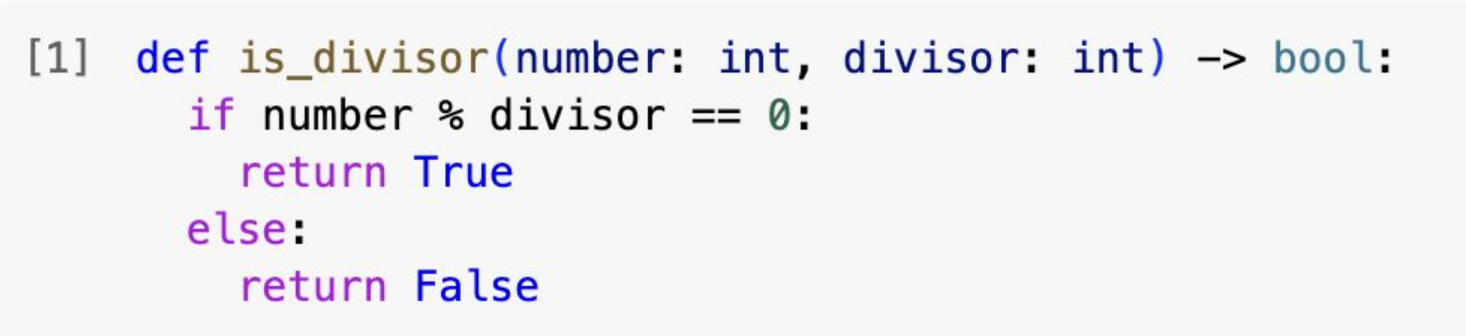
Functions continued: keywords, higher order, lambdas

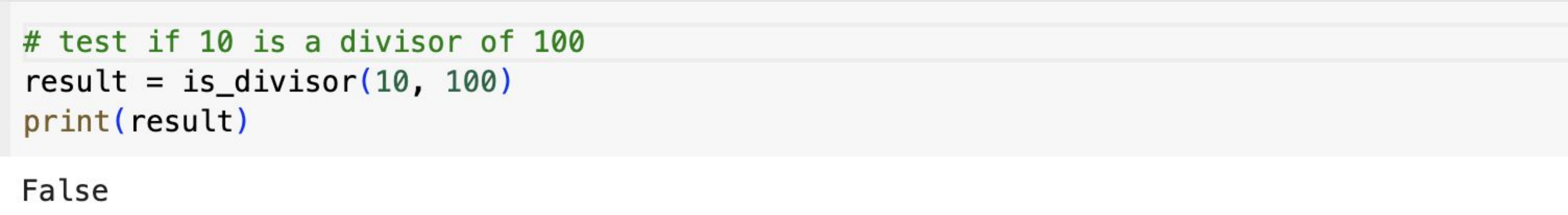
Function arguments

formal parameters are used in the function signature

actual parameters are passed in during **function call**



```
[1]  def is_divisor(number: int, divisor: int) -> bool:
       if number % divisor == 0:
         return True
       else:
         return False
```

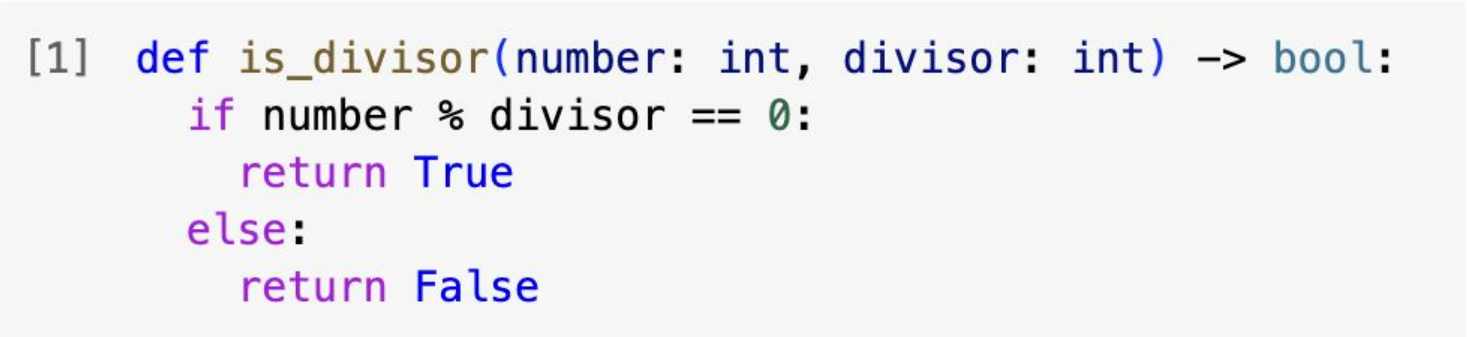


```
# test if 10 is a divisor of 100
result = is_divisor(10, 100)
print(result)
```

