Jan Kahánek - Unsplash

Writing functions

Quick recap

What is a function?

Quick recap

What is a function?

A reusable chunk of code that does something specific

Quick recap

What is a function?

A reusable chunk of code that does something specific

How do we use an existing function? What verb do we use?

Quick recap

What is a function?

A reusable chunk of code that does something specific

How do we use an existing function? What verb do we use?

We <u>call</u> functions, syntax looks like this `function( )`

Quick recap

What is a function?

A reusable chunk of code that does something specific

How do we use an existing function? What verb do we use?

We <u>call</u> functions, syntax looks like this `function( )`

How do we pass information to a function? How do we get info back?

Quick recap

What is a function?

A reusable chunk of code that does something specific

How do we use an existing function? What verb do we use?

We call functions, syntax looks like this `function( )`

How do we pass information to a function? How do we get info back?

We pass information via arguments: `function(x, 12)`

Quick recap

What is a function?

A reusable chunk of code that does something specific

How do we use an existing function? What verb do we use?

We <u>call</u> functions, syntax looks like this `function( )`

How do we pass information to a function? How do we get info back?

We pass information via arguments: `function(x, 12)`

Functions can return data, e.g., `len(s)` returns the length of `s`

Quick recap

What is a function?

- A reusable chunk of code that does something specific

How do we use an existing function? What verb do we use?

- We call functions, syntax looks like this `function( )`

How do we pass information to a function? How do we get info back?

- We pass information via arguments: `function(x, 12)`
- Functions can return data, e.g., `len(s)` returns the length of `s`
  - Do all functions return data?

Quick recap

What is a function?

A reusable chunk of code that does something specific

How do we use an existing function? What verb do we use?

We <u>call</u> functions, syntax looks like this `function( )`

How do we pass information to a function? How do we get info back?

We pass information via arguments: `function(x, 12)`

Functions can return data, e.g., `len(s)` returns the length of `s`

Do all functions return data? No!

Examples of functions we’ve discussed?

Examples of functions we’ve discussed?

`print()`, `input()`, `int()`, `str()`, `float()`, `len()`, `min()`, `max()`, `abs()`, etc!

Accessing even more functions

How do we go beyond built-in functions?

Accessing even more functions

How do we go beyond built-in functions?

Work with other modules!

Accessing even more functions

How do we go beyond built-in functions?

Work with other modules!

What do you think this does?

```
import random
num = random.randint(0, 10)
print(num)
```

Accessing even more functions

How do we go beyond built-in functions?

Work with other modules!

What do you think this does? (docs)

```
import random
num = random.randint(0, 10)
print(num)
```

Writing our own functions

So far we’ve been *calling* existing functions

Writing our own functions

So far we’ve been *calling* existing functions

Now we’ll be *defining* our own functions!

Writing our own functions

Why?

- Often, we have things unique to us that we want to do repeatedly
- Functions *should* improve code readability
- Enable modularity in your code
- Eliminates redundancy

Parts of a function

Based on what we know, what are the main pieces of a function?

Parts of a function

Based on what we know, what are the main pieces of a function?

(Think about the parts of a variable: name, type, and value)

Parts of a function

Based on what we know, what are the main pieces of a function?
(Think about the parts of a variable: name, type, and value)

My picks:

- Name
- Information passed in as arguments (types?)
- What the function *does* (main code)
- Returned information, if any (type?)

