Functions

CS for Social Good

Naming Conventions

Two main types of naming systems:

- Camel case: `camelCase`
- Snake case: `snake_case`
- Snake case is the standard for Python!

Naming Conventions

Two main types of naming systems:

- Camel case: `camelCase`
- Snake case: `snake_case`
- Snake case is the standard for Python!

- Names must start with a letter or the underscore character (`letter` and `_letter` are both valid)
    - Cannot start with a number (`1st_variable` is not allowed)
- Names can only include alphanumeric characters and underscores
    - No special characters
- Names are case-sensitive (`case` and `Case` are different variables)

Naming Conventions

Practice: are the following variable names allowed?

1. _my_var = “B”
2. my-var = “e”
3. myvar = “n”
4. my var = “i”
5. myVar = “c”
6. myvar2 = “i”
7. 2myvar = “a”
8. my_var = “High”
9. MYVAR = “School”

Naming Conventions

Practice: are the following variable names allowed?

1.	_my_var = “B”	Yes
2.	my-var = “e”	No (no special characters allowed)
3.	myvar = “n”	Yes
4.	my var = “i”	No (no spaces allowed)
5.	myVar = “c”	Yes
6.	myvar2 = “i”	Yes
7.	2myvar = “a”	No (cannot start with a number)
8.	my_var = “High”	Yes
9.	MYVAR = “School”	Yes

Coding Break

Basic Functions

A function is a block of code that runs only when it is called.
Below is the basic template for creating functions in Python.

```
def hello_world( ):
        print(“Hello, world!”)
```

All functions definitions need three things:

- The def keyword
- A name for the function followed by a pair of parentheses and a colon
- A indented block of code containing the steps to be executed

Basic Functions

Functions are a great way to decompose large blocks of code into chunks of closely related steps.

This will make you programs more readable and allow you to reuse your code in different areas.

If you find yourself typing the same lines of code over and over, that is a good indicator that you should use put these lines together into a function.

Basic Functions

```
def greet_user( ):
    name = input(“Enter your name:”)
    print(“Hello” + name)


def even_or_odd( ):
    number = int(input(“Enter a number:”))
    if number % 2 == 0:
        print(“This number is even.”)
    else:
        print(“This number is odd.”)
```

Basic Functions

You call a function in python by typing the function name followed by parentheses.

```
greet_user( )
choice = input(“cards or dice?”)
if choice == “cards”:
        card_game( )
else:
        dice_game( )
print_goodbye( )
```

Basic Functions

You call a function in python by typing the function name followed by parentheses.

```
greet_user( )
choice = input(“cards or dice?”)
if choice == “cards”:
    card_game( )
else:
    dice_game( )
print_goodbye( )
```

Basic Functions

You can call functions within other functions

```
def daily_report( ):
    print_date( )
    print_weather( )
    print_headlines( )
```

Basic Functions

Functions are super helpful with repetitive tasks!
What are some benefits of print_fancy_separator( )?

```
def print_fancy_separator( ):
        print(“°º¤ø,¸¸,ø¤º°`°º¤ø,¸,ø¤°º¤ø,¸¸,ø¤º°`°º¤ø,¸”)

def daily_report( ):
        print_fancy_separator( )
        print_date( )
        print_weather( )
        print_fancy_separator( )
        print_headlines( )
        print_fancy_separator( )
```

Arguments

Now what if you are trying to do something that is repetitive but has some differences between repetitions?

E.g. your mom is making you write thank you cards to a dozen of different relatives

Dear aunt Betty, Thank you for the sweater. I really love it! Best wishes, John
Dear uncle Tom, Thank you for the book. I really love it! Best wishes, John
Dear aunt Mary, Thank you for the scarf. I really love it! Best wishes, John
...

Arguments

Arguments to the rescue!

Arguments allow you to give the function additional information on what they should do.