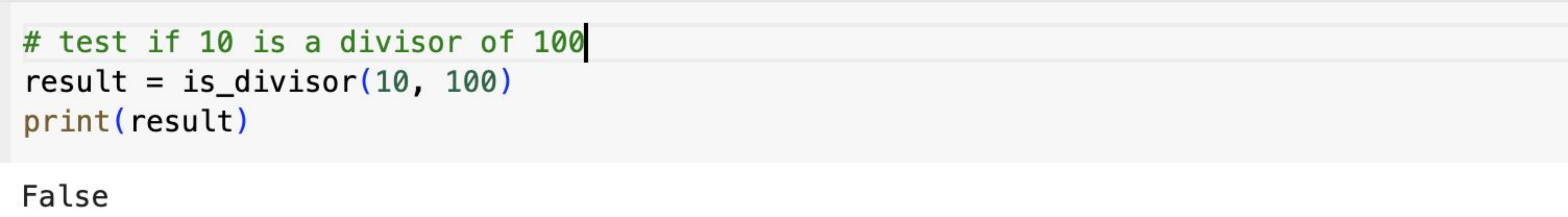
```
False
```

Positional Arguments

- Order of actual parameters matter!
- 10 will be bound to `number`; 100 will be bound to `divisor`



```
[1]  def is_divisor(number: int, divisor: int) -> bool:
       if number % divisor == 0:
         return True
       else:
         return False
```

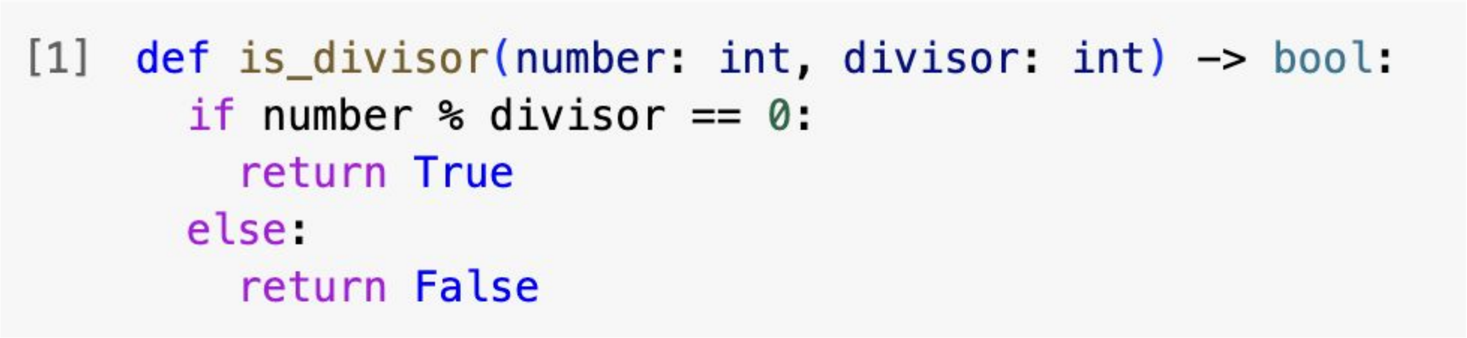


```
# test if 10 is a divisor of 100
result = is_divisor(10, 100)
print(result)
```

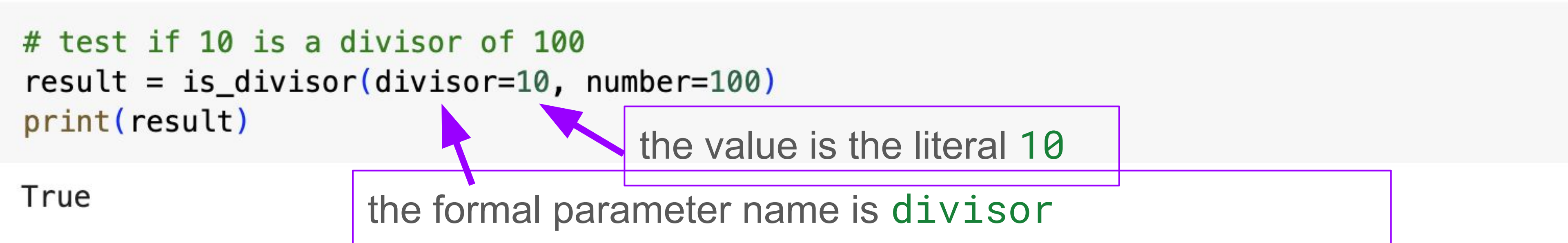
```
False
```

Keyword arguments

argument names are specified, followed by = and an actual parameter



```
[1]  def is_divisor(number: int, divisor: int) -> bool:
       if number % divisor == 0:
         return True
       else:
         return False
```



```
# test if 10 is a divisor of 100
result = is_divisor(divisor=10, number=100)
print(result)
```

