

# Python programming basics

Johannes Soltwedel

With material from

Robert Haase

Till Korten

# Who has programming experience?

---

I have never  
programmed

I have adjusted  
existing  
scripts/macros

I have written my  
own script

- Variables can hold numeric values and you can do math with them

```
# initialize parameters  
room_width = 5  
room_length = 6  
  
# run algorithm on given parameters  
room_area = room_width * room_length  
  
print(room_area)
```

30

- Also text (called strings) as values for variables are supported

Single and double quotes allowed

```
first_name = "Robert"  
last_name = 'Haase'  
  
print("Hello " + first_name + " " + last_name)
```

Hello Robert Haase

- String **f**ormatting is made easy using f-strings.

```
f"This is an f-string. a's value is {a}. Doubling the value of a gives {2*a}."
```

```
"This is an f-string. a's value is 5. Doubling the value of a gives 10."
```

**Comments** should contain additional information such as

- User documentation
  - What does the program do?
  - How can this program be used?
- Your name / institute in case a reader has a question
- Comment why things are done.
- Do not comment what is written in the code already!

```
#  
# This program sums up two numbers.  
#  
# Usage:  
# * Run it in Python 3.8  
#  
# Author: Robert Haase, PoL TUD  
#         Robert.haase@tu-dresden.de  
# April 2021  
  
# initialise program  
a = 1  
b = 2.5  
  
# run complicated algorithm  
final_result = a + b  
  
# print the final result  
print( final_result )
```

# Handling many items: lists

- Lists are variables, where you can store multiple values

Give me a "0", five times!

```
array = [0] * 5
```

Computer memory

array

1	0	5	0	Rab bit
---	---	---	---	------------



- Modifying lists entries

```
▶ numbers = [0, 1, 2, 3, 4]
# write in one array element
numbers[1] = 5
print(numbers)
[0, 5, 2, 3, 4]
```

Note: The first element has index 0!

- Creating lists of defined size

What? How many?

```
▶ zeros = [0] * 10
print(zeros)
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

- Concatenating lists

```
▶ ones = [1, 1, 1]
twos = [2, 2, 2, 2]
# concatenate arrays
numbers = ones + twos
print(numbers)
[1, 1, 1, 2, 2, 2, 2]
```

+ means appending

```
▶ # Arrays  
numbers = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]  
print(numbers)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

- Creating subsets of lists

Start

End

```
▶ subset = numbers[2:4]  
print(subset)
```

```
[2, 3]
```

Step

```
▶ subset_with_gaps = arr[1:8:2]  
print(subset_with_gaps)
```

```
[1, 3, 5, 7]
```

**data [start : stop : step]**

- “Indexing” is addressing certain elements in lists. The first element is “0” away from the start.

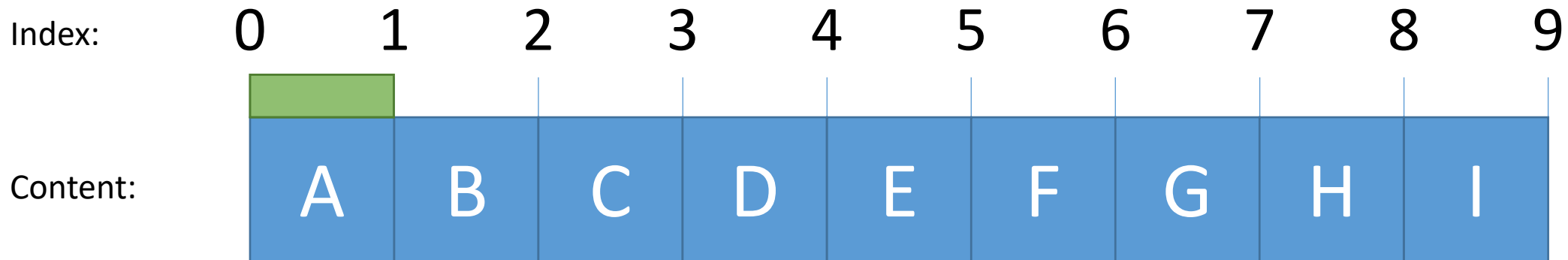
```
data = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I']
```

Index:	0	1	2	3	4	5	6	7	8	9
Content:	A	B	C	D	E	F	G	H	I	

# Indexing, cropping, subsets

- “Indexing” is addressing certain elements in lists. The first element is “0” away from the start.

```
data = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I']
```



