COMS BC1016

Introduction to Computational Thinking and Data Science

Lecture 10: Conditionals and Iteration

Sept. 3, 2025

February 23, 2026

Logistics

- Due to the blizzard, office hours are remote today
  - Professor office hours will be by appointment
  - See EdStem for Elena’s office hours link
- We’ll update via EdStem if more things get moved remote
- Due to midterm, **there will not be any labs next week (March 4/5) or the week after (March 11/12)**

Midterm Logistics

- Midterm will be an in-class paper exam on Wednesday, March 11
    - This is the week before Spring Recess
    - *Please contact CARDS or ODS as soon as possible if you need any accommodations*
- You are permitted to bring a single formula sheet (5”x8” index card, double-sided) that will be submitted along with the exam
    - Exam is otherwise closed-note and no computers
- Questions will be a mix of multiple choice and short answer
    - You will not be asked to write programs on paper but you should expect to read code and understand it
- There will be a review session during class on Monday, March 9

Control Statements

Control Statements

Control Statements modify *if* and/or *how many times* a block of code is executed in a program

- Two major types are **`if`** and **`for`**
    - **`if`** statements specify code that should be run conditioned on something being true
        - They can also specify if alternative code should be run otherwise
    - **`for`** loops allow executing code over each element in some sequence of items

`if` statements

- Conditionals begin with an `if` followed by a boolean statement
    - Runs code based on whether a boolean statement evaluates to `True`
- Conditionals can include a combination of `if`, `elif`, and `else` clauses
    - Maximum of one `if` and one `else`

if statements

```
if statement_1:
    first_code_block
```

Runs if statement_1 == True

if statements

```
if statement_1:
    first_code_block
else:
    second_code_block
```

Runs if statement_1 == True

nothing above == True

if statements

Shorthand for "else if"

```
if statement_1:
    first_code_block
elif statement_2:
    second_code_block
else:
    third_code_block
```

Runs if statement_1 == True

Runs if statement_1 != True
AND statement_2 == True

nothing above == True

`if` statements

Shorthand for "else if"

```
if statement_1:
    first_code_block
elif statement_2:
    second_code_block
```

Runs if statement_1 == True

Runs if statement_1 != True

AND statement_2 == True

if statements

```
if statement_1:
    first_code_block
elif statement_2:
    second_code_block
elif statement_3:
    third_code_block
else:
    fourth_code_block
```

Shorthand for "else if"

Runs if statement_1 == True

Runs if statement_1 != True
AND statement_2 == True

statement_1 != True
AND statement_2 != True
AND statement_3 == True

nothing above == True

Booleans and Comparisons

Boolean Data Type

- Booleans are data types for truth values: **`True`** or **`False`**
  - **`True`** is equivalent to `1`
  - **`False`** is equivalent to `0`
- `bool(x)` turns `x` into a boolean
  - e.g., `bool(1)` evaluates to **`True`** and `bool(0)` evaluates to **`False`**

Comparison Operators

Operation	Meaning
`>`	greater than
`>=`	greater than or equal to
`<`	less than
`<=`	less than or equal to
`==`	equal to
`!=`	not equal to

Comparison Operators

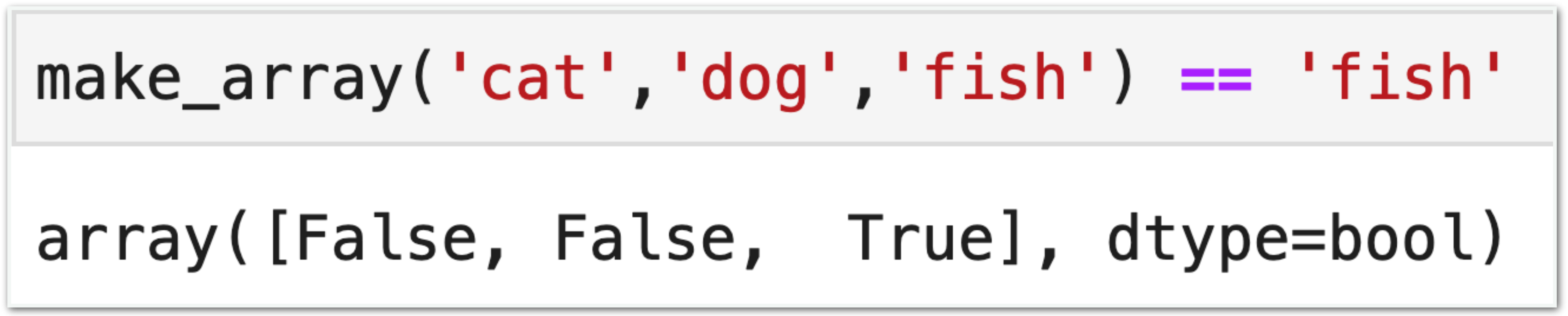
Example	Result	Explanation
`3 > 2`	**`True`**	3 is greater than 2
`3 > 3`	**`False`**	3 is not (*strictly*) *greater than* 3
`4 <= 4`	**`True`**	4 is less than or equal to 4

Comparison Operators

Example	Result	Explanation
`'4' == 4`	**`False`**	`'4'` is a string and `4` is an int
`(3 - 2)==(4 - 3)`	**`True`**	`3-2` equals `1` and `4-3` equals `1`; `1` equals `1`
`2 != 2`	**`False`**	`2` is not *not* equal to `2`

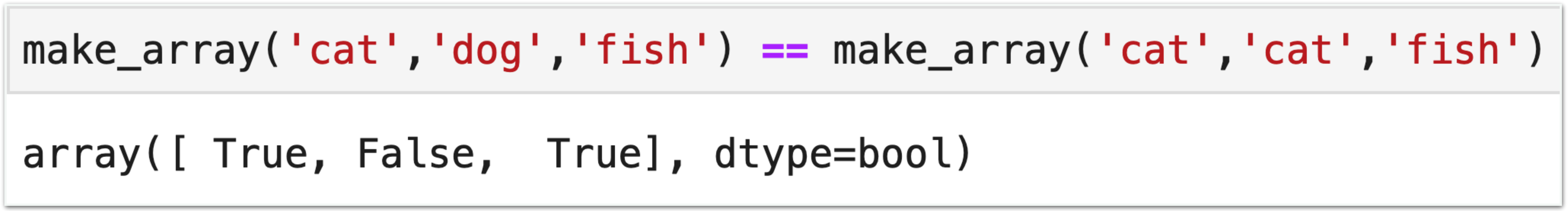
Comparisons with Arrays

- Single values can be compared against each element in an array
- Comparing two arrays will compare element-by-element



```
make_array('cat','dog','fish') == 'fish'
```

```
array([False, False,  True], dtype=bool)
```



```
make_array('cat','dog','fish') == make_array('cat','cat','fish')
```

```
array([ True, False,  True], dtype=bool)
```

Aggregating Comparisons

- `True` is equivalent to `1`, so summing an array or list of bool values counts the number of `True` values

Example	Result
`True + False + True`	2
`1 + 0 + 1`	2
`sum([True, False, True])`	2

Logical operators

- You can combine conditional statements using **`and`** & **`or`**
  - **`and`** will return **`True`** if **all** expressions are **`True`** (and **`False`** otherwise)
    - “Is a Barnard student” and “Is a first year”
  - **`or`** will return **`True`** if **any** expressions is **`True`** (and **`False`** otherwise)
    - “Is a Barnard student” or “Is a first year”

Logical operators

- You can negate a boolean value using **`not`**
  - **`not True`** will evaluate to **`False`**
  - **`not False`** will evaluate to **`True`**
- You can chain many expressions together, be careful using too many at once (it can be tricky for you or any reader to keep track of)
  - When it doubt, it can be helpful to put parentheses around each statement
  - Example: `(not x in my_array) and (i < 500)`

`and`, `or`, and `not`

Example	Result
`True and True`	

`and`, `or`, and `not`

Example	Result
`True and True`	`True`

`and`, `or`, and `not`

Example	Result
`True and True`	`True`
`True and False`	

`and`, `or`, and `not`

Example	Result
`True and True`	`True`
`True and False`	`False`

`and`, `or`, and `not`

Example	Result
`True and True`	`True`
`True and False`	`False`
`True or False`	

`and`, `or`, and `not`

Example	Result
`True and True`	`True`
`True and False`	`False`
`True or False`	`True`

and, or, and not

Example	Result
`True and True`	`True`
`True and False`	`False`
`True or False`	`True`
`False or False`	

`and`, `or`, and `not`

Example	Result
`True and True`	`True`
`True and False`	`False`
`True or False`	`True`
`False or False`	`False`

`and`, `or`, and `not`

Example	Result
`True and True`	`True`
`True and False`	`False`
`True or False`	`True`
`False or False`	`False`
`not False`	

`and`, `or`, and `not`

Example	Result
`True and True`	`True`
`True and False`	`False`
`True or False`	`True`
`False or False`	`False`
`not False`	`True`

For Loops

Iteration

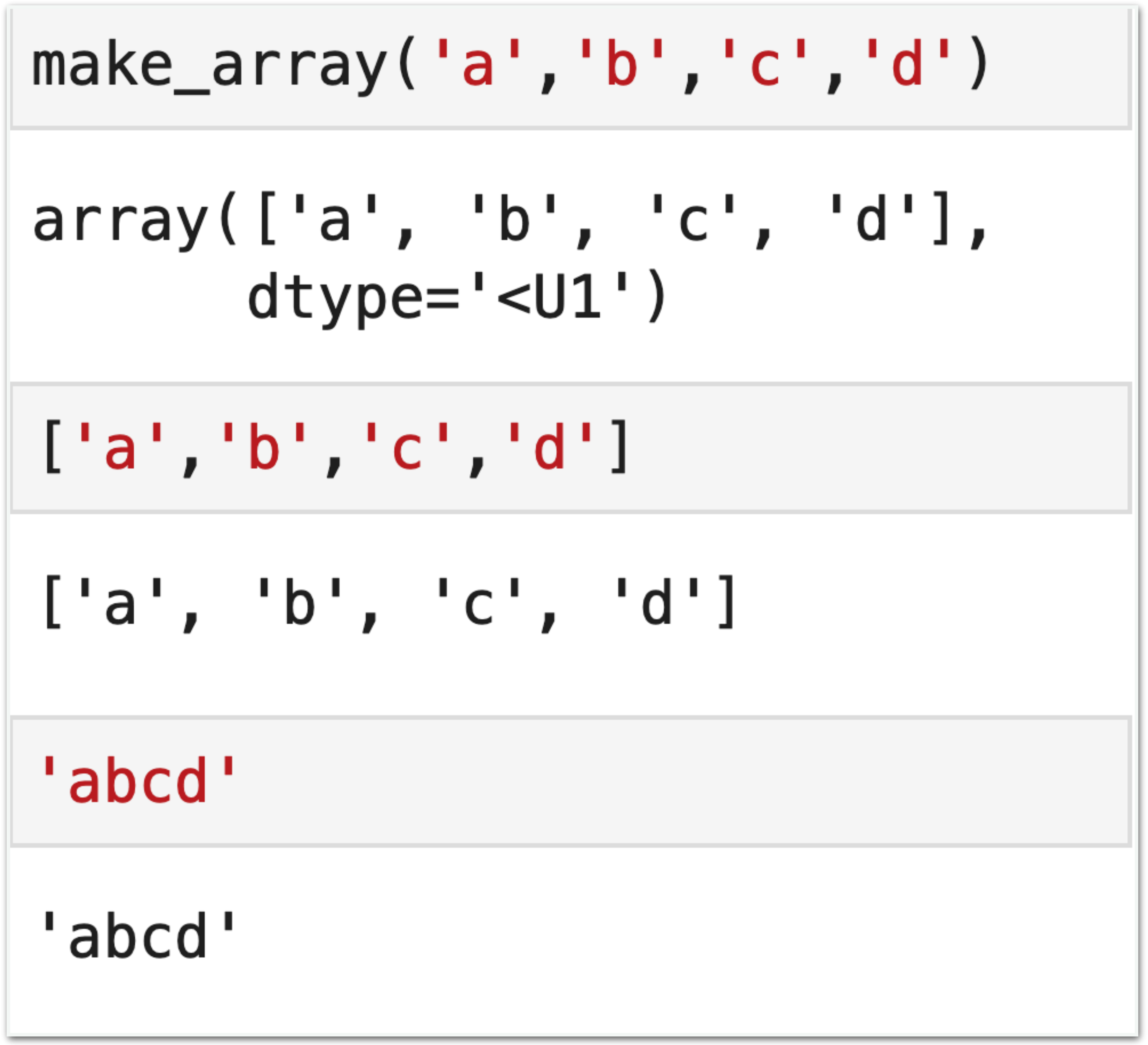
- **Iteration** means to repeat a process or steps
  - For example, coming up with a design, prototyping, testing, and then repeating these steps based on the outcome
- In programming we use this term to refer to executing code repeatedly over every element in a list/array/sequence/collection/…
  - The object being iterated over is referred to as an **iterable**

