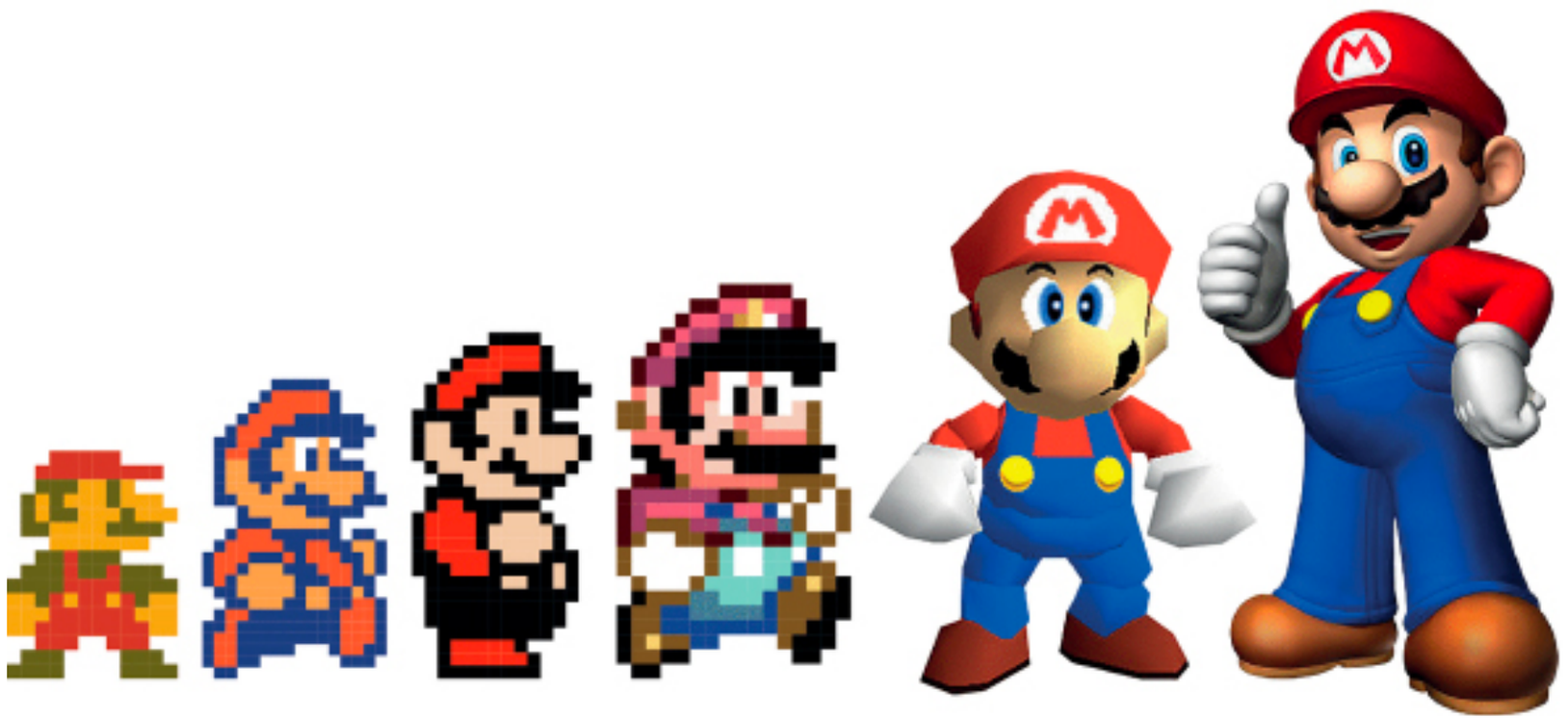
Function, Modules and Packages

Contents

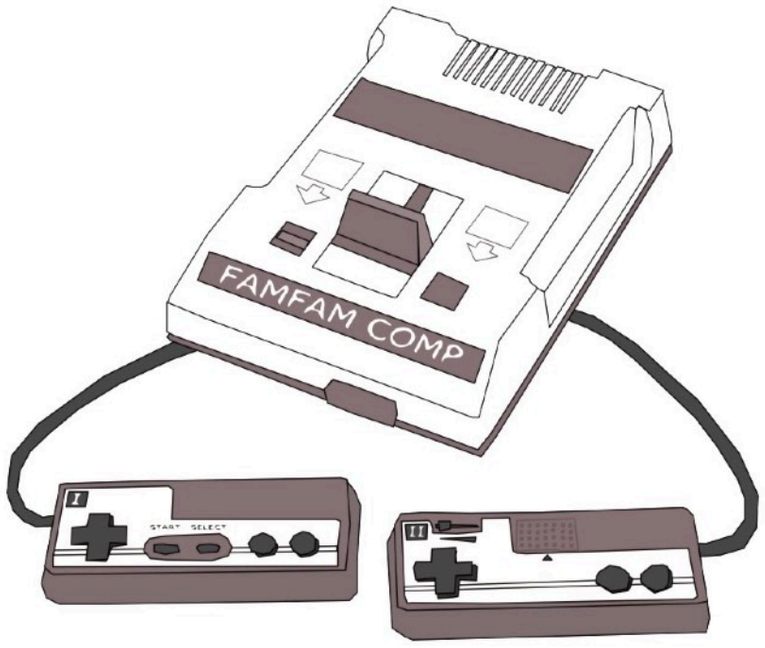
Contents

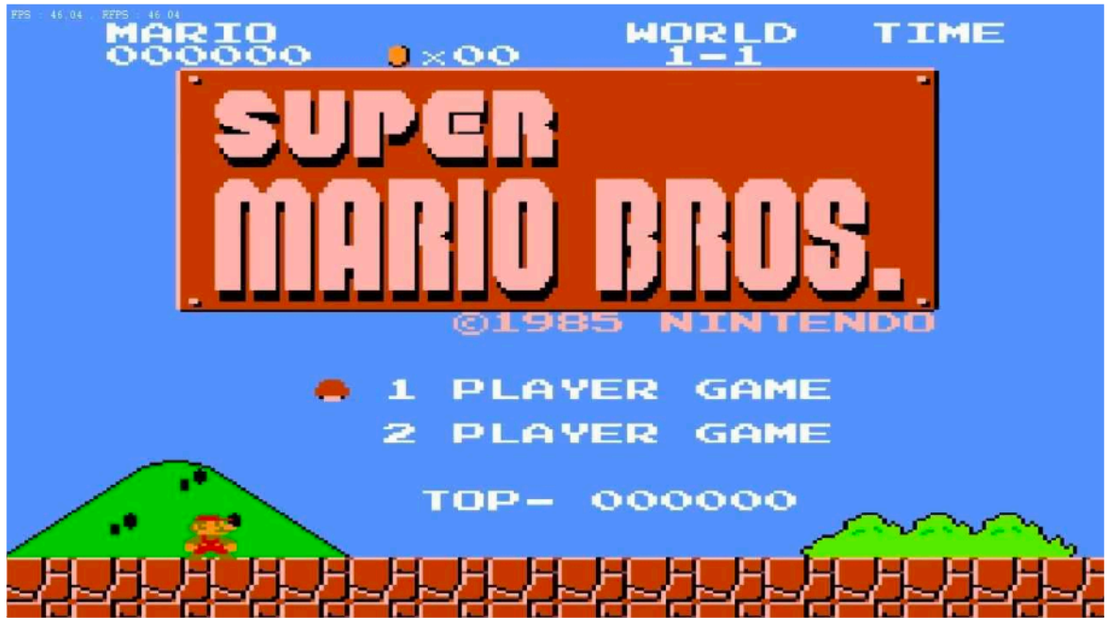