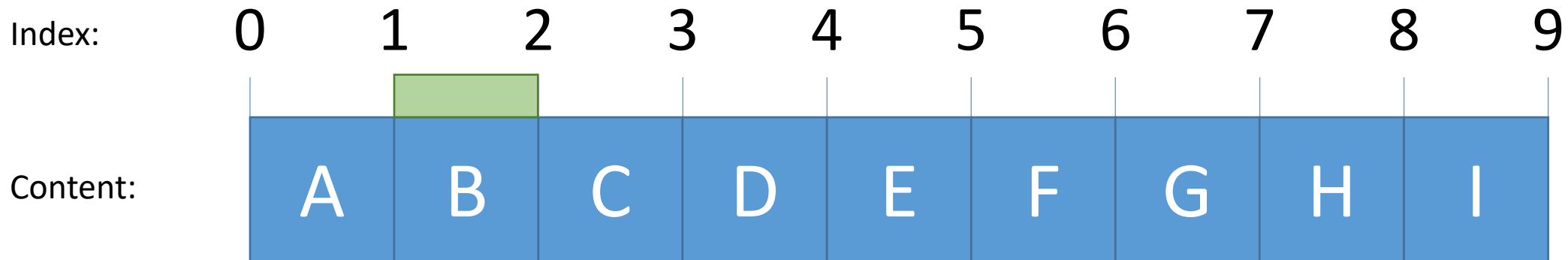
```
data[0]
```

'A'

# Indexing, cropping, subsets

- “Indexing” is addressing certain elements in lists. The first element is “0” away from the start.

```
data = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I']
```



```
data[0]
```

'A'

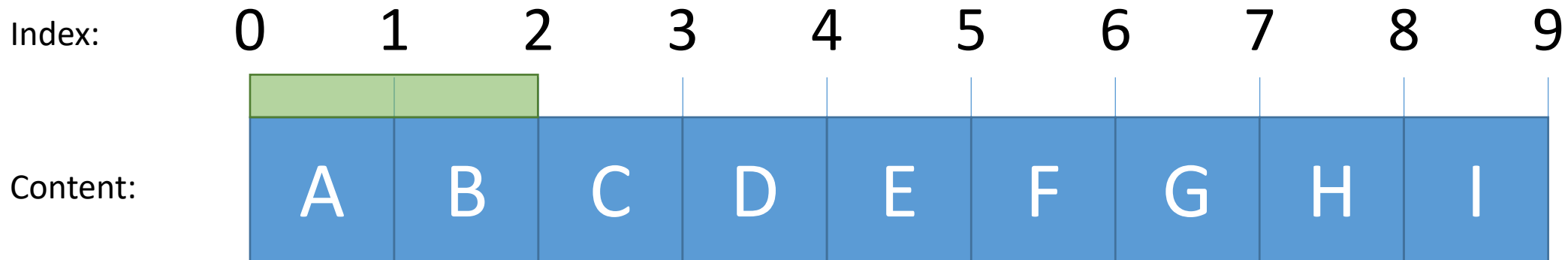
```
data[1]
```

'B'

# Indexing, cropping, subsets

- “Indexing” is addressing certain elements in lists. The first element is “0” away from the start.

```
data = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I']
```



```
data[0]
```

'A'

```
data[1]
```

'B'

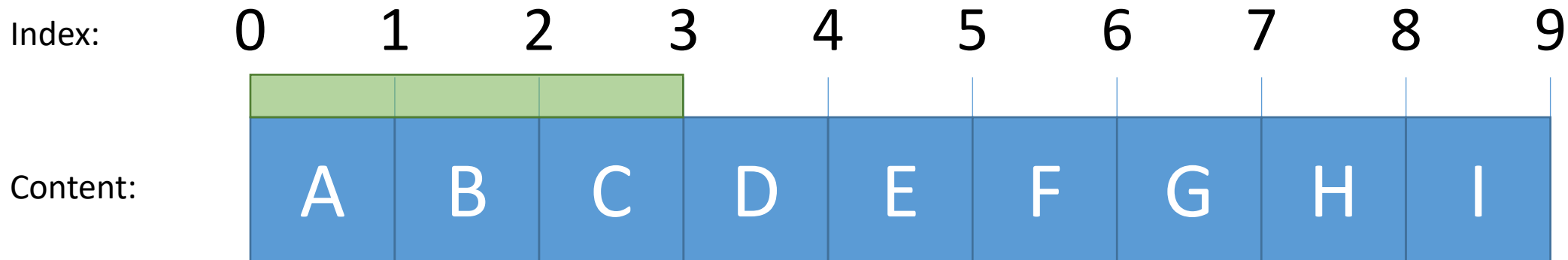
```
data[0:2]
```

['A', 'B']