Iterables

- Formally, an iterable is any Python object capable of returning its members one at a time

- Iterables we've seen in this class include:
  - Arrays
  - Lists
  - String

We'll mostly focus on arrays



```
make_array('a','b','c','d')
```

```
array(['a', 'b', 'c', 'd'],
      dtype='<U1')
```

```
['a','b','c','d']
```

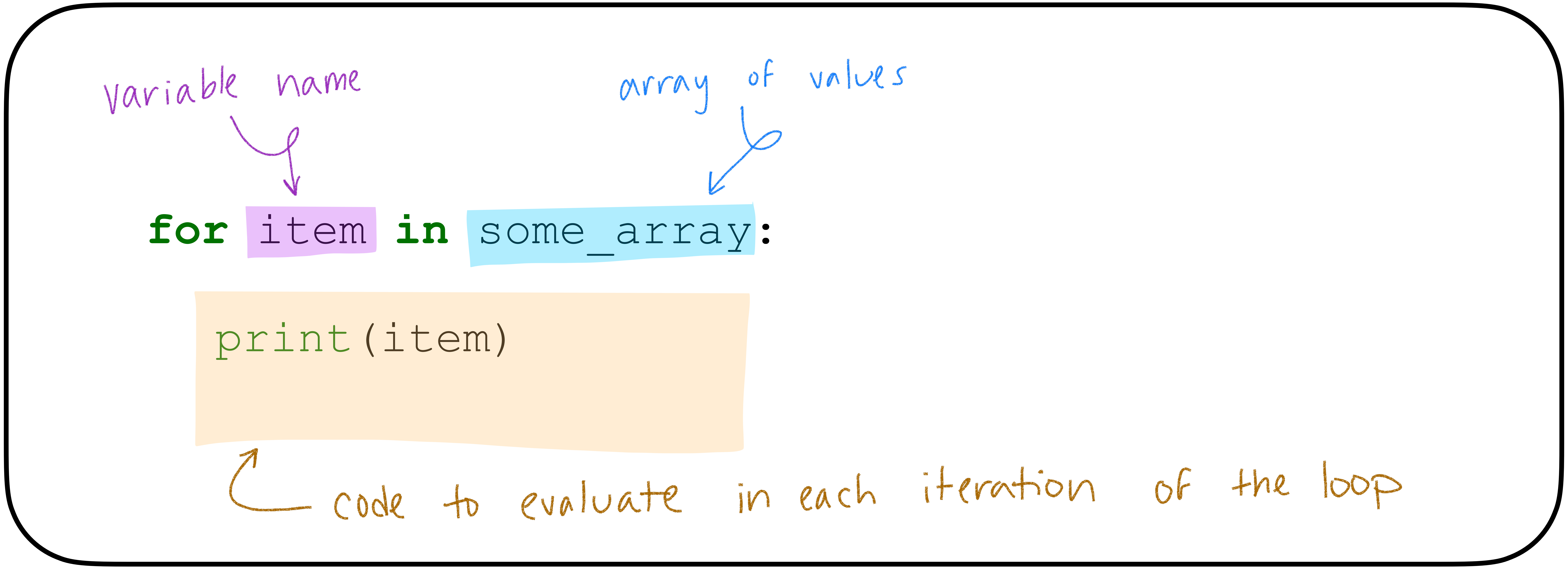
```
['a', 'b', 'c', 'd']
```

```
'abcd'
```

```
'abcd'
```

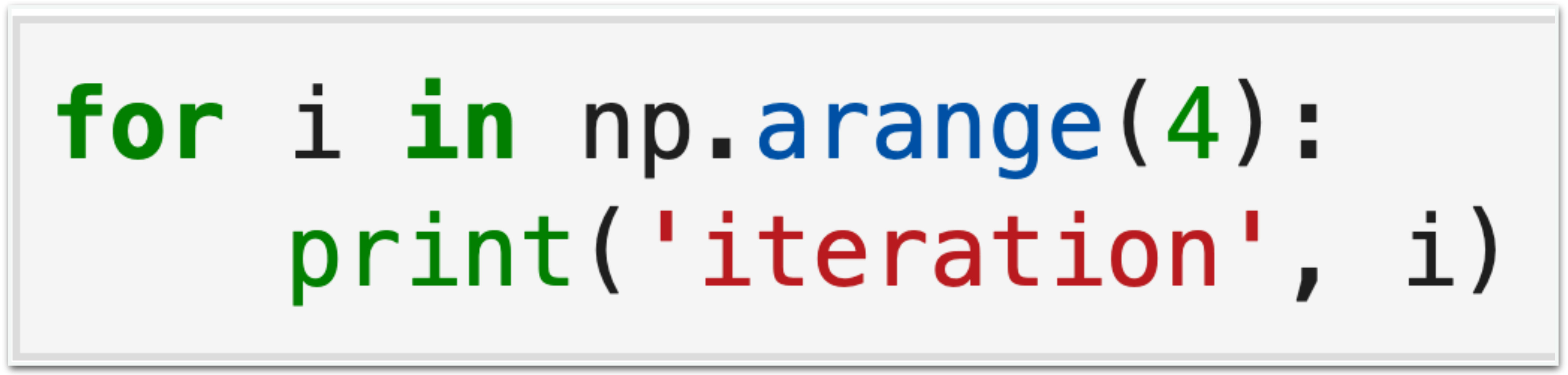
`for` Statements

- Executing a `for` runs code with each element in an iterable



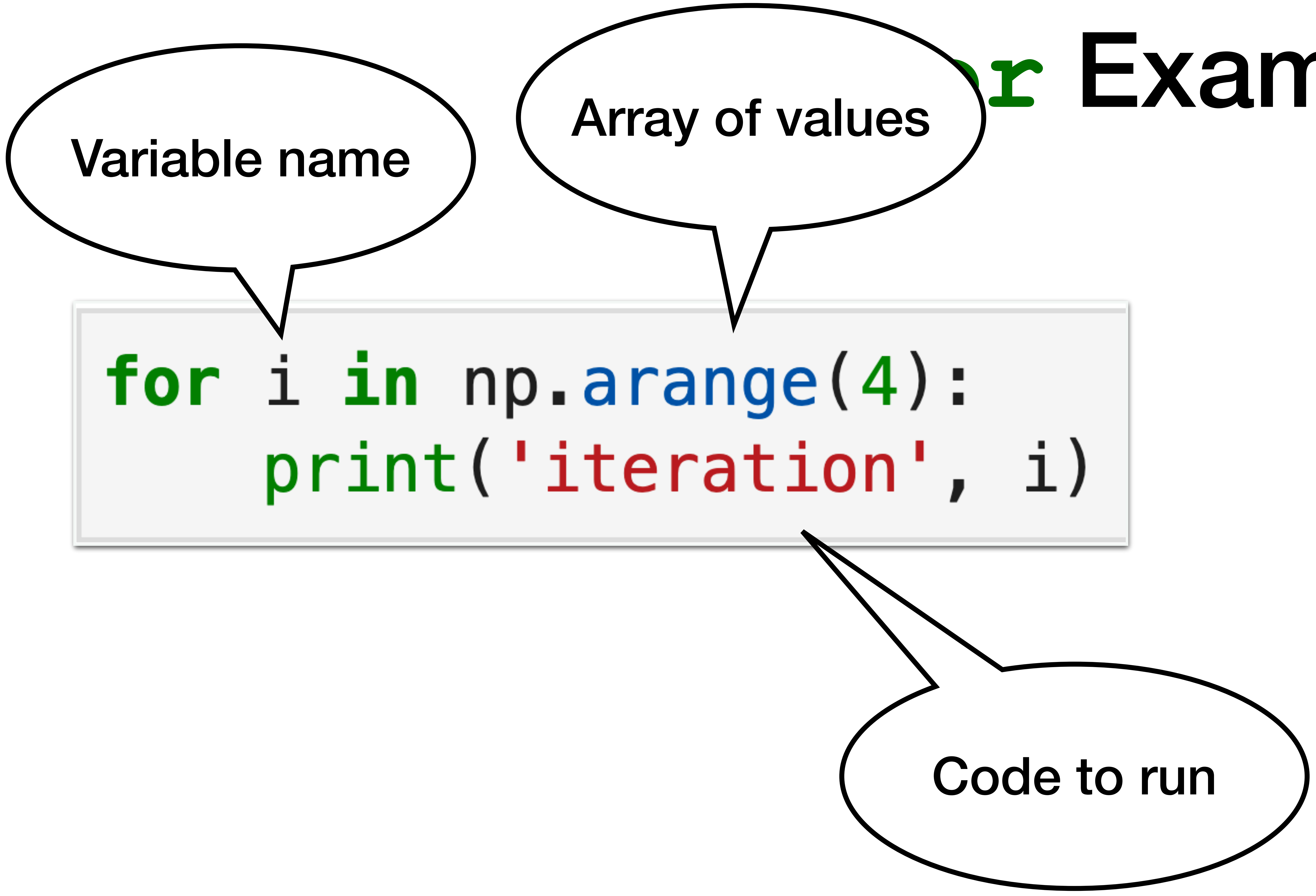
```
for item in some_array:
    print(item)
```

for Example



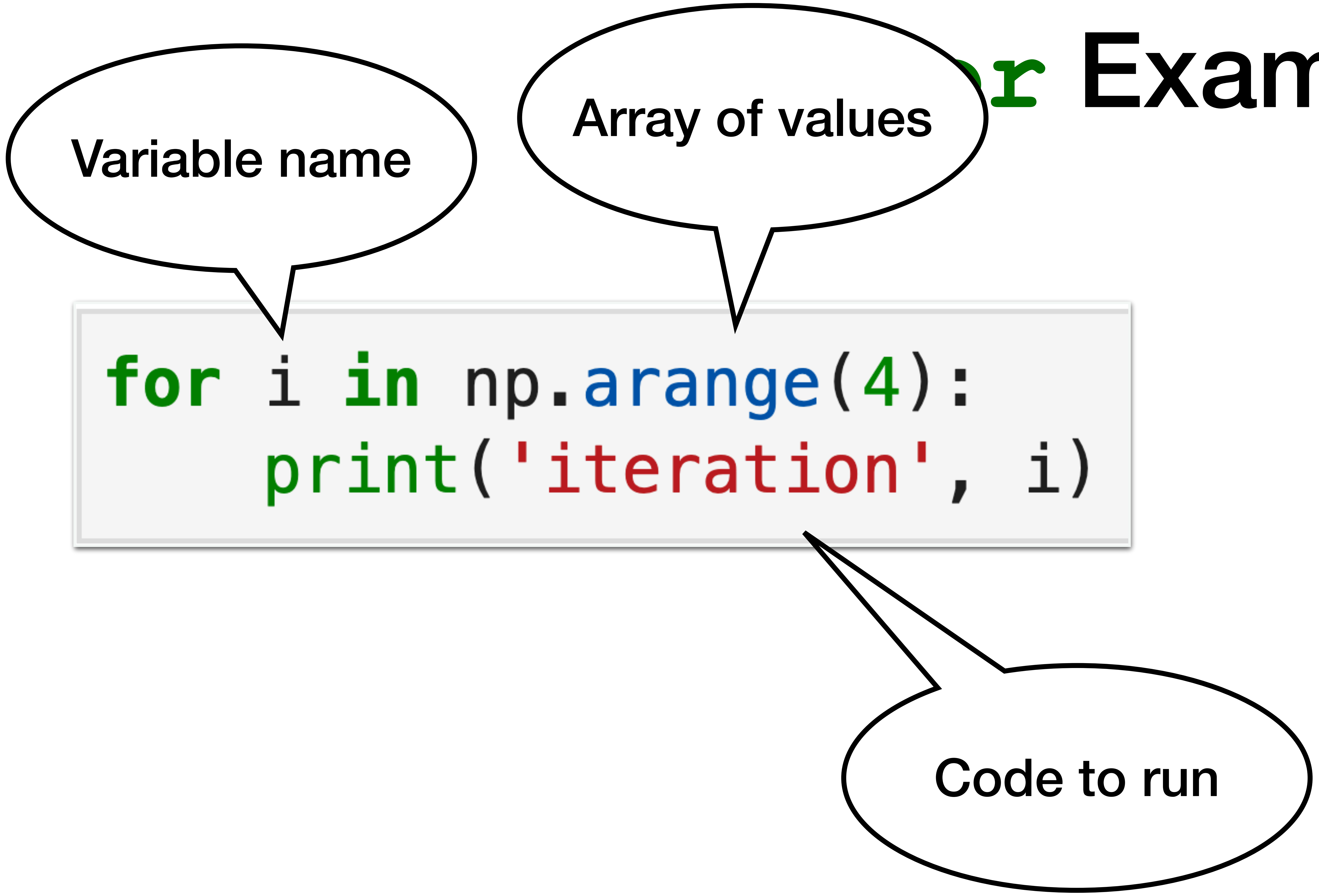
```
for i in np.arange(4):
    print('iteration', i)
```