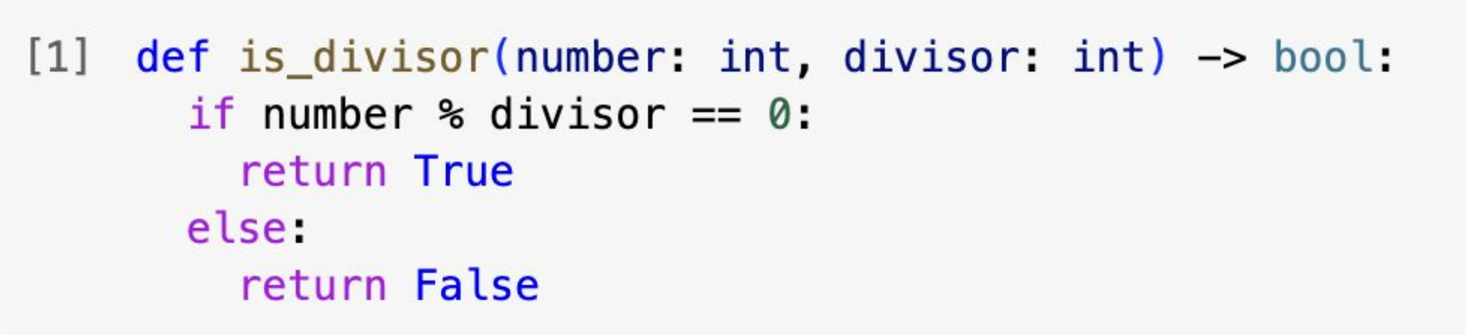
```
True
```

the value is the literal `10`

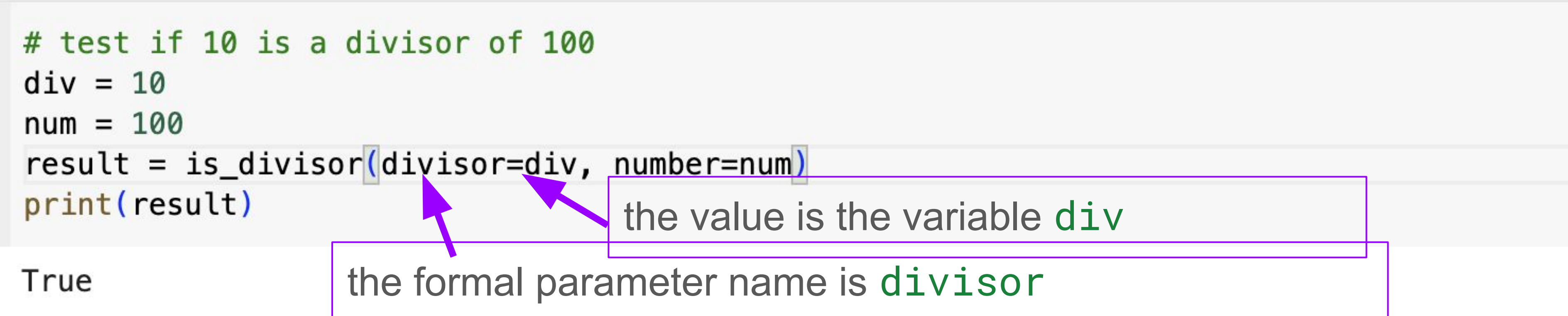
the formal parameter name is `divisor`

Keyword arguments

argument names are specified, followed by = and an actual parameter



```
[1]  def is_divisor(number: int, divisor: int) -> bool:
       if number % divisor == 0:
         return True
       else:
         return False
```



```
# test if 10 is a divisor of 100
div = 10
num = 100
result = is_divisor(divisor=div, number=num)
print(result)
```

```
True
```