Anatomy of a function in Python

```
def func_name(arg1, arg2):
```

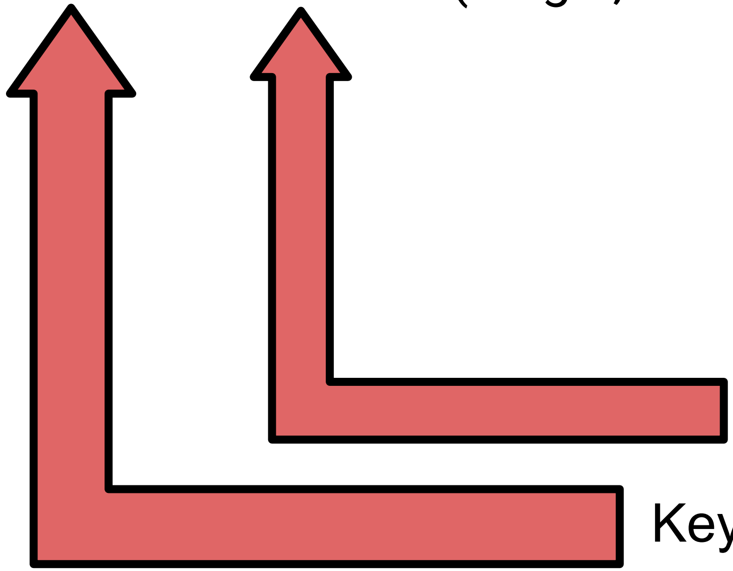
Anatomy of a function in Python

```
def func_name(arg1, arg2):
```

Keyword to tell Python we are defining a function

Anatomy of a function in Python

```
def func_name(arg1, arg2):
```

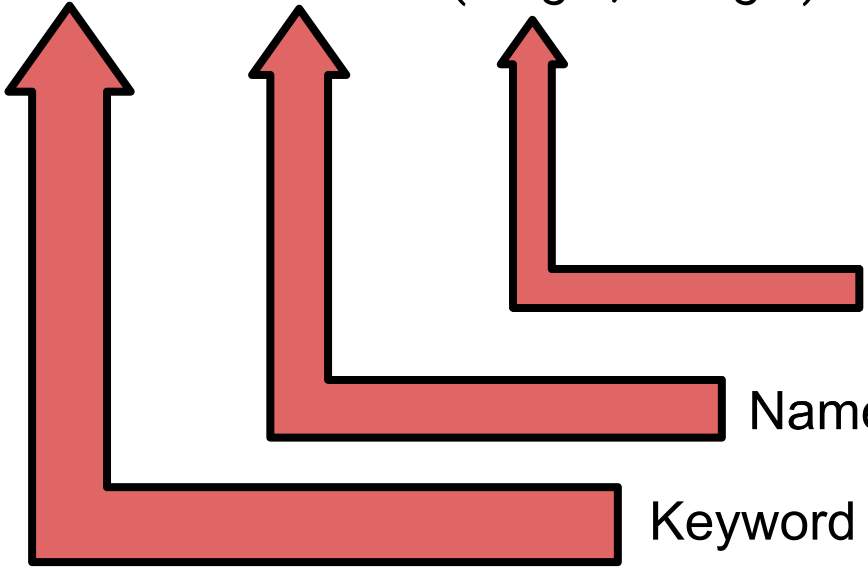


Name of the function (still snake_case)

Keyword to tell Python we are defining a function

Anatomy of a function in Python

```
def func_name(arg1, arg2):
```



List of arguments that user should pass in (parameters)

Name of the function ([still snake_case]())

Keyword to tell Python we are defining a function

Anatomy of a function in Python

```
def func_name(arg1, arg2):
```

Anatomy of a function in Python

```
def func_name(arg1, arg2):
   x = arg1 * 2
   print(x)
   # Can be many lines…
```

Anatomy of a function in Python

```
def func_name(arg1, arg2):
    x = arg1 * 2
    print(x)
    # Can be many lines…
```

Normal lines of code make up the **body** of the function

Anatomy of a function in Python

```
def func_name(arg1, arg2):
    x = arg1 * 2
    print(x)
    # Can be many lines…
```

Normal lines of code make up the **body** of the function

Exact lines will vary wildly depending on the function!

Anatomy of a function in Python

```
def func_name(arg1, arg2):
    x = arg1 * 2
    print(x)
    # Can be many lines…
```

Normal lines of code make up the **body** of the function

Exact lines will vary wildly depending on the function!