r Example



```python
for i in np.arange(4):
    print('iteration', i)
```

r Example

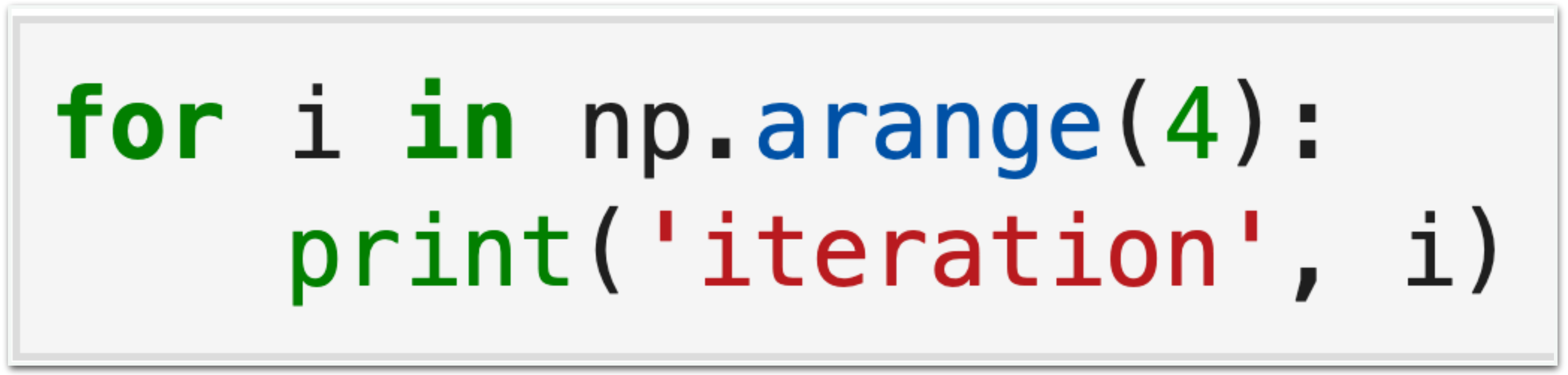


```python
for i in np.arange(4):
    print('iteration', i)
```

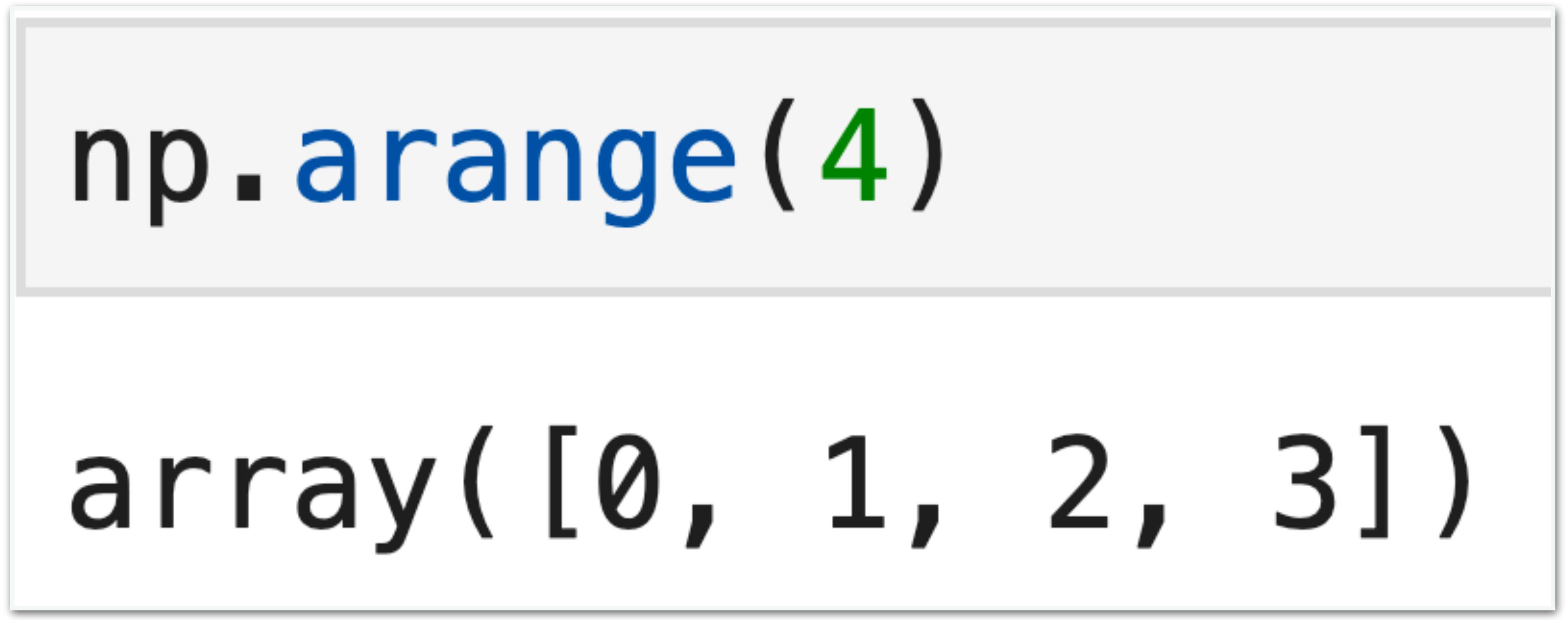
```python
np.arange(4)
```

```
array([0, 1, 2, 3])
```

for Example



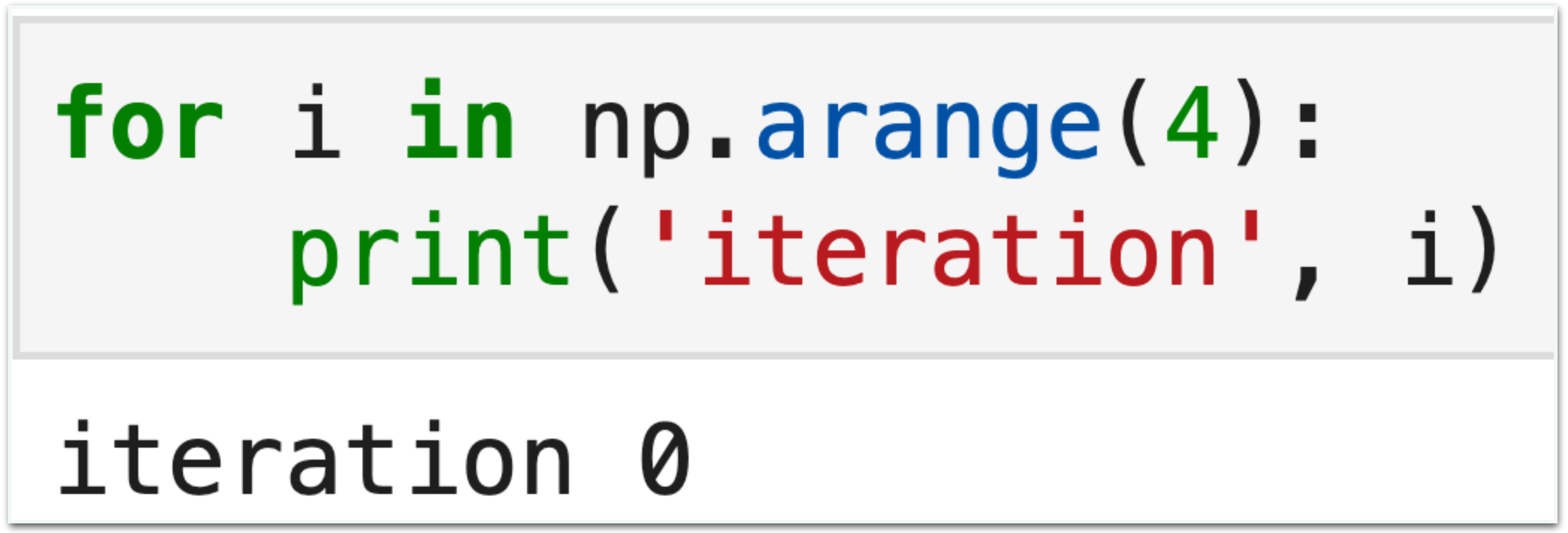
```python
for i in np.arange(4):
    print('iteration', i)
```



```python
np.arange(4)
```

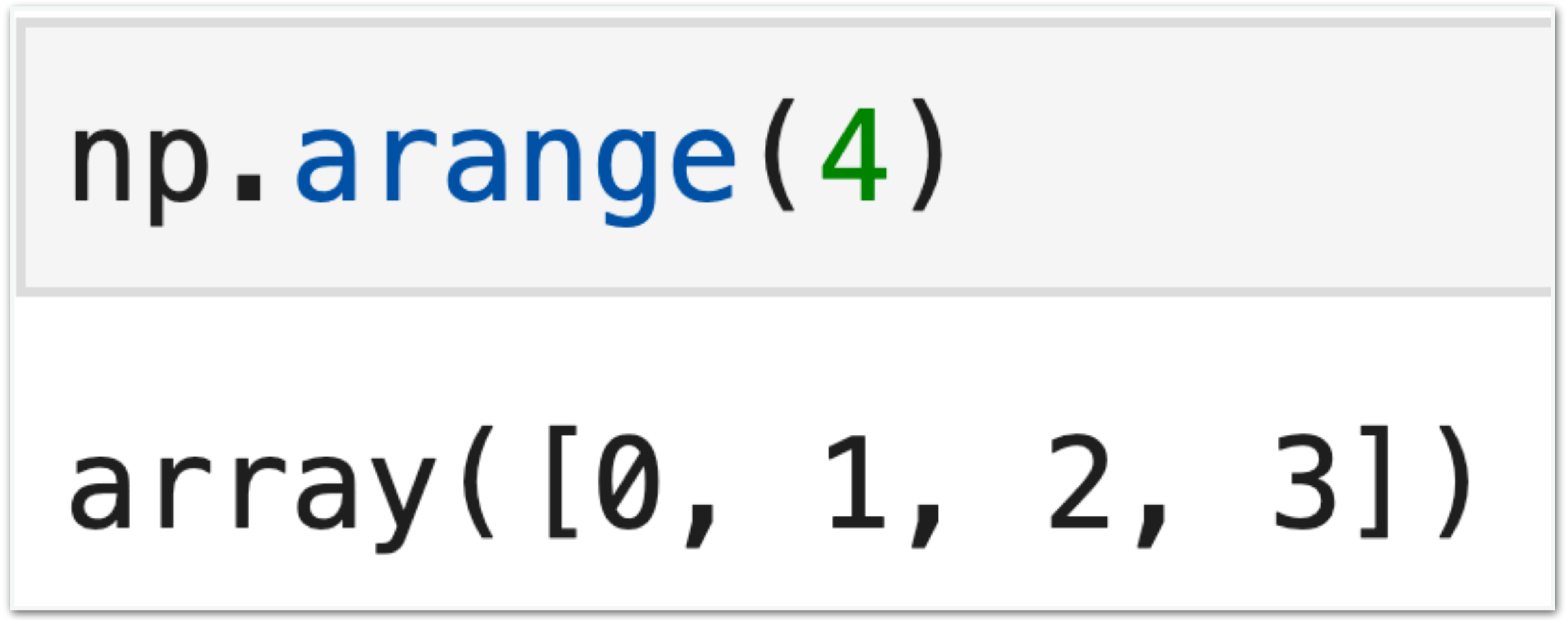
```
array([0, 1, 2, 3])
```

for Example



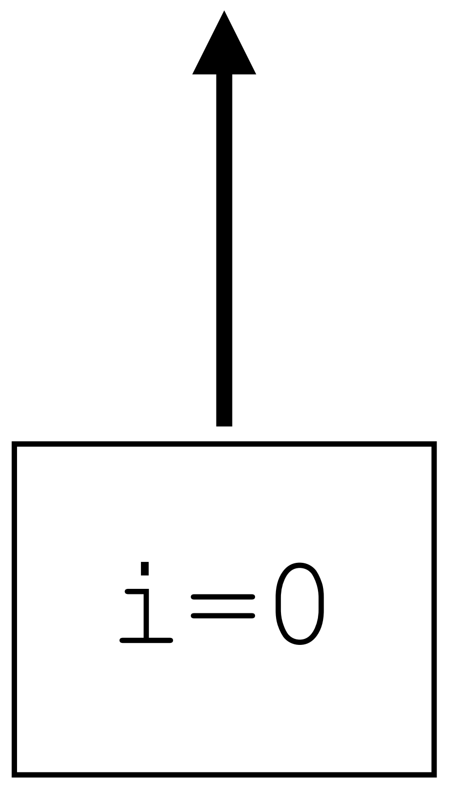
```
for i in np.arange(4):
    print('iteration', i)
```

```
iteration 0
```

