(2, 2)
pow2 = power(2, 2, 2)
pow3 = power(2, exp_den=2)

print(pow1, pow2, pow3)
```

**Positional argument passing**

**Keyword argument passing** allows you to skip a parameter. The skipped parameter gets assigned the default value.

What are other ways you could call the function to get the same output?

# Exercise

Write a function called `center` that returns either the mean or median of a list of numbers.

This function should take two parameters: A list of numbers, and an optional parameter called `use_median` which should default to `False`.

If `use_median` is `False`, return the mean of the list.

If `use_median` is `True`, return the median of the list.

Test your function by calling it with different arguments.

# Returning Multiple Values

Sometimes it is useful to return multiple values from a function.

A convenient way to do this is by returning multiple values separated by commas, and you can set each return value to a variable when you call the function, similar to unpacking a tuple.

Make sure the number of return values lines up, otherwise you will get an error.

```
def get_vowels_and_consonants(word):
    vowels = ""
    consonants = ""
    for letter in word:
        if letter in "aeiou":
            vowels += letter
        elif letter in "bcdfghjklmnpqrstvwxyz":
            consonants += letter
    return vowels, consonants

vowels, consonants = get_vowels_and_consonants("apple")
print(vowels)
print(consonants)
```

What is the output of this code?

# Exercise

Write a Python function called `get_stats` that takes in a list of numbers and returns the following three values: The mean, the median, and the mode of the list.

Call the function on a list, and print each statistic on a separate line.

```
my_list = [1,2,4,5,5]
```

Output:

```
Mean: 3.4
Median: 4
Mode: 5
```

# Global Variables

If you create a variable in a function, it's only available while the function is running. Once the function ends, it goes out of scope.

If you create a variable outside a function, that variable is **global**, which means it's always in scope.

If you want to create a global variable in a function, you must use the global keyword. You can also do this to modify existing global variables within a function.

# Exercise

What is the output of this code?

```
x = "challenging"

def change_x():
    x = "fun"

change_x()

print("Programming is", x)
```

What about this code?

```
x = "challenging"

def change_x():
    global x
    x = "fun"

change_x()

print("Programming is", x)
```

Is the output the same or different? Why?

# Documenting a Function

# Docstring

On the first line of your function, after the header, you can write a comment called a doc string.

The docstring should contain:

- A description of the function
- Each parameter and its purpose
- The return value of the function

```
def area_of_rectangle(height, width):
    '''Return the area of a rectangle.

    - height: the height of the rectangle.
    - width: the width of a rectangle.
    '''
    return height * width
```

Docstring PEP: https://peps.python.org/pep-0257/

# Type Hinting

Many programming languages are statically typed, so you must declare the type of each variable.

Python is dynamically typed, so it figures out the type of each variable automatically.

Sometimes when you're writing functions, you want to specify the type of each variable. In Python, this is **optional**, and called type hinting.

```
def area_of_rectangle(height: float, width: float) -> float:
    '''Return the area of a rectangle.

    - height: the height of the rectangle.
    - width: the width of a rectangle.
    '''
    return height * width
```

# Exercise

Go back to one function that you wrote earlier in this class and add some documentation to it.

Add type hinting and a docstring.

# Lambdas: Anonymous Functions

# Lambda

A lambda is a small anonymous function. It can take any number of arguments, but it can only have one expression, which is returned.

Syntax: `lambda arguments : expression`

A lambda is a way to define a function in one line.

**Example:**

```
add_2 = lambda a : a + 2
print(add_2(2))
```

What is the output when you run the above two lines of code?

# Lambdas vs Functions

This lambda:

```
name = lambda arg : expr
```

Is equivalent to this function:

```
def name(arg):
        return expr
```

For example, these two code snippets are equivalent:

1.
```
add_nums = lambda a, b : a + b
```

2.
```
def add_nums(a, b):
    return a + b
```

# Why use Lambdas?

You can put lambdas inside functions to make a more versatile function.

**Exercise:** What is the output of this code?

```
def multiplier(n):
        return lambda num : num * n

doubler = multiplier(2)
tripler = multiplier(3)

print(doubler(5))
print(tripler(5))
```

# Exercise

Write a lambda that computes the n-th power of a number, given two arguments, `num` and `n`.

Now, write a function that is equivalent to the lambda.

# More on Lambdas

Lambdas are derived from Lambda calculus, and are a core part of functional programming languages such as Lisp, Scheme, and Haskell.

Python, along with most programming languages, is considered an imperative language. A few functional programming topics (such as lambdas) have been added to Python, and they are more bonus features instead of a core part of the language.

- Some other topics include **filter**, **map**, and **reduce**.

Lambdas can be used alongside filter, map, and reduce, and also as a key with the sorted() function in Python.

# Example: Lambda and Map

Here's a more complex use of lambdas in Python.

The map() function in Python takes a function and applies it to every item in a list. You can use map() with a lambda:

```
L = list(map(lambda x: x.upper(), ['cat', 'dog', 'cow']))
print(L)
```

What do you think the output of this code is?

How does this compare to using list comprehensions?

# Example: Sorting with Lambda

You can use a lambda as the `key` parameter to the `sorted()` function to customize how an iterable is sorted.

Let's say we have a list of dictionaries that represent students. Each student has a name and a grade from 0 to 100.

Use the sorted function to sort the students by name:

```
students_by_name = sorted(students, key = lambda s: s['name'])
```

Now, use the sorted function to sort the students by grade:

```
students_by_grade = sorted(students, key = lambda s: s['grade'])
```

# Resources

Global variables: https://www.w3schools.com/python/gloss_python_global_variables.asp

Docstring conventions: https://peps.python.org/pep-0257/

Type hinting in Python (for function documentation): https://docs.python.org/3/library/typing.html

Lambda basics: https://www.w3schools.com/python/python_lambda.asp

Longer article on lambdas: https://realpython.com/python-lambda/