# Indexing, cropping, subsets

- “Indexing” is addressing certain elements in lists. The first element is “0” away from the start.

```
data = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I']
```



```
data[0]
```

```
'A'
```

```
data[1]
```

```
'B'
```

```
data[0:2]
```

```
['A', 'B']
```

```
data[0:3]
```

```
['A', 'B', 'C']
```

```
data[1:2]
```

```
['B']
```

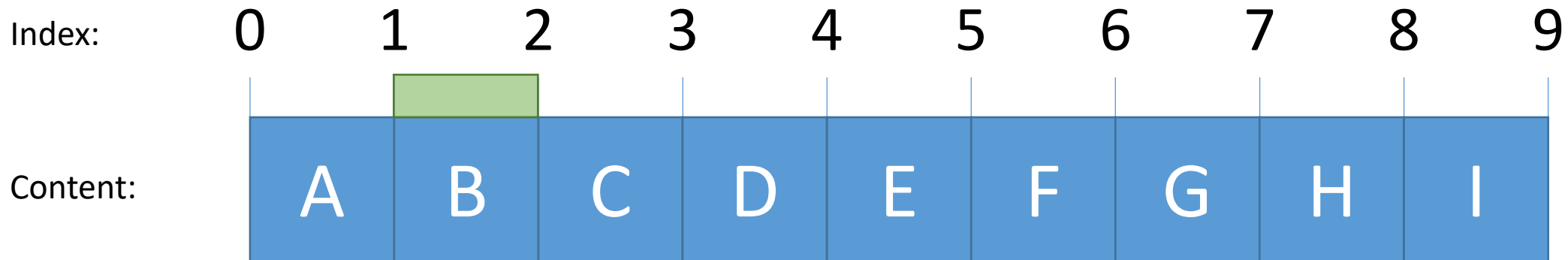
```
len(data)
```

```
9
```

# Indexing, cropping, subsets

- “Indexing” is addressing certain elements in lists. The first element is “0” away from the start.

```
data = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I']
```



```
data[0]
```

```
'A'
```

```
data[1]
```

```
'B'
```

```
data[0:2]
```

```
['A', 'B']
```

```
data[0:3]
```

```
['A', 'B', 'C']
```

```
data[1:2]
```

```
['B']
```



# Indexing, cropping, subsets

- “Indexing” is addressing certain elements in lists. The first element is “0” away from the start.

```
data = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I']
```

Index:	0	1	2	3	4	5	6	7	8	9
Content:	A	B	C	D	E	F	G	H	I	

```
data[0]
```

```
'A'
```

```
data[1]
```

```
'B'
```

```
data[0:2]
```

```
['A', 'B']
```

```
data[0:3]
```

```
['A', 'B', 'C']
```

```
data[1:2]
```

```
['B']
```

```
len(data)
```

```
9
```

- You can leave start and end out when specifying index ranges

```
data = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I']
```

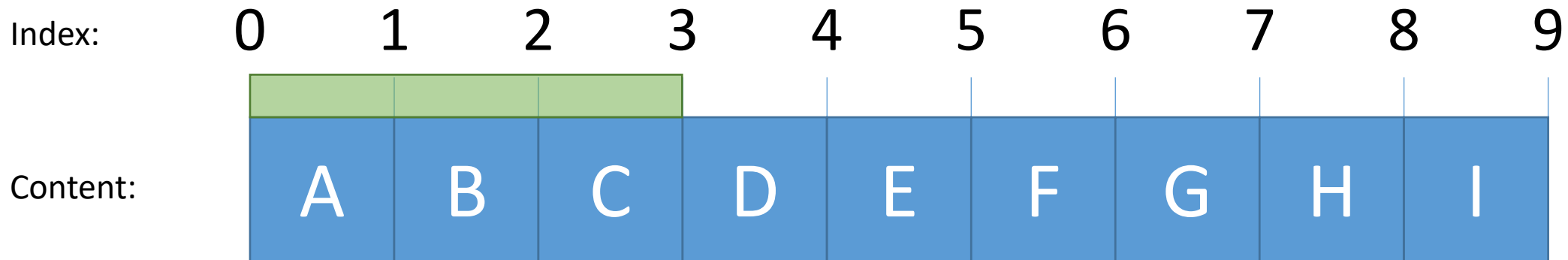
Index:	0	1	2	3	4	5	6	7	8	9
Content:	A	B	C	D	E	F	G	H	I	

```
data[:2]
```

```
['A', 'B']
```