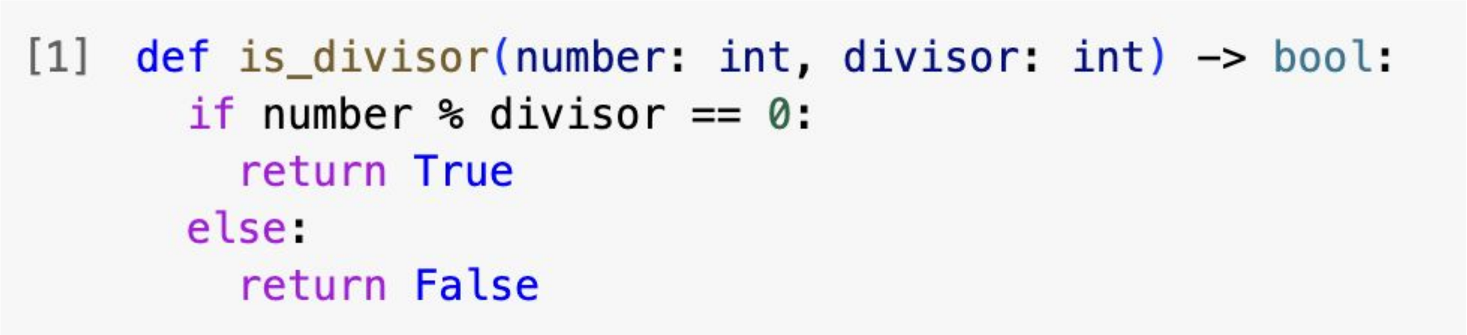
the value is the variable div

the formal parameter name is divisor

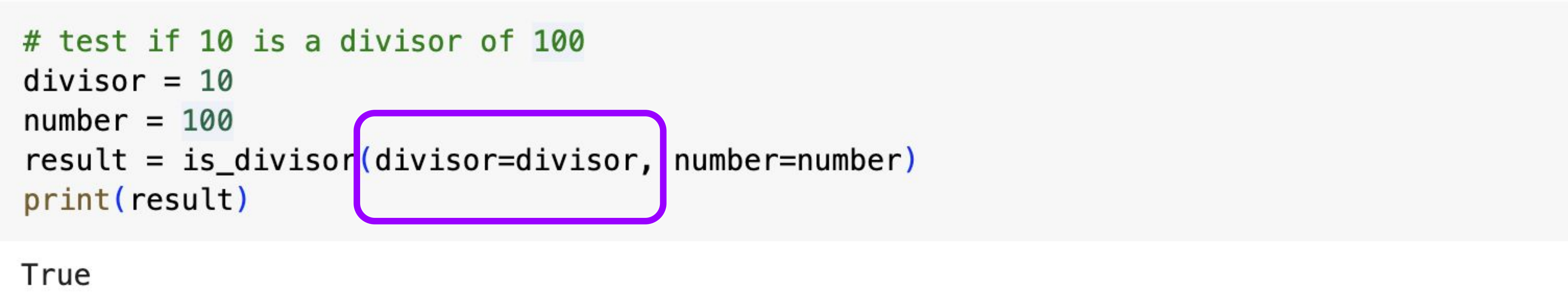
Keyword arguments

Discuss with your neighbor!

argument names are specified, followed by = and an actual parameter



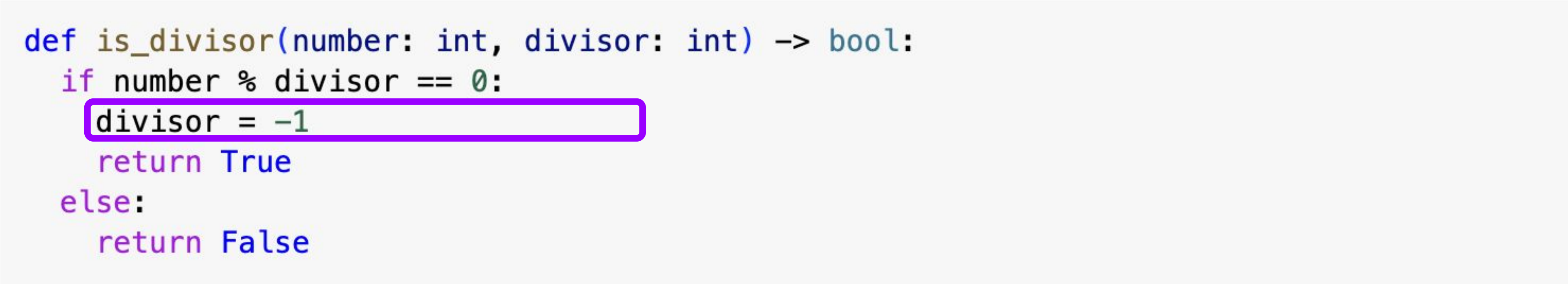
```
[1]  def is_divisor(number: int, divisor: int) -> bool:
       if number % divisor == 0:
         return True
       else:
         return False
```



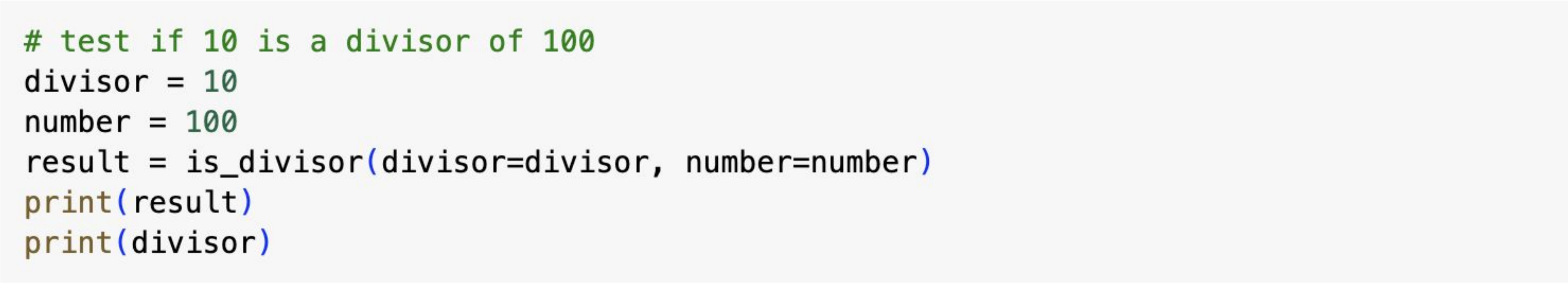
```
# test if 10 is a divisor of 100
divisor = 10
number = 100
result = is_divisor(divisor=divisor, number=number)
print(result)
```

```
True
```