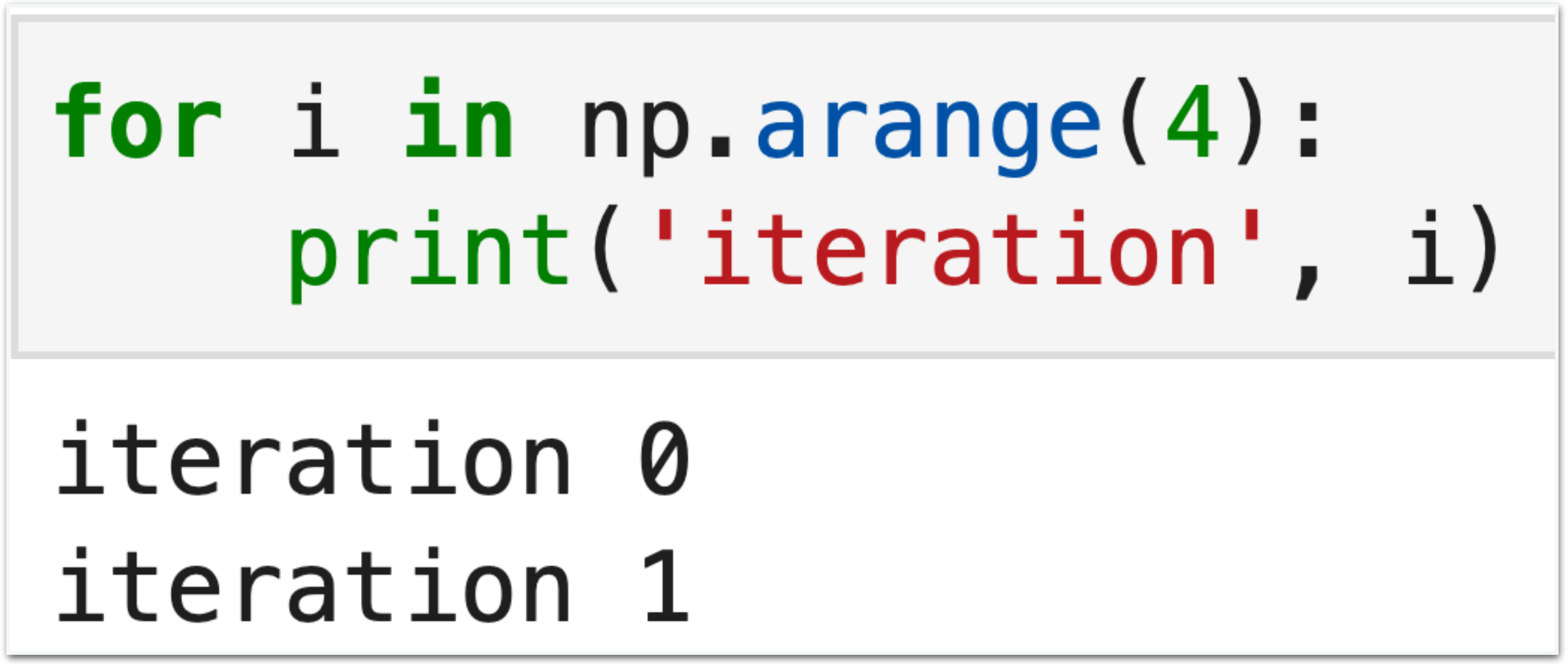


```
np.arange(4)
```

```
array([0, 1, 2, 3])
```

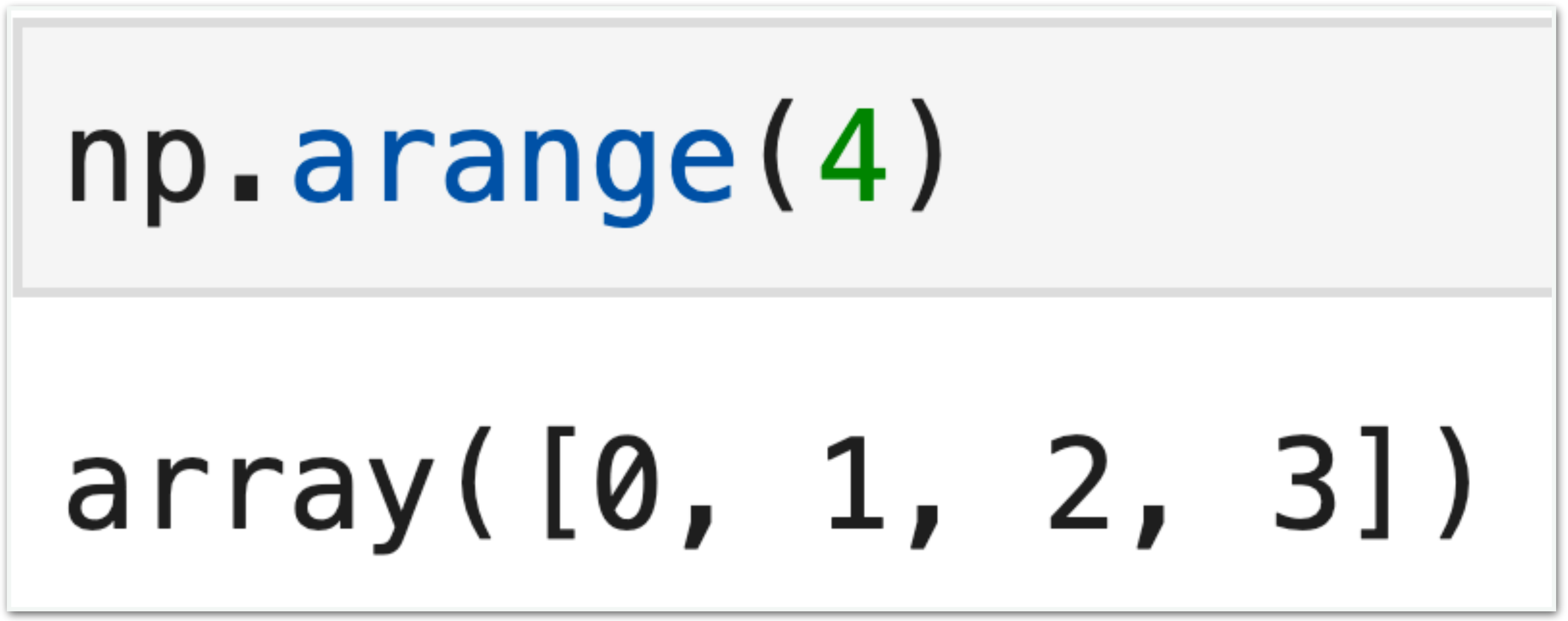


for Example



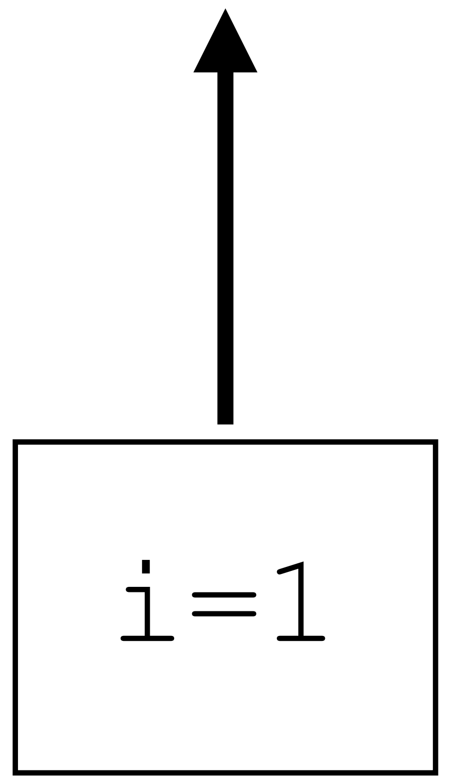
```
for i in np.arange(4):
    print('iteration', i)
```

```
iteration 0
iteration 1
```

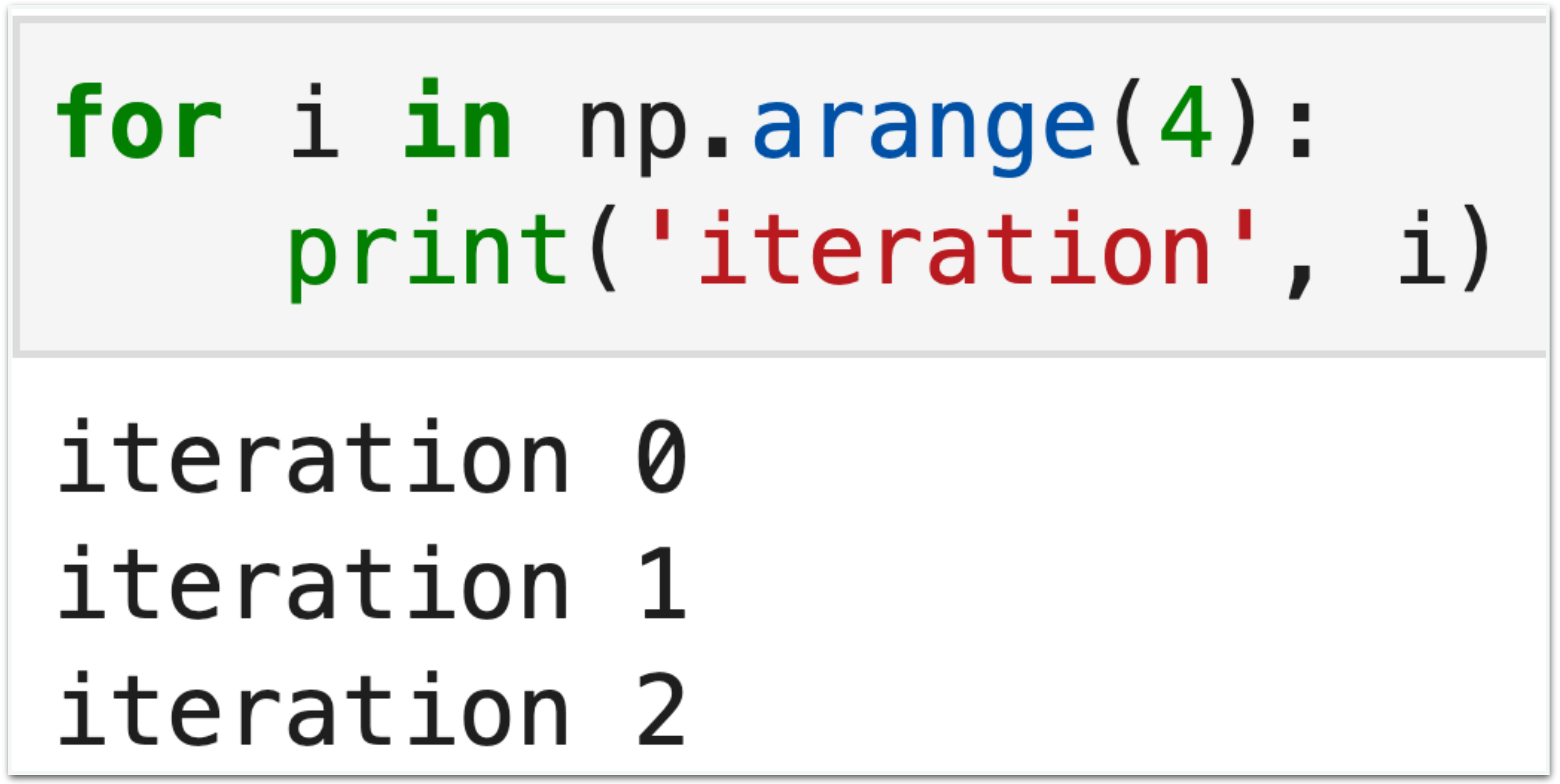


```
np.arange(4)
```

```
array([0, 1, 2, 3])
```