- You can leave start and end out when specifying index ranges

```
data = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I']
```



```
data[:2]
```

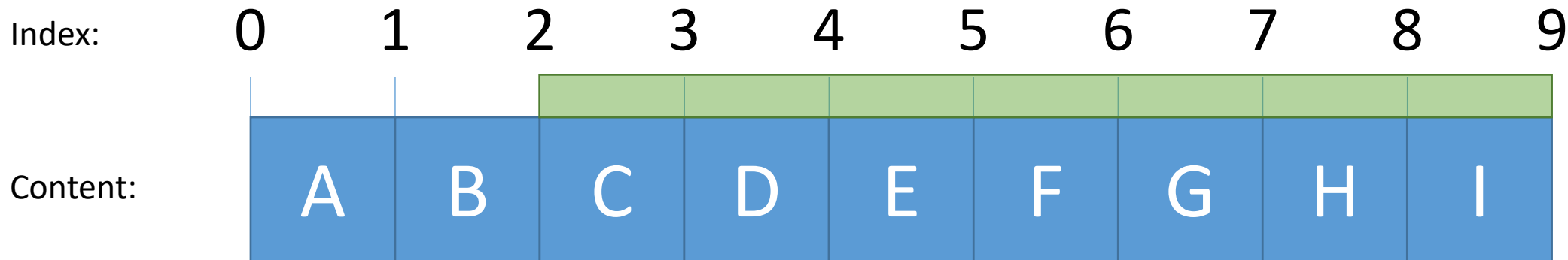
```
['A', 'B']
```

```
data[:3]
```

```
['A', 'B', 'C']
```

- You can leave start and end out when specifying index ranges

```
data = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I']
```



```
data[:2]
```

```
['A', 'B']
```

```
data[:3]
```

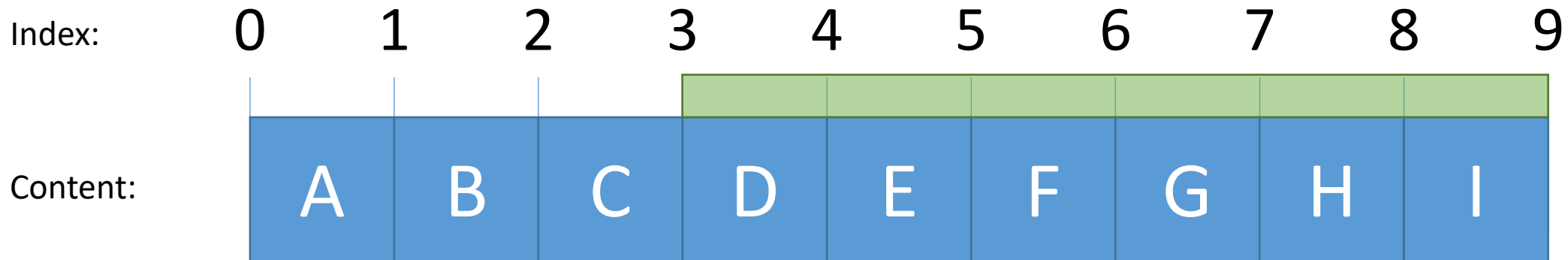
```
['A', 'B', 'C']
```

```
data[2:]
```

```
['C', 'D', 'E', 'F', 'G', 'H', 'I']
```

- You can leave start and end out when specifying index ranges

```
data = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I']
```



```
data[:2]
```

```
['A', 'B']
```

```
data[:3]
```

```
['A', 'B', 'C']
```

```
data[2:]
```

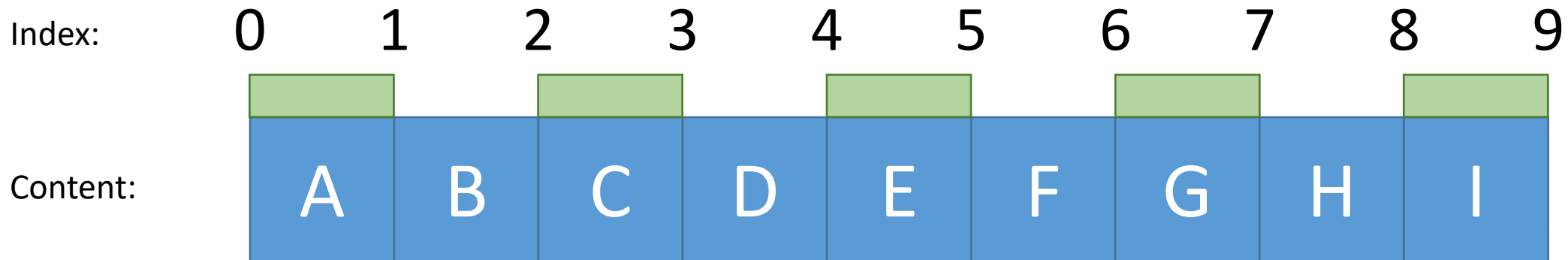
```
['C', 'D', 'E', 'F', 'G', 'H', 'I']
```

```
data[3:]
```

```
['D', 'E', 'F', 'G', 'H', 'I']
```