Answer in Google Form - https://forms.gle/MLfcj1MWHrZWALZs6



```python
def is_divisor(number: int, divisor: int) -> bool:
  if number % divisor == 0:
    divisor = -1
    return True
  else:
    return False
```

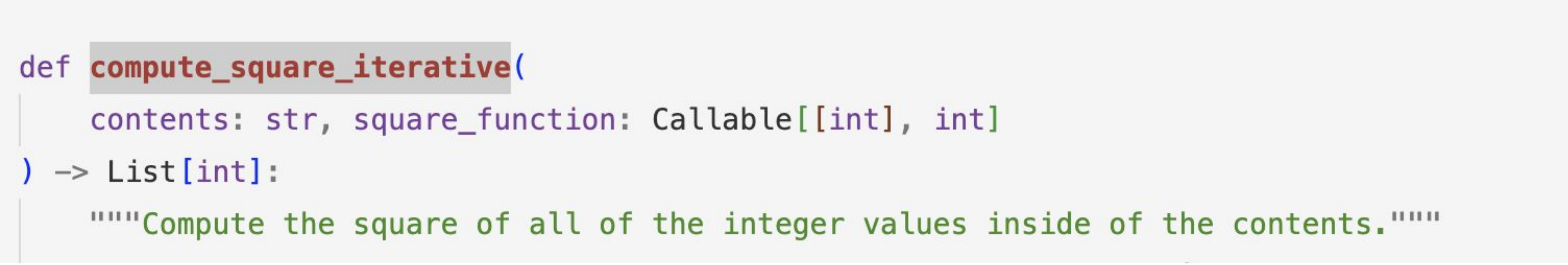


```python
# test if 10 is a divisor of 100
divisor = 10
number = 100
result = is_divisor(divisor=divisor, number=number)
print(result)
print(divisor)
```

Higher Order Functions

Functions can take functions as parameters

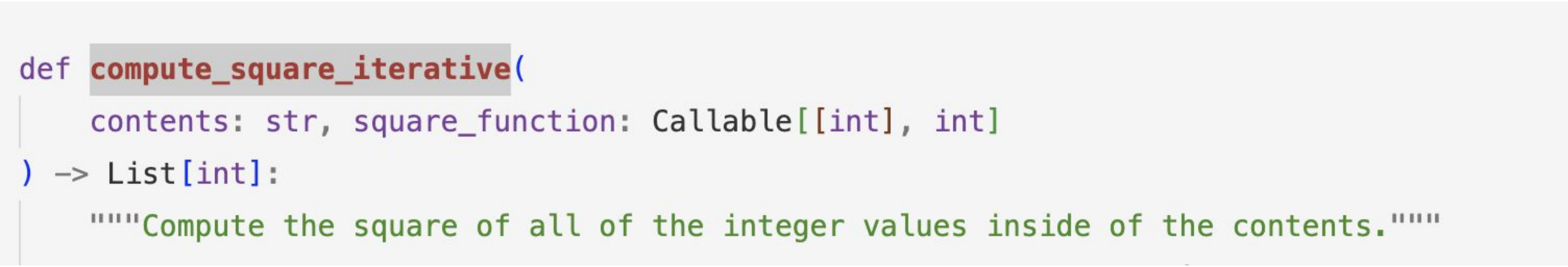
- positional or keyword!