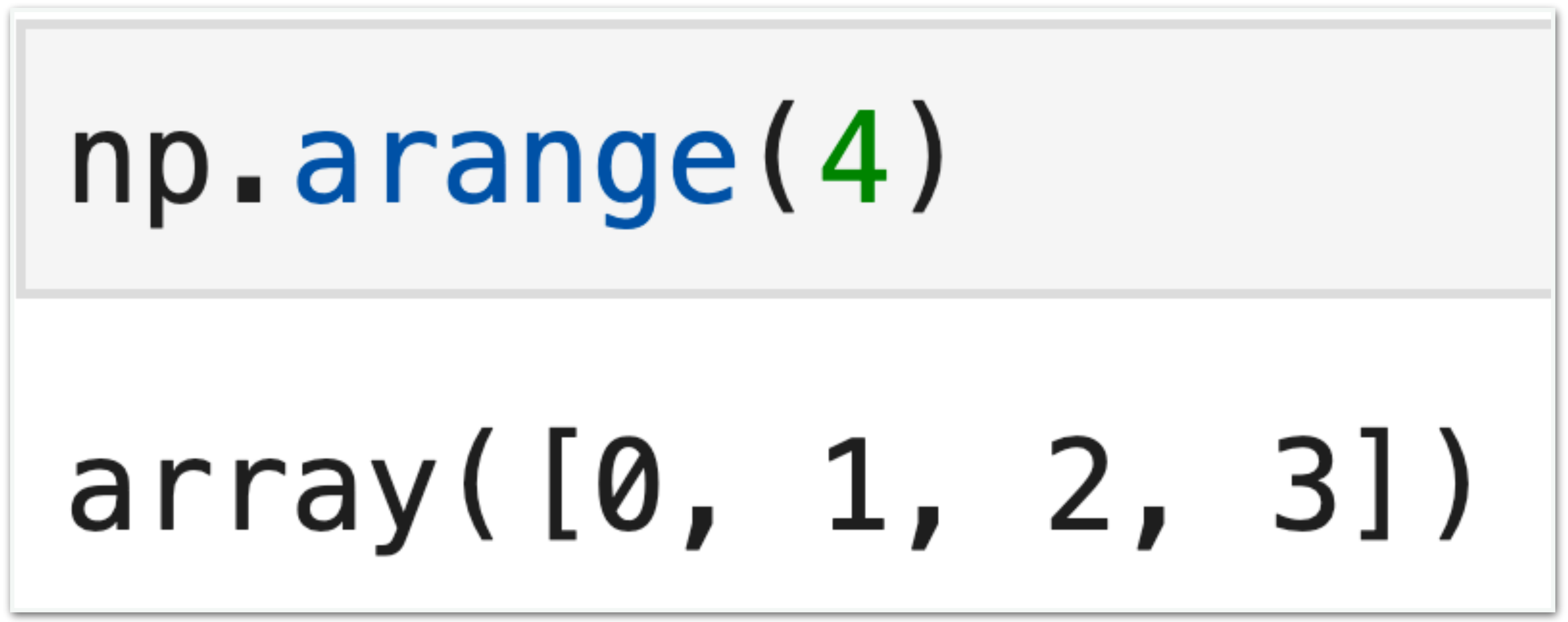


for Example



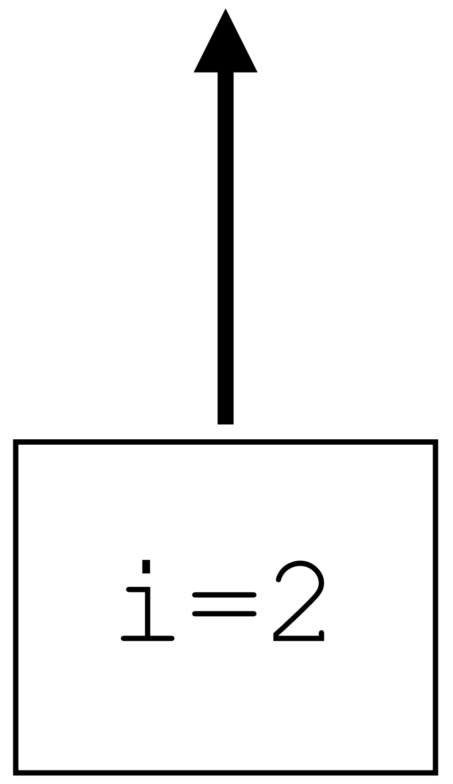
```
for i in np.arange(4):
    print('iteration', i)
```

```
iteration 0
iteration 1
iteration 2
```

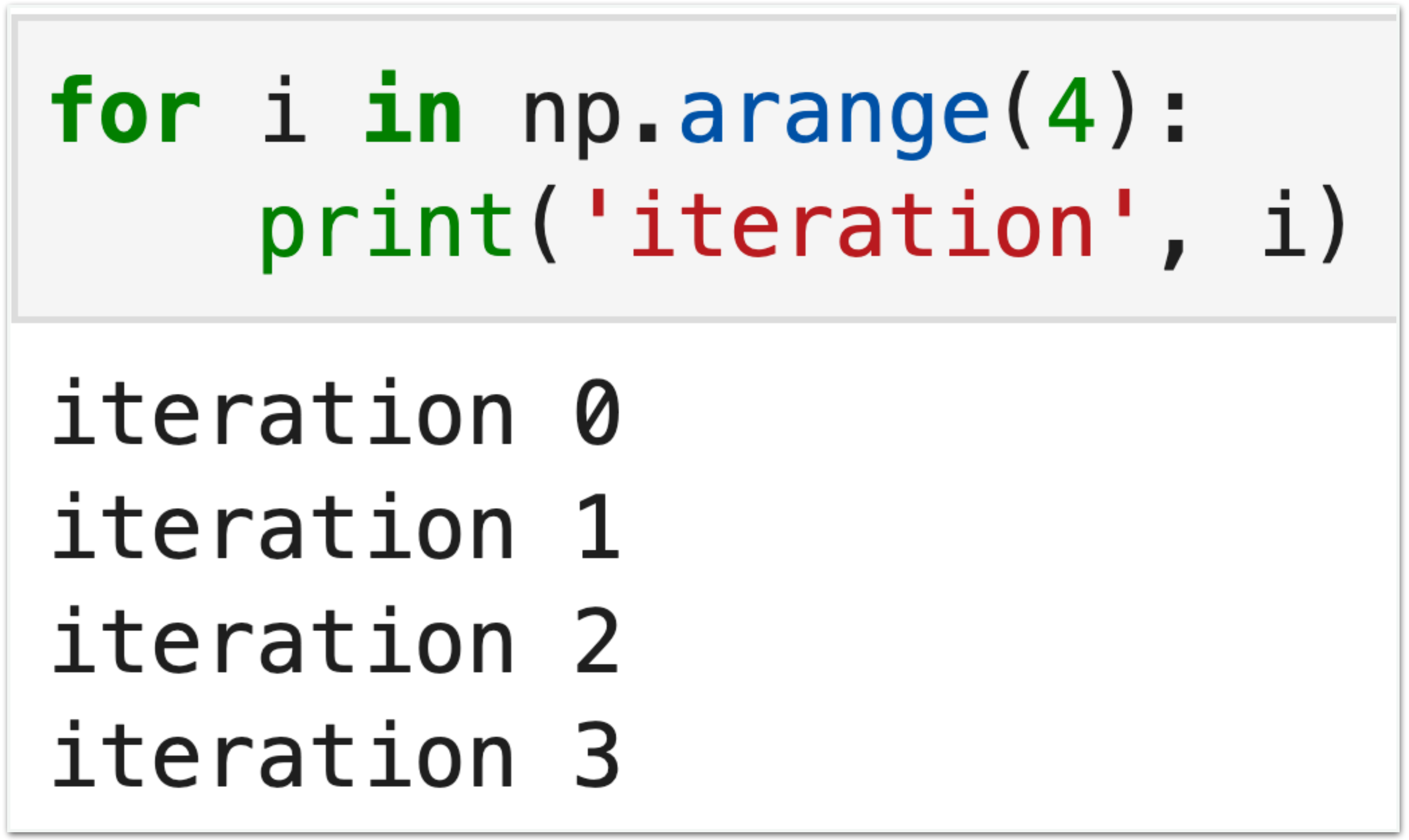


```
np.arange(4)
```

```
array([0, 1, 2, 3])
```

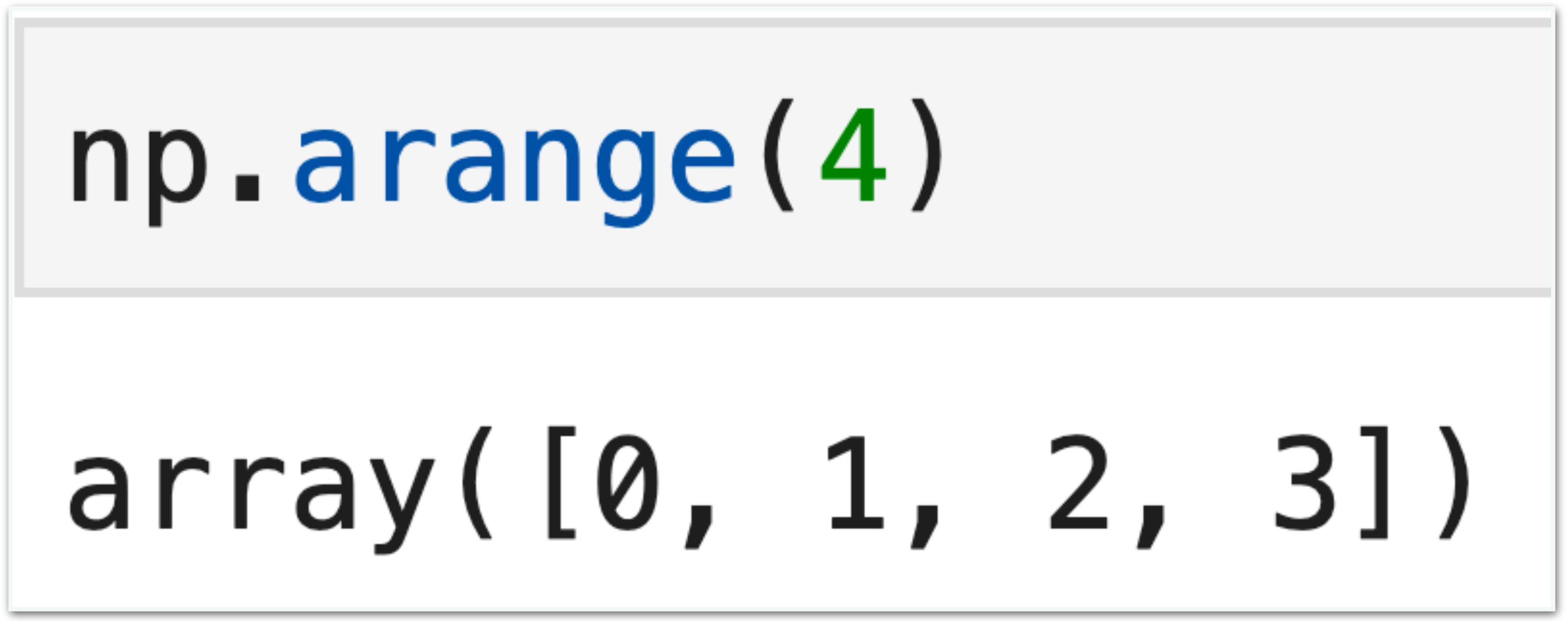


for Example



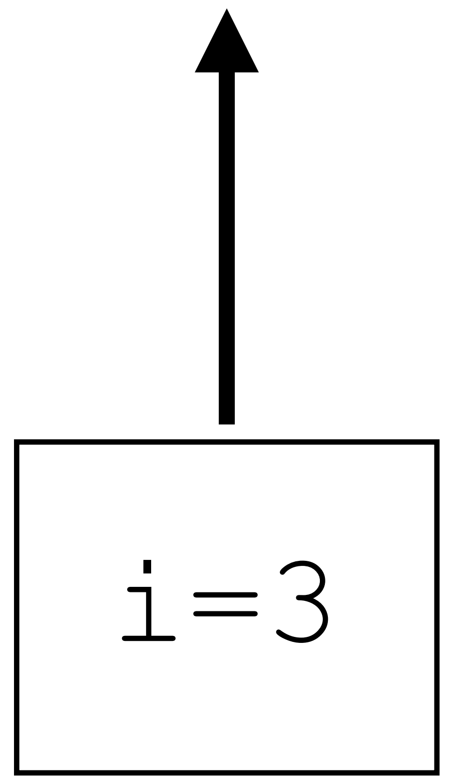
```python
for i in np.arange(4):
    print('iteration', i)
```

```
iteration 0
iteration 1
iteration 2
iteration 3
```