- The step-size allows skipping elements

```
data = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I']
```

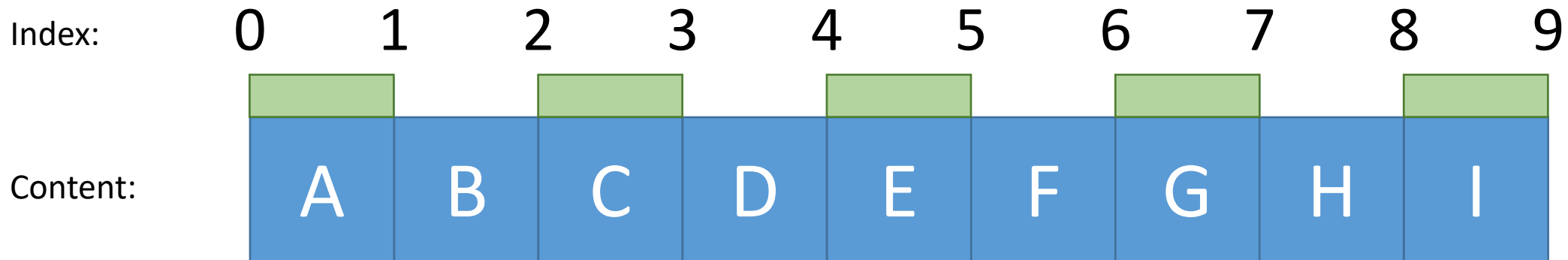


```
data[0:10:2]
```

```
['A', 'C', 'E', 'G', 'I']
```

- The step-size allows skipping elements

```
data = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I']
```



```
data[0:10:2]
```

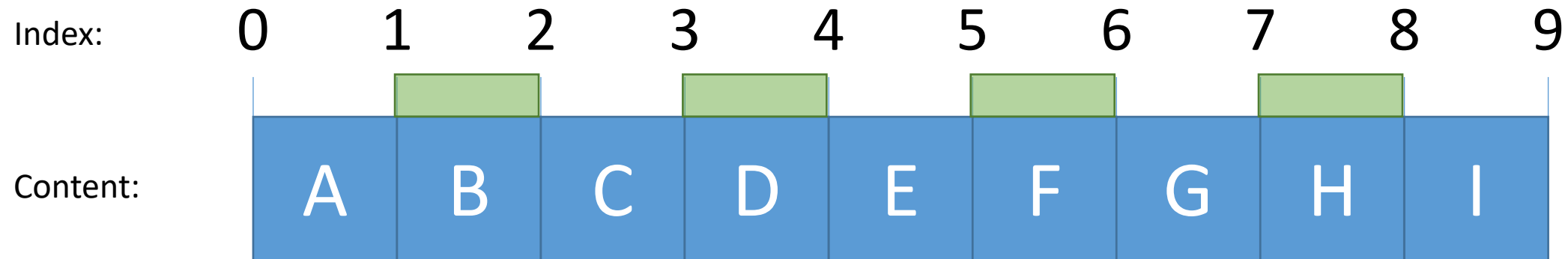
```
['A', 'C', 'E', 'G', 'I']
```

```
data[::2]
```

```
['A', 'C', 'E', 'G', 'I']
```

- The step-size allows skipping elements

```
data = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I']
```



```
data[0:10:2]
```

```
data[::2]
```

```
data[1::2]
```

```
['A', 'C', 'E', 'G', 'I']
```