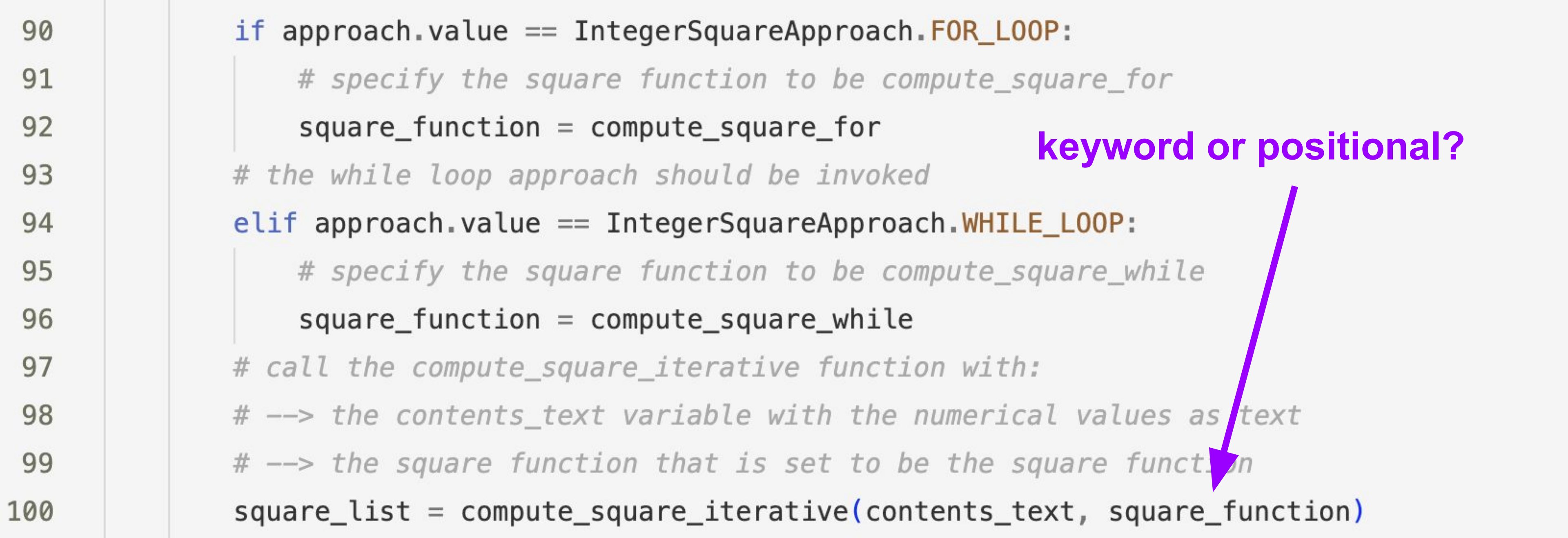
```python
def compute_square_iterative(
    contents: str, square_function: Callable[[int], int]
) -> List[int]:
    """Compute the square of all of the integer values inside of the contents."""
```

TODO

- locate: renaming of a function without calling it
- locate: function name inside function call
- identify which function is the higher-order function



```python
def compute_square_iterative(
    contents: str, square_function: Callable[[int], int]
) -> List[int]:
    """Compute the square of all of the integer values inside of the contents."""
```

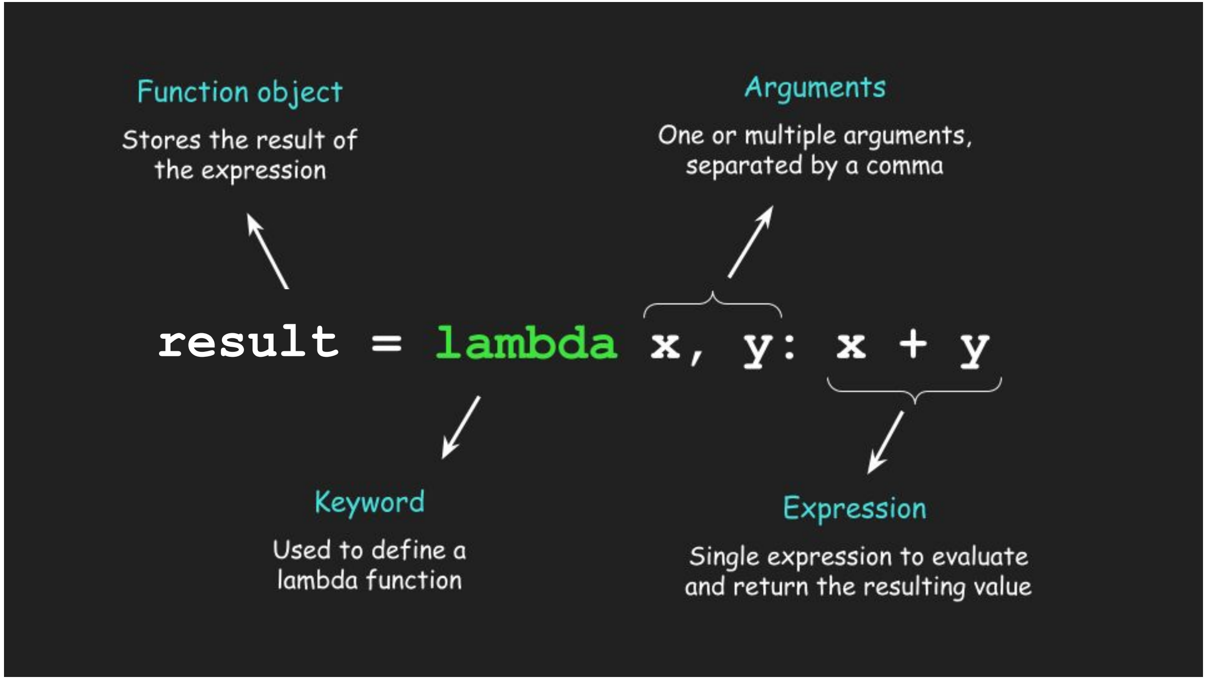


```python
        if approach.value == IntegerSquareApproach.FOR_LOOP:
            # specify the square function to be compute_square_for
            square_function = compute_square_for
        # the while loop approach should be invoked
        elif approach.value == IntegerSquareApproach.WHILE_LOOP:
            # specify the square function to be compute_square_while
            square_function = compute_square_while
        # call the compute_square_iterative function with:
        # --> the contents_text variable with the numerical values as text
        # --> the square function that is set to be the square function
        square_list = compute_square_iterative(contents_text, square_function)
```

keyword or positional?