Functions



Functions

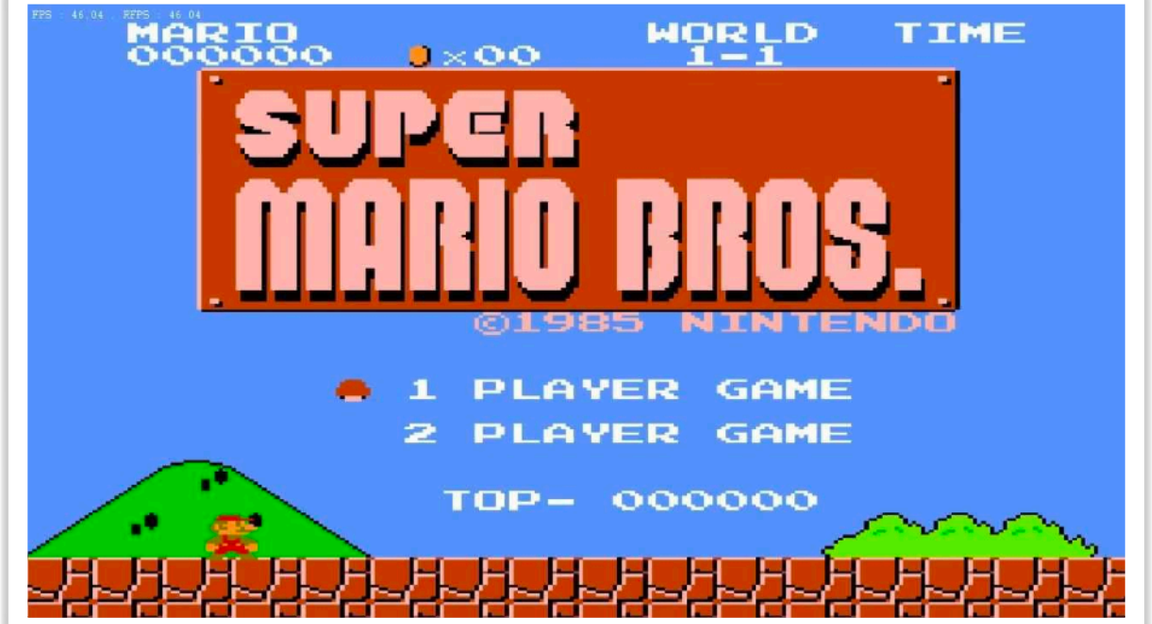
- Super Mario Bro.
  - A Nintendo game released in 1985
  - Incredibly small