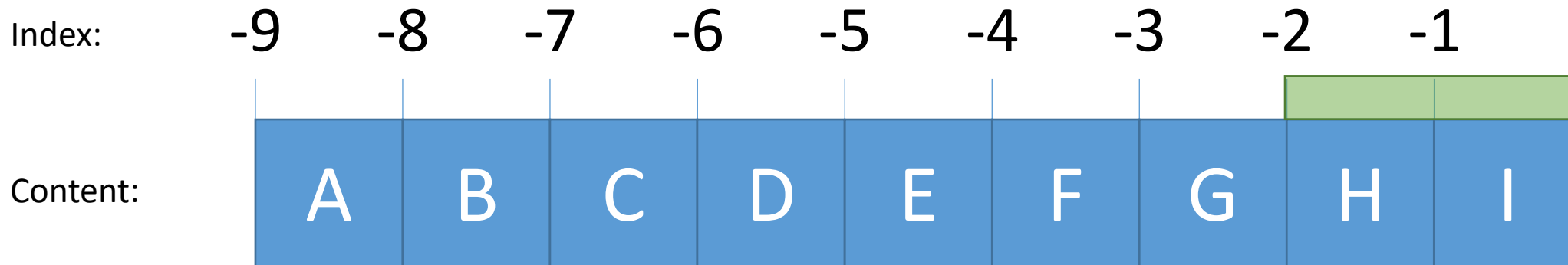
```
['A', 'C', 'E', 'G', 'I']
```

```
['B', 'D', 'F', 'H']
```



- Indexing also works with negative indices

```
data = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I']
```

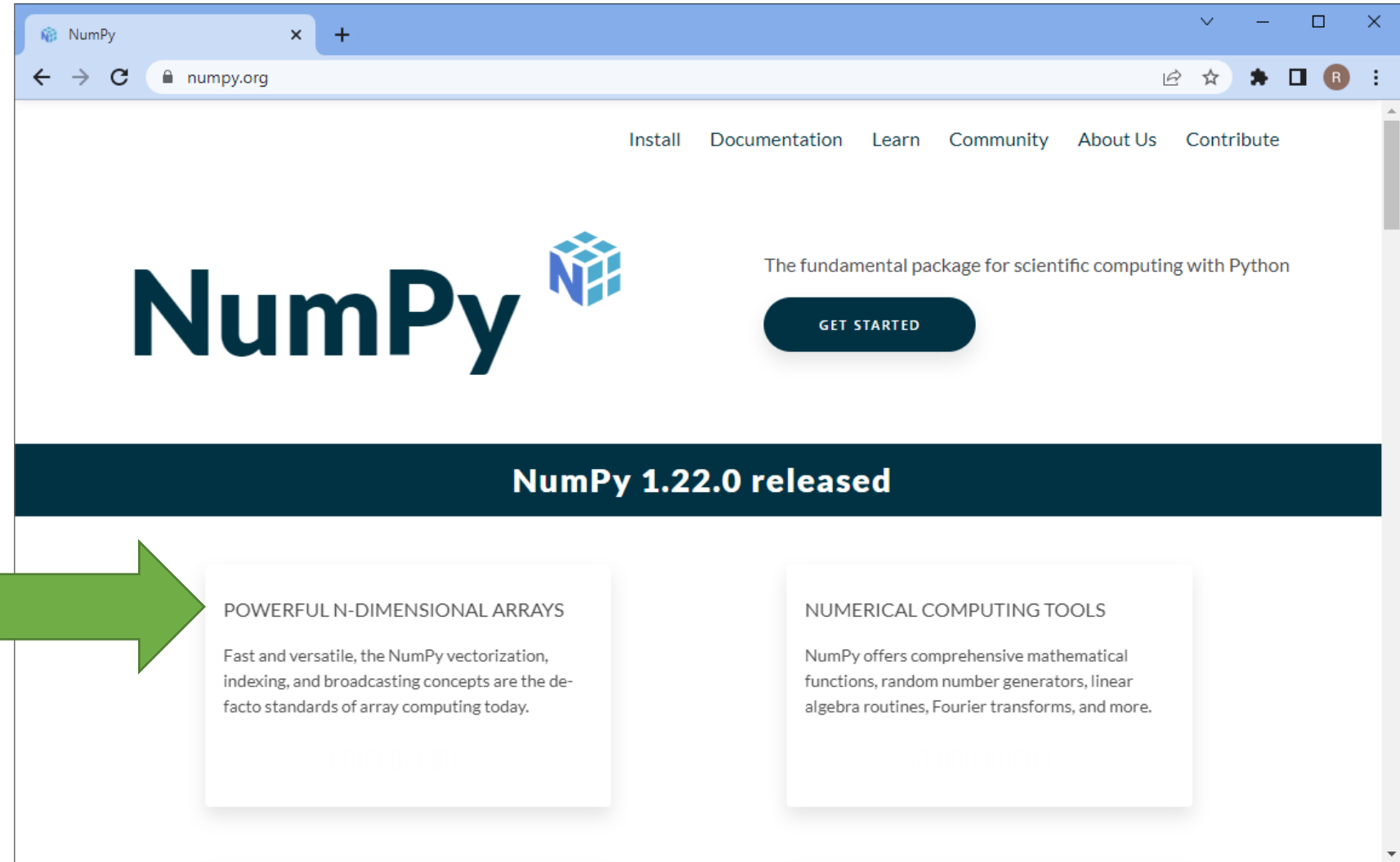


```
data[-2:]
```

```
['H', 'I']
```

- The fundamental package for scientific computing with python.