(In fancier words, you **pass** arguments to a function)

Arguments

When you define functions, you can put arguments into the parentheses, then pass in the value when you call the function.

```
def greet_user( ):
        name = input(“What is your name?”)
        print(“Hello “ + name)
```

Arguments

When you define functions, you can put arguments into the parentheses, then pass in the value when you call the function.

```
def greet_user_specified_by_me(name):
    print(“Hello “ + name)
```

Arguments

When you define functions, you can put arguments into the parentheses, then pass in the value when you call the function.

```
def greet_user_specified_by_me(name):
    print(“Hello “ + name)


                    greet_user_specified_b
y_me(“Alice”)
```

Output: Hello Alice

Arguments

When you define functions, you can put arguments into the parentheses, then pass in the value when you call the function.

```
def greet_user_specified_by_me(name):
        print(“Hello “ + name)


greet_user_specified_by_me(“Bob”)
```

Output: Hello Bob

Coding Break

Arguments

Note that the content of an argument inside the function is only based on what’s passed to it. Functions can’t see what’s outside of their **scope**

```
def print_var(var):
        print(var)

var = “Hi”
a = “Hello”
print_var(a)         Output: Hello
print(var)           Output: Hi
```

Coding Break

Arguments

Tracing problem:

```
def add(y, x):
   z = x + y
   print("in function: ", x, y, z)


x = 4
y = 8
z = y - 1
add(x, z)
print("outside of function: ", x, y, z)
```

Arguments

Tracing problem:

```
def add(y, x):
   z = x + y
   print("in function: ", x, y, z)


⇨ x = 4
  y = 8
  z = y - 1
  add(x, z)
  print("outside of function: ", x, y, z)
```

x = 4

Arguments

Tracing problem:

```
def add(y, x):
   z = x + y
   print("in function: ", x, y, z)


x = 4
y = 8
z = y - 1
add(x, z)
print("outside of function: ", x, y, z)
```

x = 4

y = 8

Arguments

Tracing problem:

```
def add(y, x):
    z = x + y
    print("in function: ", x, y, z)


x = 4
y = 8
z = y - 1
add(x, z)
print("outside of function: ", x, y, z)
```

x = 4

y = 8

z = 7

Arguments

Tracing problem:

```
def add(y, x):
   z = x + y
   print("in function: ", x, y, z)


x = 4
y = 8
z = y - 1
⇨ add(x, z)
print("outside of function: ", x, y, z)
```

x = 4

y = 8

z = 7

Arguments

Tracing problem:

```
⇨ def add(y, x):
     z = x + y
     print("in function: ", x, y, z)


  x = 4
  y = 8
  z = y - 1
  add(x, z)
  print("outside of function: ", x, y, z)
```

```
x = 4
y = 8
z = 7
y = ?
x = ?
```

Arguments

Tracing problem:

```
def add(y, x):
   z = x + y
   print("in function: ", x, y, z)


x = 4
y = 8
z = y - 1
add(x, z)
print("outside of function: ", x, y, z)
```

x = 4

y = 8

z = 7

y = ?

x = ?

Arguments

Tracing problem:

```
⇨ def add(y, x):
     z = x + y
     print("in function: ", x, y, z)


  x = 4
  y = 8
  z = y - 1
  add(x, z)
  print("outside of function: ", x, y, z)
```

x = 4

y = 8

z = 7

y = 4

x = ?

Arguments

Tracing problem:

```
def add(y, x):
   z = x + y
   print("in function: ", x, y, z)


x = 4
y = 8
z = y - 1
add(x, z)
print("outside of function: ", x, y, z)
```

x = 4

y = 8

z = 7

y = 4

x = ?

Arguments

Tracing problem:

```
⇨ def add(y, x):
      z = x + y
      print("in function: ", x, y, z)


  x = 4
  y = 8
  z = y - 1
  add(x, z)
  print("outside of function: ", x, y, z)
```

x = 4

y = 8

z = 7

y = 4

x = 7

Arguments

Tracing problem:

```
def add(y, x):
    z = x + y
    print("in function: ", x, y, z)


x = 4
y = 8
z = y - 1
add(x, z)
print("outside of function: ", x, y, z)
```

x = 4

y = 8

z = 7

y = 4

x = 7

z = 11

Arguments

Tracing problem:

```
   def add(y, x):
     z = x + y
⇨    print("in function: ", x, y, z)


   x = 4
   y = 8
   z = y - 1
   add(x, z)
   print("outside of function: ", x, y, z)
```

```
x = 4
y = 8
z = 7
y = 4
x = 7
z = 11
```

```
in function: 7, 4, 11
```

Arguments

Tracing problem:

```
def add(y, x):
   z = x + y
   print("in function: ", x, y, z)


x = 4
y = 8
z = y - 1
add(x, z)
⇨ print("outside of function: ", x, y, z)
```

x = 4

y = 8

z = 7

y = 4

x = 7

z = 11

```
in function: 7, 4, 11
outside of function: 4, 8, 7
```

Returns

When a function ends, you have the option to have it **return** a value!