Functions

- Super Mario Bro.
  - A Nintendo game released in 1985
  - Incredibly small: 32KB



Functions

- Super Mario Bro.
  - A Nintendo game released in 1985
  - Incredibly small: 32KB
  - Reuse of textures
    - ✓ Clouds and bushes are the same image with different colors and locations
    - ✓ The same code could be used to generate two different objects

Functions

- Super Mario Bro.
  - Reuse of textures

Functions

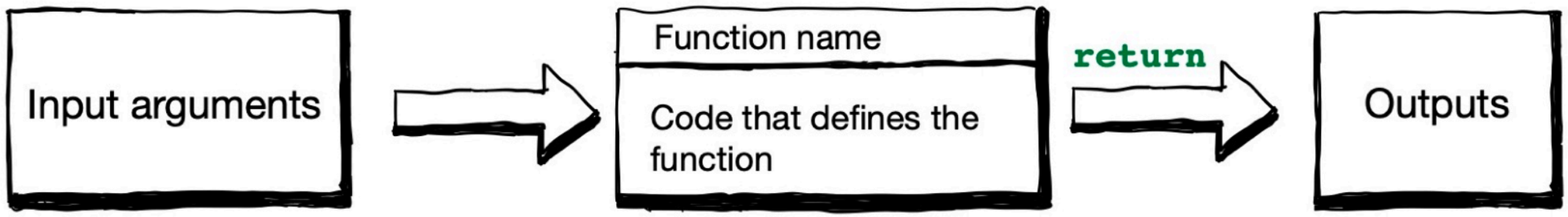
- Super Mario Bro.
  - Reuse of textures