- `conda install numpy`



- Simplifying mathematical operations on n-dimensional arrays

Tell python that you want to use a library called numpy

- Python arrays of arrays (lists of lists)

```
▶ # multidimensional arrays  
matrix = [  
    [1, 2, 3],  
    [2, 3, 4],  
    [3, 4, 5]  
]  
  
print(matrix)
```

```
[[1, 2, 3], [2, 3, 4], [3, 4, 5]]
```

```
▶ result = matrix * 2  
print(result)
```

```
[[1, 2, 3], [2, 3, 4], [3, 4, 5], [1, 2, 3], [2, 3, 4], [3, 4, 5]]
```

- numpy arrays

```
▶ import numpy as np  
np_matrix = np.asarray(matrix)  
print(np_matrix)
```

If "numpy" is too long, you can give an alias "np"

```
[[1 2 3]  
 [2 3 4]  
 [3 4 5]]
```

```
▶ np_result = np_matrix * 2  
print(np_result)
```

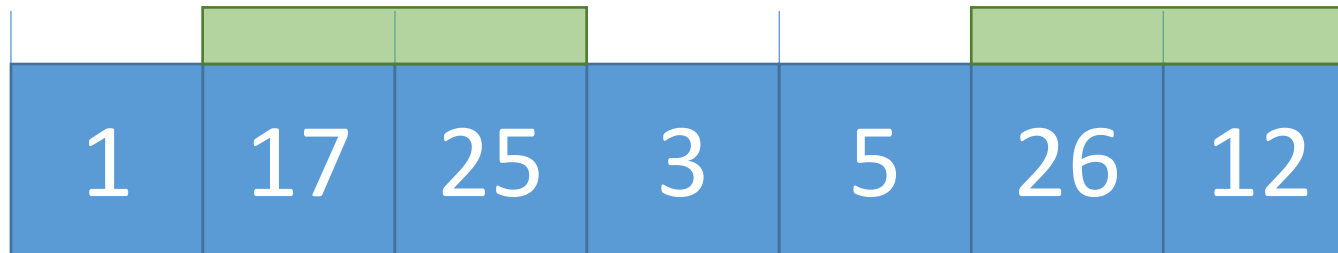
```
[[ 2  4  6]  
 [ 4  6  8]  
 [ 6  8 10]]
```

- “Masking” is addressing certain elements in numpy arrays, e.g. depending on their content

```
import numpy
measurements = numpy.asarray([1, 17, 25, 3, 5, 26, 12])
measurements
```

```
array([ 1, 17, 25,  3,  5, 26, 12])
```

Content:



```
mask = measurements > 10
mask
```

```
array([False,  True,  True, False, False,  True,  True])
```

```
measurements[mask]
```

```
array([17, 25, 26, 12])
```

