Calling a function

```
def sum(num1: int, num2: int) -> int:

def main() -> None:
```

- Look at the function signature!
- General formula: function_name(<arguments>)

```
sum(num1 = 11, num2 = 3)

main()
```

How can I tell if something is a variable vs a function?

- main() vs main

Positional vs Keyword Arguments

```
def sum(num1: int, num2: int) -> int:
```

Keyword:

```
sum(num1 = 11, num2 = 3)
```

```
sum(num2 = 3, num1 = 11)
```

- Keyword being the name of the parameter
- Order of arguments doesn’t matter, since we’re explicitly assigning arguments to parameters

Positional:

```
sum(11, 3)
```

- Order DOES matter
- 11 is assigned the first parameter of sum (num1) and 3 is assigned the second argument (num2)

Return values of functions

sum(num1 = 11, num2 = 3) *will return 14*

- Need to do something with the value that is returned
- Depending on what you want to do with the returned value
  - Print the value?
  - Store it in some variable?
  - Compare it with something else?

emojified(guess, secret) *returns a string of emojis*

- EX03 - needed to print the string of emojis
- Two options:
  - Store the returned value in a variable, then print the variable
  - Directly print what is returned from the function call

Depends on how the function works and what you want to do with the function call! Think about what the function does, what is returned, and if/how you need to interact with the return value

Testing your code

- REPL
  - Type ‘python’ in your terminal to enter the REPL
  - If you see ‘>>>’ before your cursor, you are in the REPL!
  - Use to test specific functions
- Trailhead
  - A prettier representation of your REPL/terminal
  - Run tab - run the whole file
  - Interact - interact with specific functions without needing to import
- Running the file
  - ‘python -m path.to.file’
  - Should happen *outside* the REPL
- Now - pytest
  - ‘python -m pytest path/to/file.py’
  - Should happen *outside* the REPL
  - Or with testing tab (beaker)