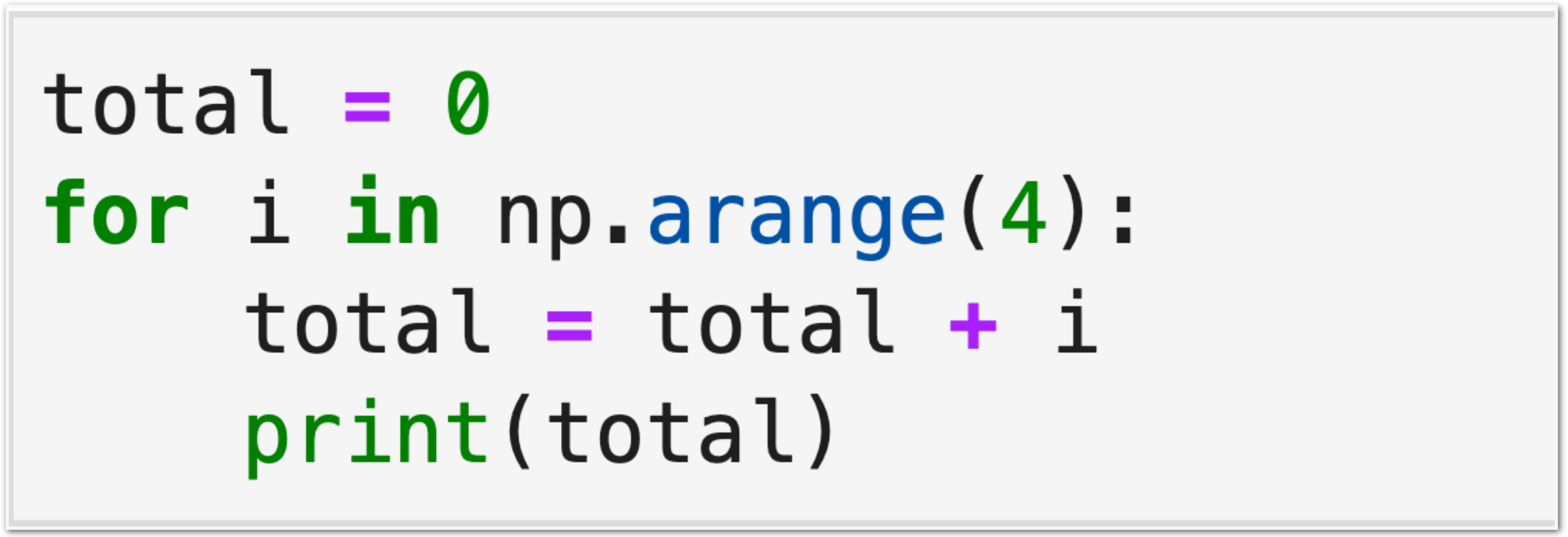


```python
np.arange(4)
```

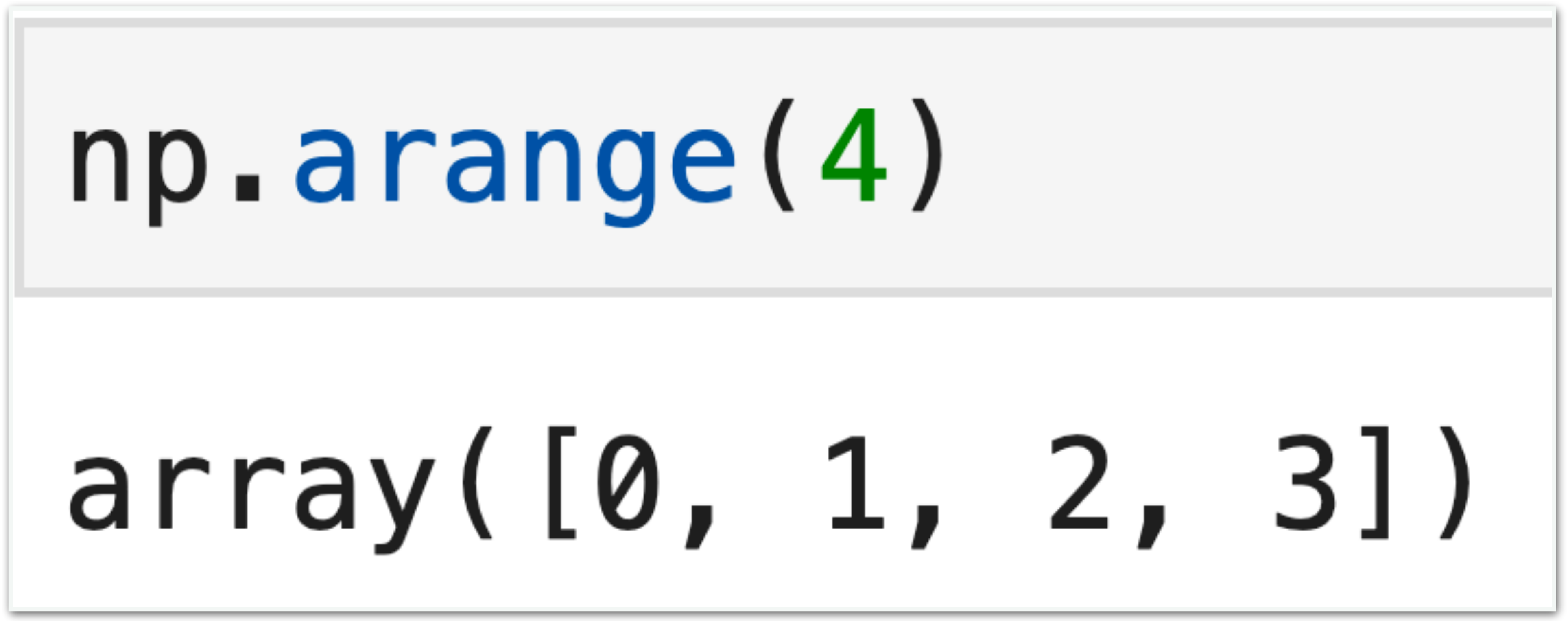
```
array([0, 1, 2, 3])
```



`for` Example



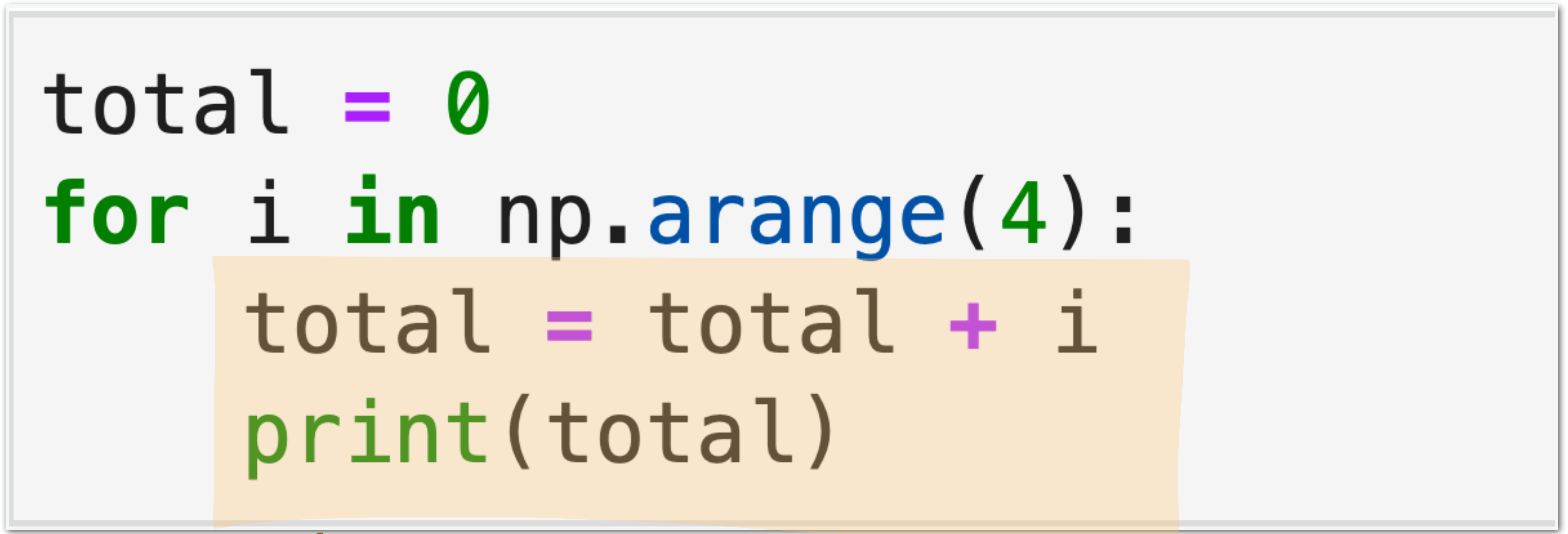
```python
total = 0
for i in np.arange(4):
    total = total + i
    print(total)
```



```python
np.arange(4)
```

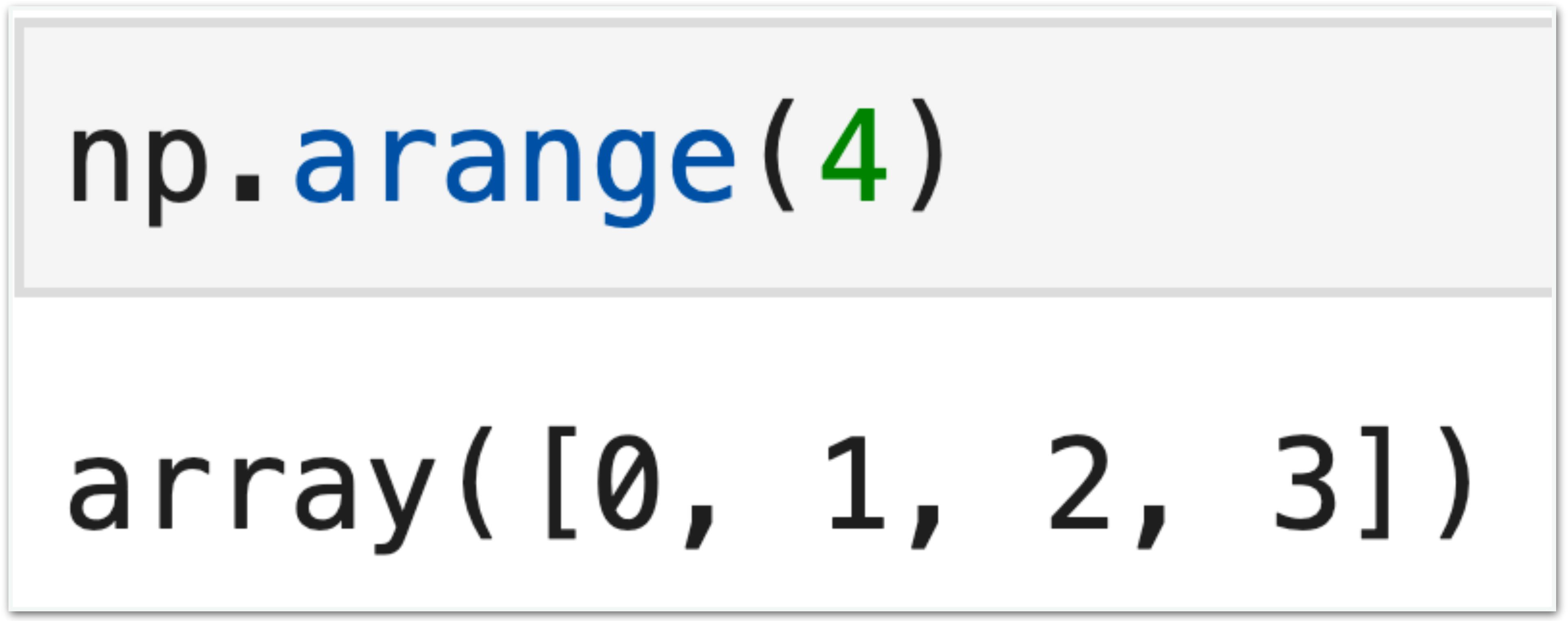
```
array([0, 1, 2, 3])
```

for Example



```
total = 0
for i in np.arange(4):
    total = total + i
    print(total)
```