These lines are indented. Why?

Anatomy of a function in Python

```
def func_name(arg1, arg2):
    x = arg1 * 2
    print(x)
    # Can be many lines…
```

Normal lines of code make up the **<u>body</u>** of the function

Exact lines will vary wildly depending on the function!

These lines are <u>indented</u>. Why?

We need to specify which lines of code are in the body of the function.

Anatomy of a function in Python

```
def func_name(arg1, arg2):
    x = arg1 * 2
    print(x)
    # Can be many lines…
```

Normal lines of code make up the **body** of the function

Exact lines will vary wildly depending on the function!

These lines are indented. Why?
We need to specify which lines of code are in the body of the function.
Some languages use symbols, Python uses indentation!

Anatomy of a function in Python

```
def func_name(arg1, arg2):
    x = arg1 * 2
    print(x)
    # Can be many lines…
```

$\}$

Normal lines of code make up the **body** of the function

Exact lines will vary wildly depending on the function!

Anatomy of a function in Python

```
def func_name(arg1, arg2):
    x = arg1 * 2
    print(x)
    # Can be many lines…
    return x + 1
```

Normal lines of code make up the **body** of the function

Exact lines will vary wildly depending on the function!

Anatomy of a function in Python

```
def func_name(arg1, arg2):
    x = arg1 * 2
    print(x)
    # Can be many lines…
    return x + 1
```

Normal lines of code make up the **body** of the function

Exact lines will vary wildly depending on the function!

If our function returns something, use a return statement

Anatomy of a function in Python

```
def func_name(arg1, arg2):
    x = arg1 * 2
    print(x)
    # Can be many lines…
    return x + 1
```

Normal lines of code make up the **body** of the function

Exact lines will vary wildly depending on the function!

If our function returns something, use a return statement
Still indented, no parentheses necessary!

Calling user-defined functions

```
def greet(name):
    print(“Hello there, {name}!”)
```

Calling user-defined functions

```
def greet(name):
   print(“Hello there, {name}!”)


greet(“Louie”)
```

Calling user-defined functions

```
def greet(name):
   print(“Hello there, {name}!”)


greet(“Louie”)
# Note: stop indentation when function body stops!
```

Calling user-defined functions

```
import math

def circle_circumference(radius):
    return math.pi * radius * 2
```

Calling user-defined functions

```
import math

def circle_circumference(radius):
   return math.pi * radius * 2



circle_rad = 3
line_length = circle_circumference(circle_rad)
```

Living coding example :^)

More on return statements

What does this code return?

```
def mystery_func(x):
    return x
    y = x**2 + 1
    return y
```

More on return statements

What does this code return?

```
def mystery_func(x):
    return x
    y = x**2 + 1
    return y
```

It will always return x!

Return statements immediately stop the function

pass

Call a function that does not exist? Error!

pass

Call a function that does not exist? Error!

Sometimes we want to “stub” functions

- The function exists
- But does not have a *meaningful* body

pass

Call a function that does not exist? Error!

Sometimes we want to “stub” functions

- The function exists
- But does not have a *meaningful* body

Use pass!

pass

Call a function that does not exist? Error!

Sometimes we want to “stub” functions

- The function exists
- But does not have a *meaningful* body

Use pass!

```
def check_balance(account_num):
    pass
```

Function scoping

What will the following code output?

```
def example_func(x):
   y = x+1
   return y**2
example_func(3)
print(y)
```

Function scoping

What will the following code output?

```
def example_func(x):
   y = x+1
   return y**2
example_func(3)
print(y)
```

Error! y only exists in `example_func`

Function scoping

What will the following code output?

```
def example_func(x):
    y = x+1
    return y**2
example_func(3)
print(y)
```