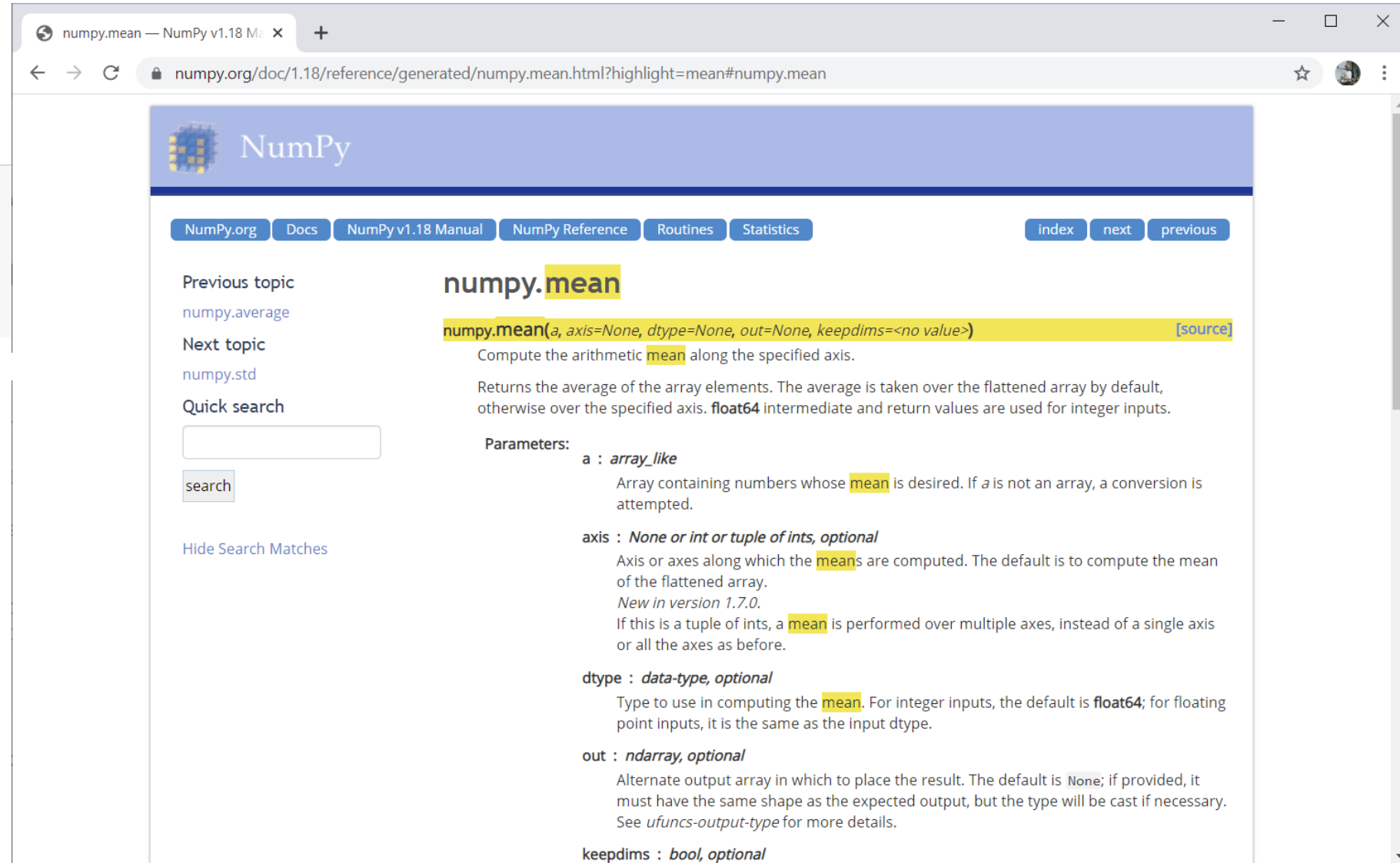
- Basic descriptive statistics

```
import numpy as np

measurements = [1, 4, 6, 7, 2]

mean = np.mean(measurements)
print("Mean: " + str(mean))

Mean: 4.0
```



The screenshot shows a web browser displaying the NumPy documentation for the `numpy.mean` function. The browser's address bar shows the URL: `numpy.org/doc/1.18/reference/generated/numpy.mean.html?highlight=mean#numpy.mean`. The page header includes the NumPy logo and navigation links for `NumPy.org`, `Docs`, `NumPy v1.18 Manual`, `NumPy Reference`, `Routines`, and `Statistics`. There are also buttons for `index`, `next`, and `previous`. The main content area features the title `numpy.mean` and the function signature: `numpy.mean(a, axis=None, dtype=None, out=None, keepdims=<no value>)` with a `[source]` link. Below the signature, a brief description states: "Compute the arithmetic mean along the specified axis." The text continues: "Returns the average of the array elements. The average is taken over the flattened array by default, otherwise over the specified axis. float64 intermediate and return values are used for integer inputs." The `Parameters:` section lists: 

- `a` : *array\_like* - Array containing numbers whose mean is desired. If `a` is not an array, a conversion is attempted.
- `axis` : *None or int or tuple of ints, optional* - Axis or axes along which the means are computed. The default is to compute the mean of the flattened array. *New in version 1.7.0.* If this is a tuple of ints, a mean is performed over multiple axes, instead of a single axis or all the axes as before.
- `dtype` : *data-type, optional* - Type to use in computing the mean. For integer inputs, the default is `float64`; for floating point inputs, it is the same as the input dtype.
- `out` : *ndarray, optional* - Alternate output array in which to place the result. The default is `None`; if provided, it must have the same shape as the expected output, but the type will be cast if necessary. See *ufuncs-output-type* for more details.
- `keepdims` : *bool, optional*