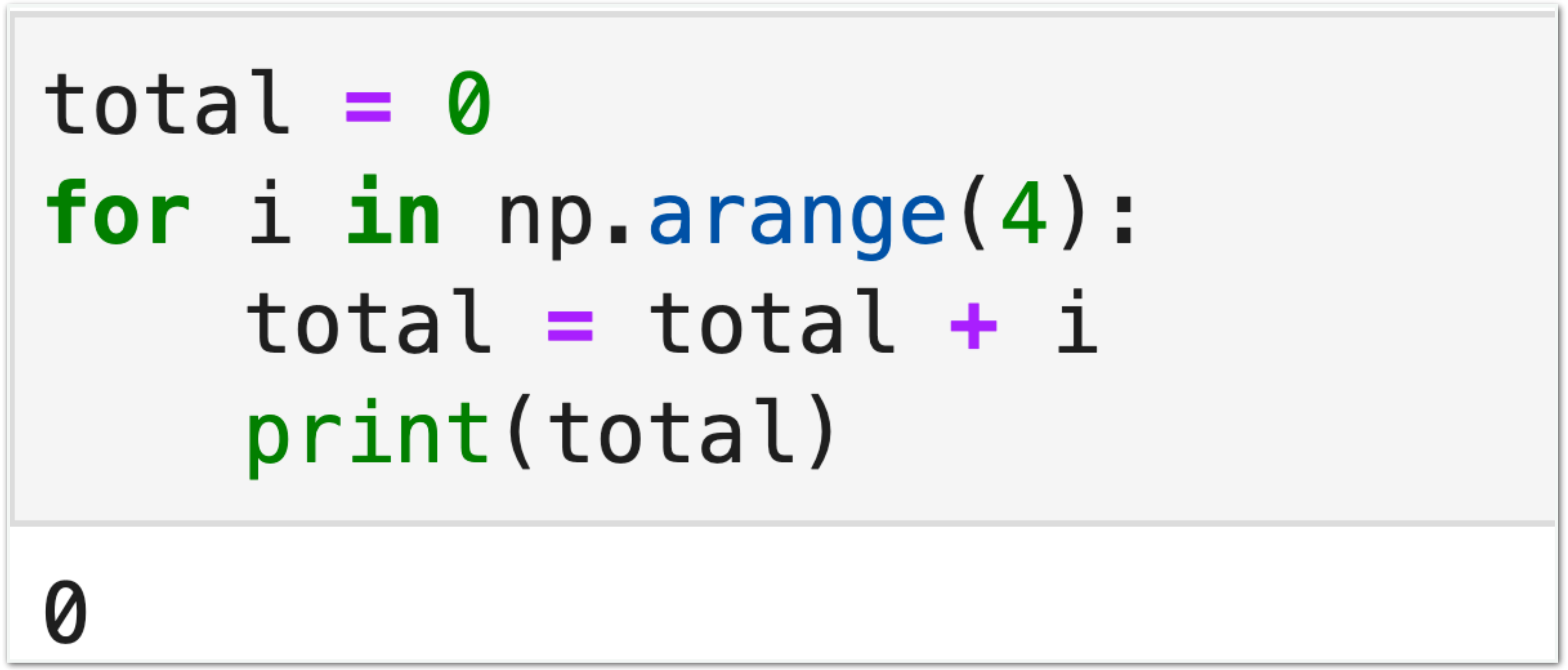
code to evaluate in each iteration of the loop



```
np.arange(4)
```

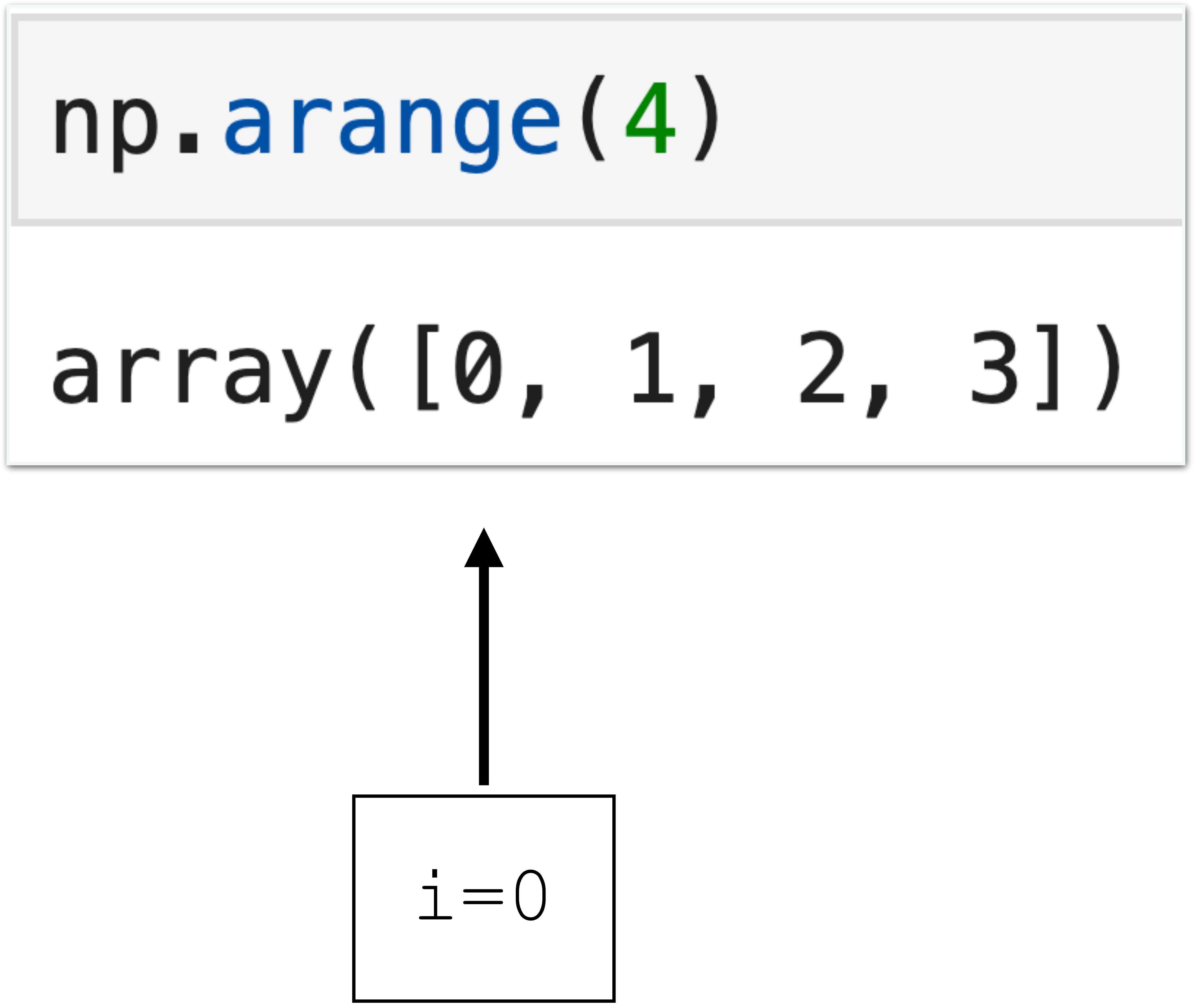
```
array([0, 1, 2, 3])
```

for Example



```python
total = 0
for i in np.arange(4):
    total = total + i
    print(total)
```

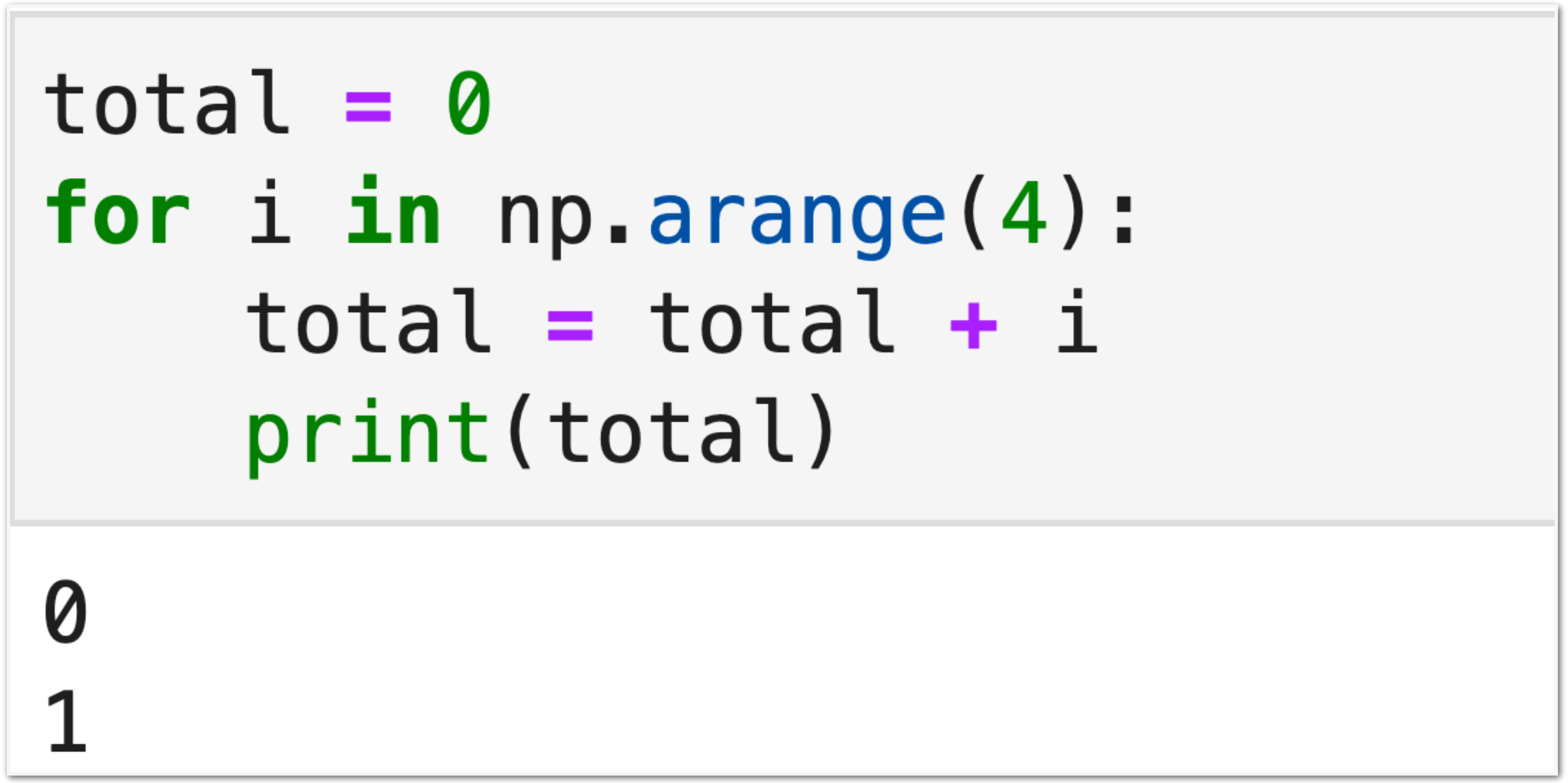
```
0
```



```python
np.arange(4)
```

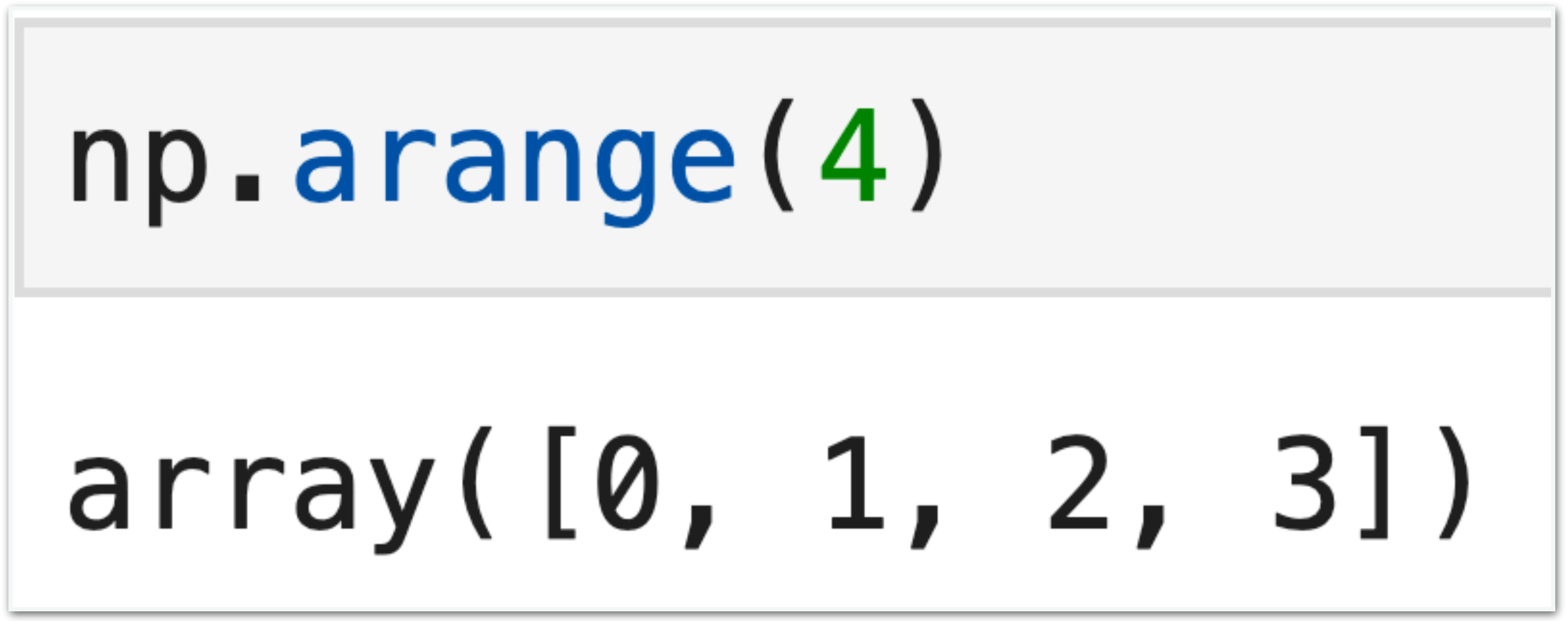
```
array([0, 1, 2, 3])
```

for Example



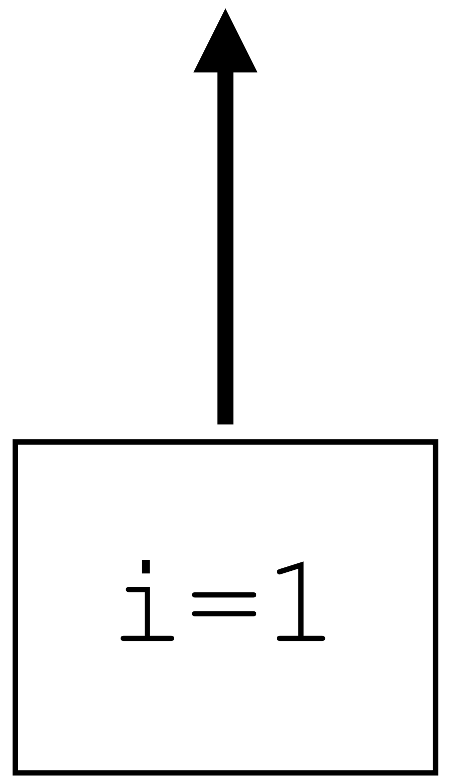
```
total = 0
for i in np.arange(4):
    total = total + i
    print(total)
```

```
0
1
```