Error! y only exists in `example_func`

It is <u>local</u> to that function

Function scoping 2

What will the following code output?

```
def example_func():
    x=2
    print(x)
x = 1
print(x)
example_func()
print(x)
```

Function scoping 2

What will the following code output?

```
def example_func():
    x=2
    print(x)
x = 1
print(x) # 1
example_func()
print(x)
```

Function scoping 2

What will the following code output?

```
def example_func():
    x=2
    print(x) # 2
x = 1
print(x) # 1
example_func()
print(x)
```

Function scoping 2

What will the following code output?

```
def example_func():
    x=2
    print(x) # 2
x = 1
print(x) # 1
example_func()
print(x) # 1
```

Function scoping 2

What will the following code output?

```
def example_func():
    x=2
    print(x) # 2
x = 1
print(x) # 1
example_func()
print(x) # 1
```

By default, a new local variable is created and takes precedence

Function scoping 2

What will the following code output?

```
def example_func():
    x=2
    print(x) # 2
x = 1
print(x) # 1
example_func()
print(x) # 1
```

Local x “falls out of scope” and disappears after end of function

By default, a new local variable is created and takes precedence

Function scoping 3

What will the following code output?

```
x = 1
print(x)
def example_func():
   x = x + 1
   print(x)
example_func()
print(x)
```

Function scoping 3

What will the following code output?

```
x = 1
print(x)
def example_func():
    x = x + 1
    print(x)
example_func()
print(x)
```

Error!

Function is trying to increment a local x, which doesn’t exist!

Function scoping 3 - global

What will the following code output?

```
x = 1
print(x)
def example_func():
    global x
    x = x + 1
    print(x)
example_func()
print(x)
```

Function scoping 3 - global

What will the following code output?

```
x = 1
print(x)
def example_func():
    global x
    x = x + 1
    print(x)
example_func()
print(x)
```

Can use “global” to tell function to use the global variable

Function scoping 3 - global

What will the following code output?

```
x = 1
print(x)
def example_func():
    global x
    x = x + 1
    print(x)
example_func()
print(x)
```

Can use “global” to tell function to use the global variable

Be careful! This gets messy FAST

Function scoping 3 - global

What will the following code output?

```
x = 1
print(x)
def example_func():
    global x
    x = x + 1
    print(x)
example_func()
print(x)
```

Can use “global” to tell function to use the global variable

Be careful! This gets messy FAST

Function scoping 3 - global

What will the following code output?

```
x = 1
print(x) # 1
def example_func():
    global x
    x = x + 1
    print(x)
example_func()
print(x)
```

Can use “global” to tell function to use the global variable

Be careful! This gets messy FAST

Function scoping 3 - global

What will the following code output?

```
x = 1
print(x) # 1
def example_func():
    global x
    x = x + 1
    print(x) # 2
example_func()
print(x)
```

Can use “global” to tell function to use the global variable

Be careful! This gets messy FAST

Function scoping 3 - global

What will the following code output?

```
x = 1
print(x) # 1
def example_func():
    global x
    x = x + 1
    print(x) # 2
example_func()
print(x)
```

Can use “global” to tell function to use the global variable

Be careful! This gets messy FAST

Function scoping 3 - global

What will the following code output?

```
x = 1
print(x) # 1
def example_func():
    global x
    x = x + 1
    print(x) # 2
example_func()
print(x) # 2
```

Can use “global” to tell function to use the global variable

Be careful! This gets messy FAST

Function scoping 3 - global

What will the following code output?

```
x = 1
print(x) # 1
def example_func():
    global x
    x = x + 1
    print(x) # 2
example_func()
print(x) # 2
```

Can use “global” to tell function to use the global variable

Be careful! This gets messy FAST

General rule: avoid globals as much as possible, use them sparingly

Function scoping 3 - global

What will the following code output?

```
x = 1
print(x) # 1
def example_func():
    global x
    x = x + 1
    print(x) # 2
example_func()
print(x) # 2
```

Can use “global” to tell function to use the global variable

Be careful! This gets messy FAST

General rule: avoid globals as much as possible, use them sparingly

Link to more info

Function scoping 4

What will the following code output?

```
example_func()
def example_func():
   print(‘hit!’)
```

Function scoping 4

What will the following code output?

```
example_func()
def example_func():
   print(‘hit!’)
```

Error!