So far, we've been working with functions that don't return anything (these are called void functions). Let's look at a function that returns the sum of two numbers:

Returns

When a function ends, you have the option to have it **return** a value!

So far, we've been working with functions that don't return anything (these are called void functions). Let's look at a function that returns the sum of two numbers:

```
def return_sum(a, b):
        sum = a + b
        return sum

sum = return_sum(3, 6)
print(sum)
```

Behind the scenes:

Returns

When a function ends, you have the option to have it **return** a value!

So far, we've been working with functions that don't return anything (these are called void functions). Let's look at a function that returns the sum of two numbers:

```
def return_sum(a, b):
        sum = a + b
        return sum

sum = return_sum(3, 6)
print(sum)
```

Behind the scenes:

```
sum = return_sum(3, 6)
```

Returns

When a function ends, you have the option to have it **return** a value!

So far, we've been working with functions that don't return anything (these are called void functions). Let's look at a function that returns the sum of two numbers:

```
def return_sum(a, b):
    sum = a + b
    return sum

sum = return_sum(3, 6)
print(sum)
```

Behind the scenes:

```
def return_sum(a, b):
    sum = a + b
    return sum
```

Returns

When a function ends, you have the option to have it **return** a value!

So far, we've been working with functions that don't return anything (these are called void functions). Let's look at a function that returns the sum of two numbers:

```
def return_sum(a, b):
        sum = a + b
        return sum

sum = return_sum(3, 6)
print(sum)
```

Behind the scenes:

```
def return_sum(3, 6):
        sum = a + b
        return sum
```

Returns

When a function ends, you have the option to have it **return** a value!

So far, we've been working with functions that don't return anything (these are called void functions). Let's look at a function that returns the sum of two numbers:

```
def return_sum(a, b):
    sum = a + b
    return sum

sum = return_sum(3, 6)
print(sum)
```

Behind the scenes:

```
def return_sum(3, 6):
    sum = 3 + 6
    return sum
```

Returns

When a function ends, you have the option to have it **return** a value!

So far, we've been working with functions that don't return anything (these are called void functions). Let's look at a function that returns the sum of two numbers:

```
def return_sum(a, b):
        sum = a + b
  ➡     return sum

sum = return_sum(3, 6)
print(sum)
```

Behind the scenes:

```
def return_sum(3, 6):
        sum = 3 + 6
        return 9
```

Returns

When a function ends, you have the option to have it **return** a value!

So far, we've been working with functions that don't return anything (these are called void functions). Let's look at a function that returns the sum of two numbers:

```
def return_sum(a, b):
        sum = a + b
        return sum

⇨ sum = return_sum(3, 6)
print(sum)
```

Behind the scenes:

```
sum = return_sum(3, 6)
```

Returns

When a function ends, you have the option to have it **return** a value!

So far, we've been working with functions that don't return anything (these are called void functions). Let's look at a function that returns the sum of two numbers:

```
  def return_sum(a, b):
        sum = a + b
        return sum

⇨ sum = return_sum(3, 6)
  print(sum)
```

Behind the scenes:

```
sum = 9
```

Returns

When a function ends, you have the option to have it **return** a value!

So far, we've been working with functions that don't return anything (these are called void functions). Let's look at a function that returns the sum of two numbers:

```
def return_sum(a, b):
        sum = a + b
        return sum

sum = return_sum(3, 6)
print(sum)
```

Behind the scenes:

```
print(9)
```

Returns

When a function ends, you have the option to have it **return** a value!

So far, we've been working with functions that don't return anything (these are called void functions). Let's look at a function that returns the sum of two numbers:

```
def return_sum(a, b):
        sum = a + b
        return sum

sum = return_sum(3, 6)
print(sum)
```

Output:

```
9
```

Coding Break

Next Time!

For-loops and lists!