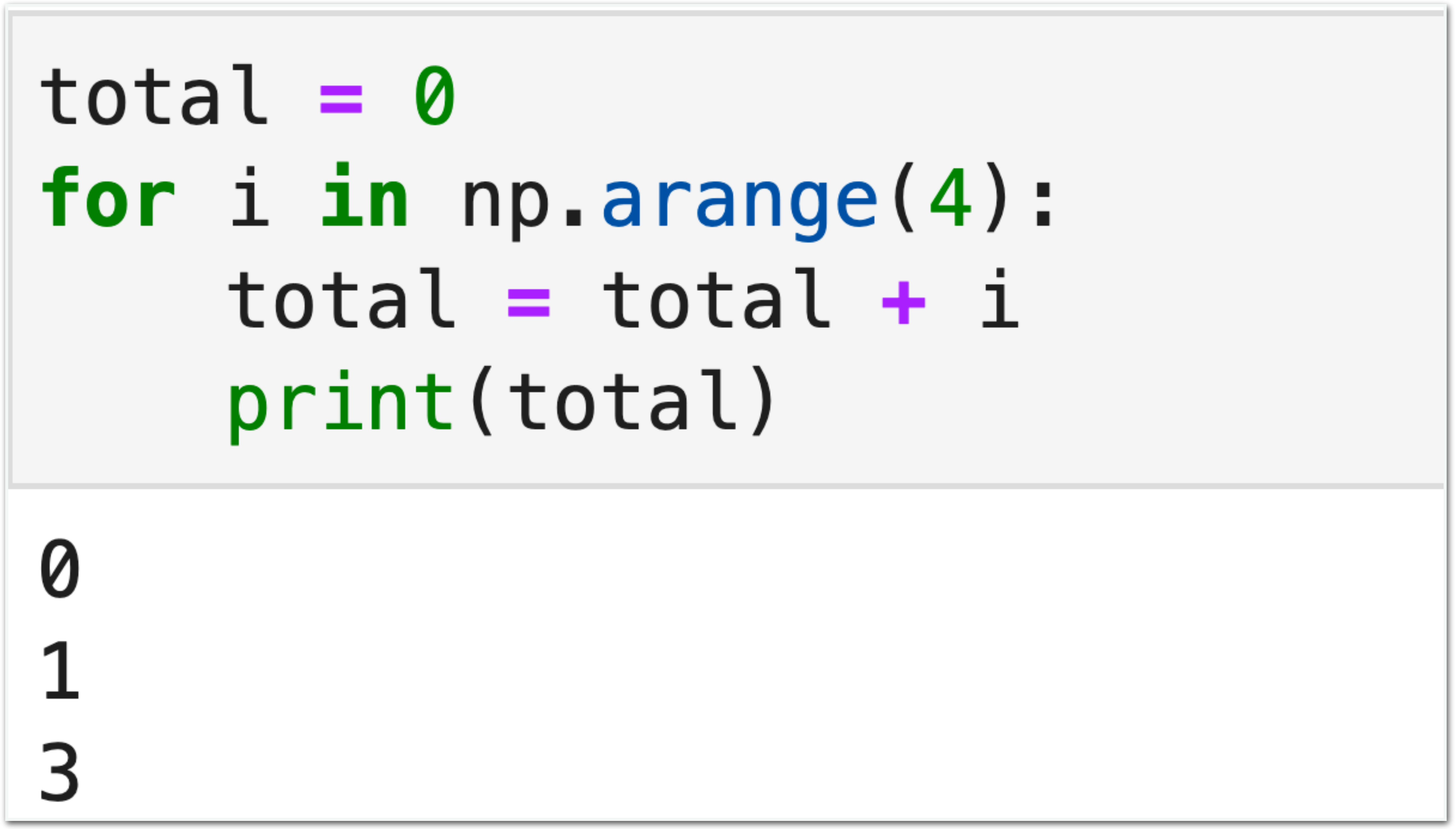


```
np.arange(4)
```

```
array([0, 1, 2, 3])
```

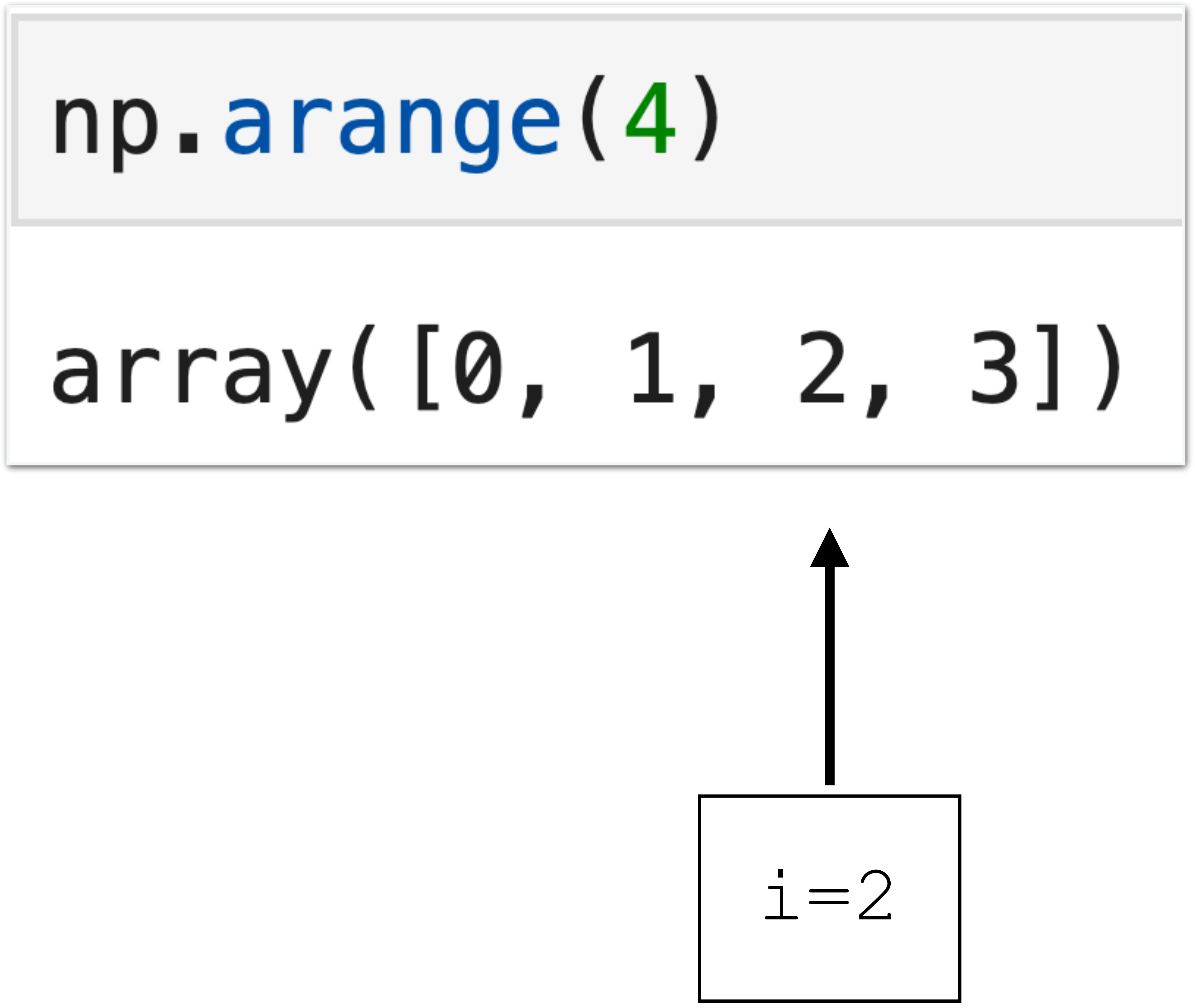


for Example



```
total = 0
for i in np.arange(4):
    total = total + i
    print(total)
```

```
0
1
3
```



```
np.arange(4)
```

```
array([0, 1, 2, 3])
```

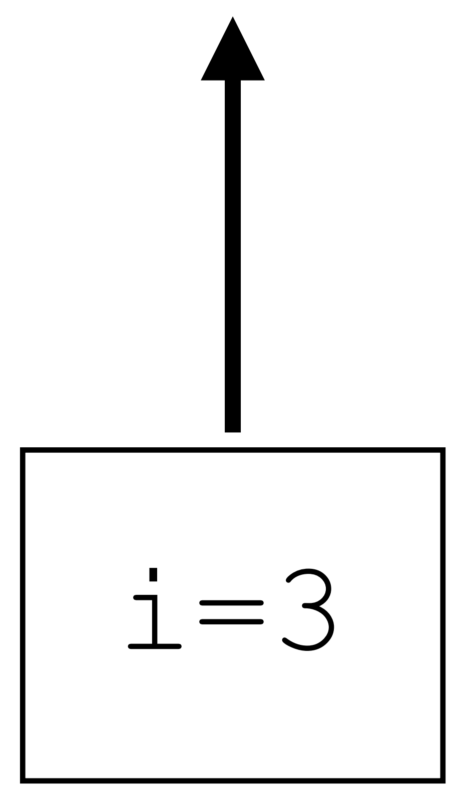
for Example

```python
total = 0
for i in np.arange(4):
    total = total + i
    print(total)
```

```
0
1
3
6
```

```python
np.arange(4)
```

```
array([0, 1, 2, 3])
```



Simulation

General Process for Simulations

1. Figure out what you want to simulate
    - Example: Outcomes of a coin toss

General Process for Simulations

1. Figure out what you want to simulate

   - Example: Outcomes of a coin toss

2. Write a function whose output is the outcome of a single simulation

General Process for Simulations

1. Figure out what you want to simulate
    - Example: Outcomes of a coin toss
2. Write a function whose output is the outcome of a single simulation
3. Repeat the simulation for some number of iterations
    - Keep track of the results of every iteration in an array

General Process for Simulations

1. Figure out what you want to simulate
    - Example: Outcomes of a coin toss
2. Write a function whose output is the outcome of a single simulation
3. Repeat the simulation for some number of iterations
    - Keep track of the results of every iteration in an array
4. Add results array to a table so you can plot the results

General Process for Simulations

1. Figure out what you want to simulate
    - Example: Outcomes of a coin toss
2. Write a function whose output is the outcome of a single simulation
3. Repeat the simulation for some number of iterations
    - Keep track of the results of every iteration in an array
4. Add results array to a table so you can plot the results

Typically what we want to know about is influenced by chance or randomness

Random Selection

```
import numpy as np
```

To select uniformly at random from array `some_array`

```
- np.random.choice(some_array)
```

To select `n` number of random elements from array `some_array`

```
- np.random.choice(some_array, n)
```

Note: Random does not mean arbitrary.

We mean each output has some chance of happening (probability)

General Process for Simulations

1. Figure out what you want to simulate
    - Example: Outcomes of a coin toss
2. Write a function whose output is the outcome of a single simulation
3. Repeat the simulation for some number of iterations
    - Keep track of the results of every iteration in an array
4. Add results array to a table so you can plot the results

To keep track of our results, we will want to append (add to the end) elements onto our arrays

Appending Arrays

```
import numpy as np
```

Return a copy of `array_1` where `value` is added onto the end

```
  np.append(array_1, value)
```

Returns an array with elements of `array_1` followed by elements of `array_2`

```
  np.append(array_1, array_2)
```

Next Class

- Today
  - Conditionals
  - Iteration
- Wednesday
  - Chance and Sampling