lambdas

- anonymous functions (no name)
- special syntax



Function object

Stores the result of the expression

Arguments

One or multiple arguments, separated by a comma

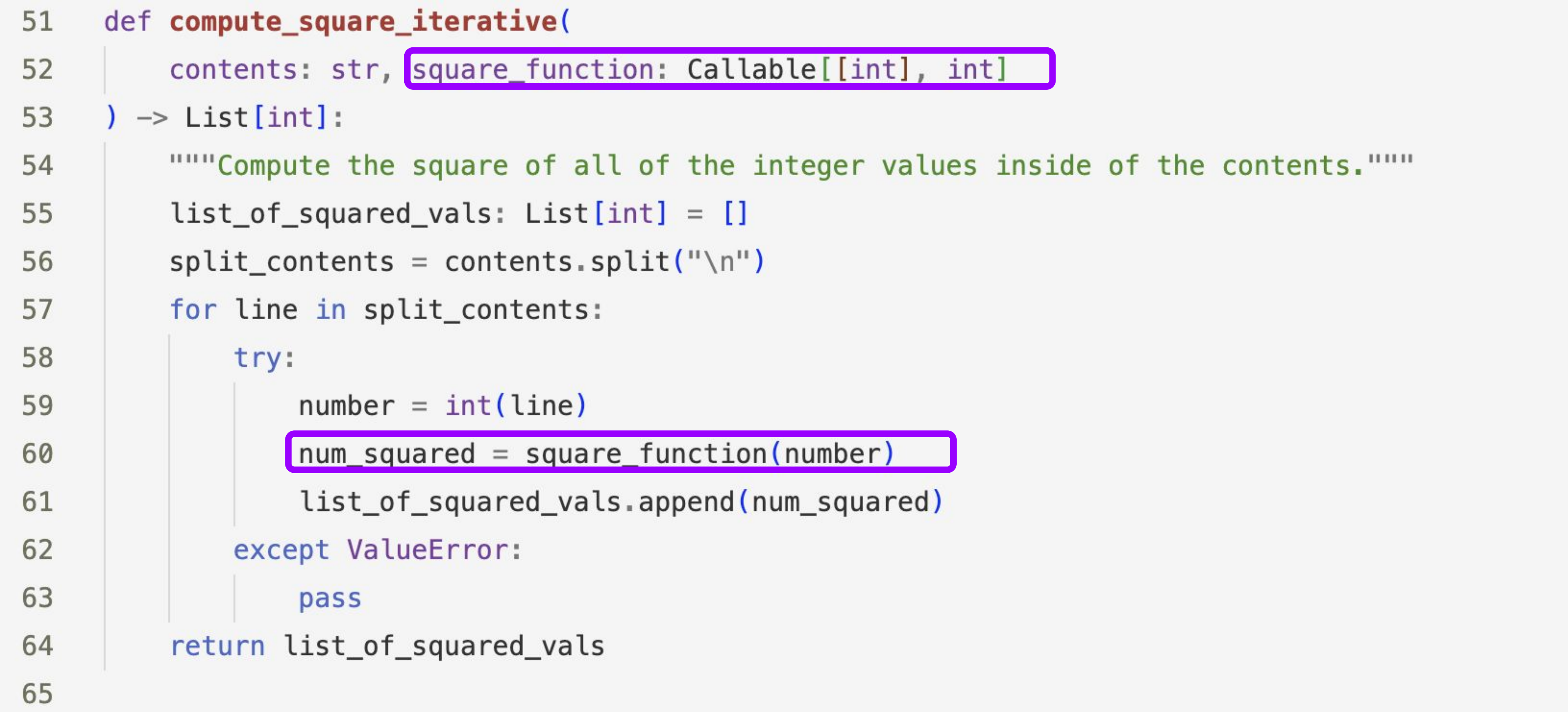
```
result = lambda x, y: x + y
```

Keyword

Used to define a lambda function

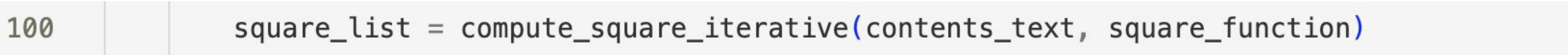
Expression

Single expression to evaluate and return the resulting value

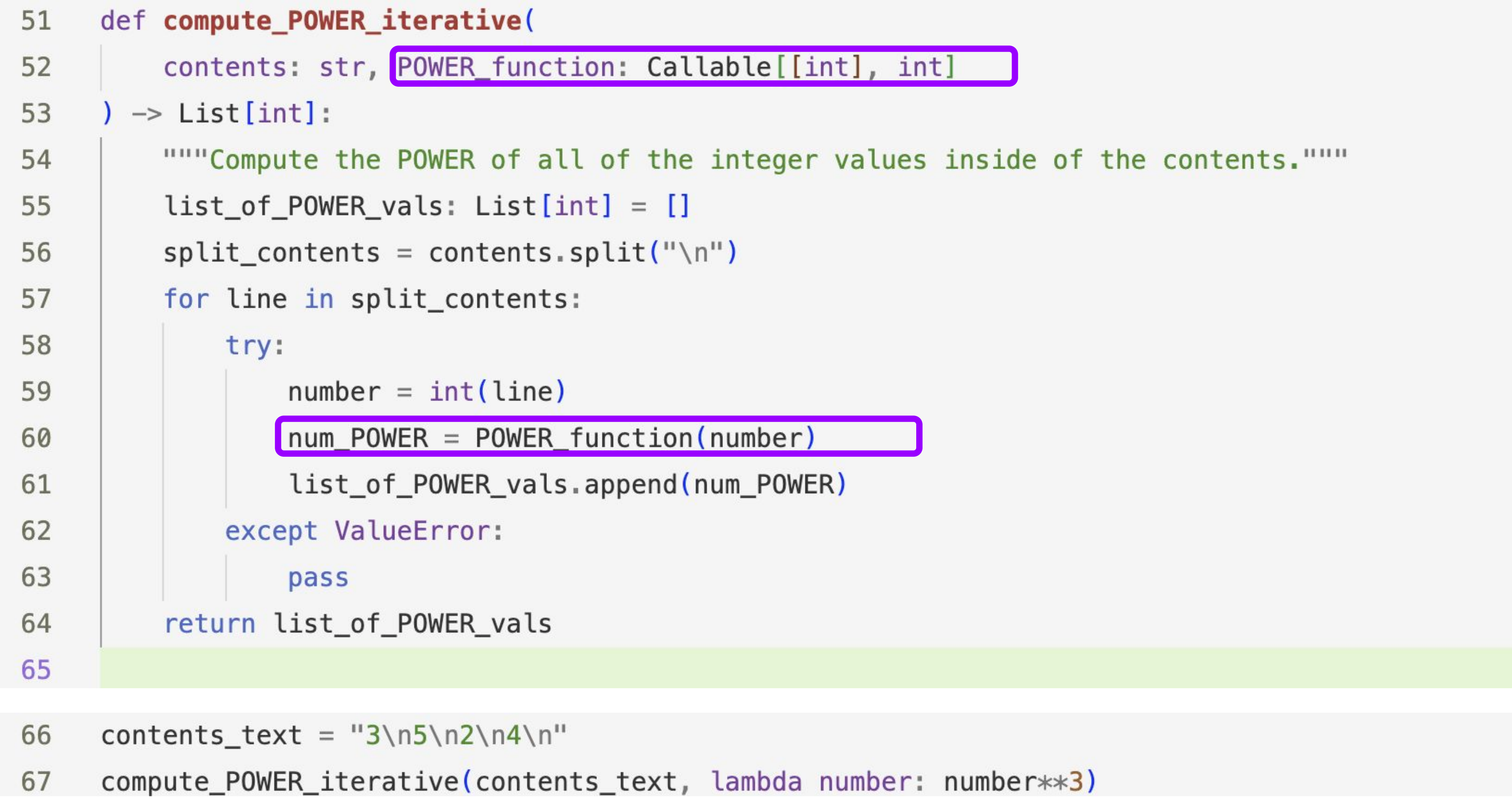


```python
def compute_square_iterative(
    contents: str, square_function: Callable[[int], int]
) -> List[int]:
    """Compute the square of all of the integer values inside of the contents."""
    list_of_squared_vals: List[int] = []
    split_contents = contents.split("\n")
    for line in split_contents:
        try:
            number = int(line)
            num_squared = square_function(number)
            list_of_squared_vals.append(num_squared)
        except ValueError:
            pass
    return list_of_squared_vals

```



```python
        square_list = compute_square_iterative(contents_text, square_function)
```



```python
def compute_POWER_iterative(
    contents: str, POWER_function: Callable[[int], int]
) -> List[int]:
    """Compute the POWER of all of the integer values inside of the contents."""
    list_of_POWER_vals: List[int] = []
    split_contents = contents.split("\n")
    for line in split_contents:
        try:
            number = int(line)
            num_POWER = POWER_function(number)
            list_of_POWER_vals.append(num_POWER)
        except ValueError:
            pass
    return list_of_POWER_vals

contents_text = "3\n5\n2\n4\n"
compute_POWER_iterative(contents_text, lambda number: number**3)
```

Explore these concepts

20240212_keyword_args_lambdas.ipynb

Midterm Reminder

Monday Feb 19th at 9am

- in person
- 50 minute exam
- you can work anywhere in Alden with an open door
- repo
- Just python scripts and markdown files (no poetry environments)
- python 3 is required. Make sure you can run a script using python 3

- create a one page cheat sheet ON PAPER to use during the exam
- Letter size, front and back