Cannot call a function before it’s defined!

Optional arguments

We can give arguments a default value:

Optional arguments

We can give arguments a default value:

```
def create_account(id, name, nick_name='', balance=0):
    # do stuff
```

Optional arguments

We can give arguments a default value:

Required (positional) args

```
def create_account(id, name, nick_name='', balance=0):
    # do stuff
```

Optional arguments

We can give arguments a default value:

Required (positional) args

```
def create_account(id, name, nick_name='', balance=0):
    # do stuff
```

Optional args

Optional arguments

We can give arguments a default value:

```
def create_account(id, name, nick_name='', balance=0):
    # do stuff
```

```
These are valid calls:
```

Optional arguments

We can give arguments a default value:

```
def create_account(id, name, nick_name='', balance=0):
   # do stuff

These are valid calls:
create_account(0, “Louie”, “Lou”, 100)
```

Optional arguments

We can give arguments a default value:

```
def create_account(id, name, nick_name='', balance=0):
   # do stuff

These are valid calls:
create_account(0, “Louie”, “Lou”, 100)
create_account(0, “Louie”) #nick_name is blank, 0 bal
```

Optional arguments

We can give arguments a default value:

```
def create_account(id, name, nick_name='', balance=0):
   # do stuff

These are valid calls:
create_account(0, “Louie”, “Lou”, 100)
create_account(0, “Louie”) #nick_name is blank, 0 bal
create_account(0, “Louie”, “Lou”) # 0 balance
```

Optional arguments

We can give arguments a default value:

```
def create_account(id, name, nick_name='', balance=0):
    # do stuff
```

```
These are invalid calls:
```

```
create_account()
create_account(0)
```

Optional arguments

We can give arguments a default value:

```
def create_account(id, name, nick_name='', balance=0):
    # do stuff
```

```
These are invalid calls:
```

```
create_stuff()
create_stuff(0)
```

All positional args are *required*

Keyword arguments

```
def create_account(id, name, nick_name='', balance=0):
    # do stuff
```

Keyword arguments

```
def create_account(id, name, nick_name='', balance=0):
    # do stuff


create_account(0, “Louie”, balance=100)
```

Keyword arguments

```
def create_account(id, name, nick_name='', balance=0):
    # do stuff


create_account(0, “Louie”, balance=100)
```

This is valid! We skip nick_name, but it has a default

Keyword arguments

```
def create_account(id, name, nick_name='', balance=0):
    # do stuff
```

```
create_account(0, “Louie”, balance=100)
```

This is valid! We skip nick_name, but it has a default

Keyword arguments MUST come after ALL positional arguments

Keyword arguments

```
def create_account(id, name, nick_name='', balance=0):
    # do stuff


create_account(0, “Louie”, balance=100)
```

This is valid! We skip nick_name, but it has a default

Keyword arguments MUST come after ALL positional arguments

Some functions have <u>many</u> arguments, this helps keep code short

Returning multiple values

We’ll talk about more this later, but we *can* return more than one value

Returning multiple values

We'll talk about more this later, but we *can* return more than one value

```
def true_sqrt(x):
   pos_sqrt = math.sqrt(x)
   neg_sqrt = -1 * pos_sqrt
   return pos_sqrt, neg_sqrt
```

Returning multiple values

We’ll talk about more this later, but we *can* return more than one value

```
def true_sqrt(x):
    pos_sqrt = math.sqrt(x)
    neg_sqrt = -1 * pos_sqrt
    return pos_sqrt, neg_sqrt

pos, neg = true_sqrt(9)
```