Functions

- Super Mario Bro.
  - Reuse of textures

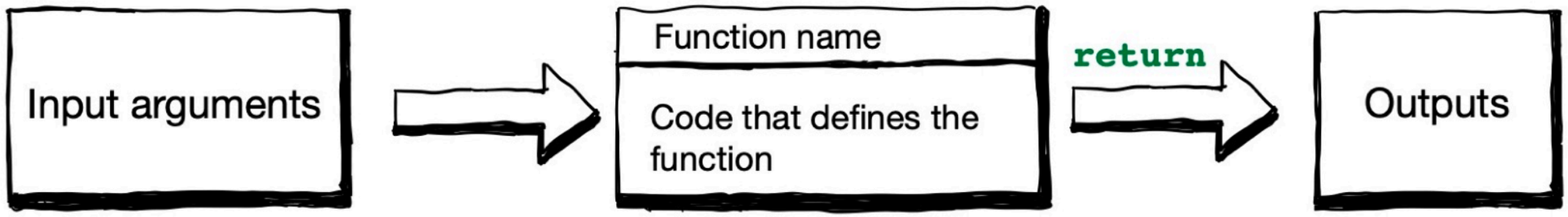
Functions

- Super Mario Bro.
  - General idea of calling functions



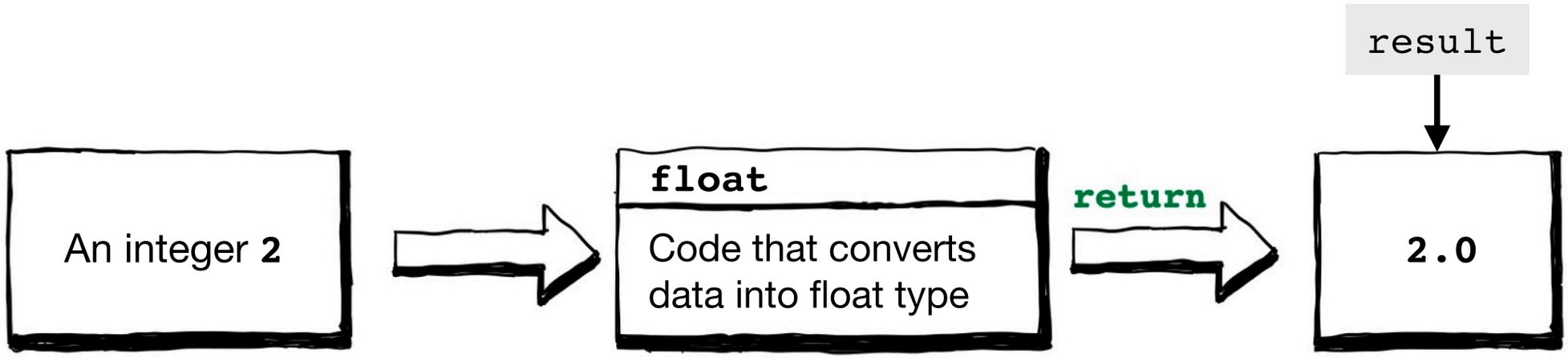
Functions

- Review of some built-in functions



Functions

- Review of some built-in functions
  - Functions that return outputs

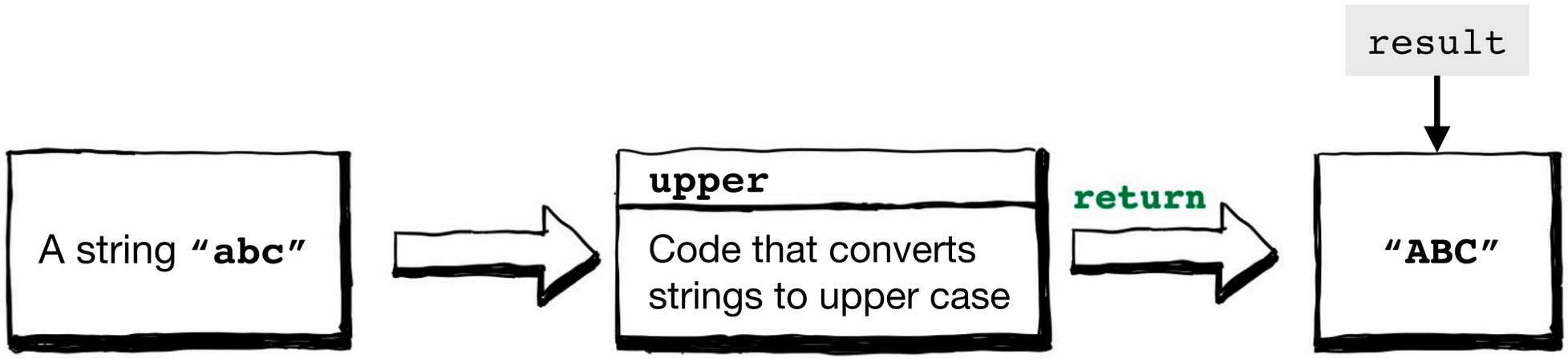



```
result = float(2)        # Function float returns 2.0 as the output
print(result)
```

```
2.0
```

Functions

- Review of some built-in functions
  - Functions that return outputs

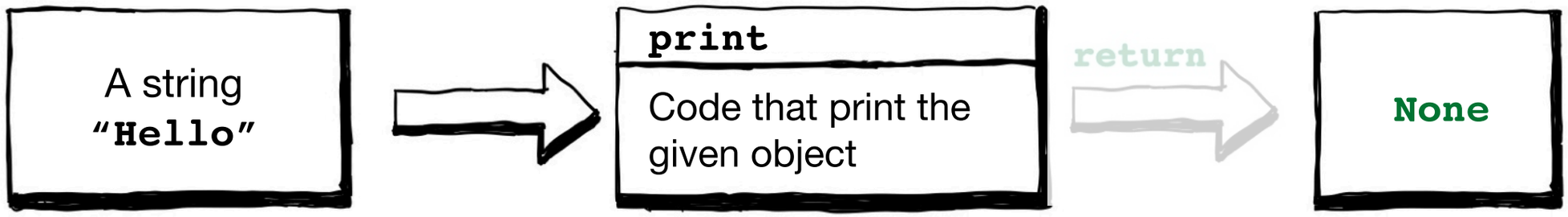



```
result = 'abc'.upper()  # Method upper returns 'ABC' as the output
print(result)
```

```
ABC
```

Functions

- Review of some built-in functions
  - Functions that return **no** output

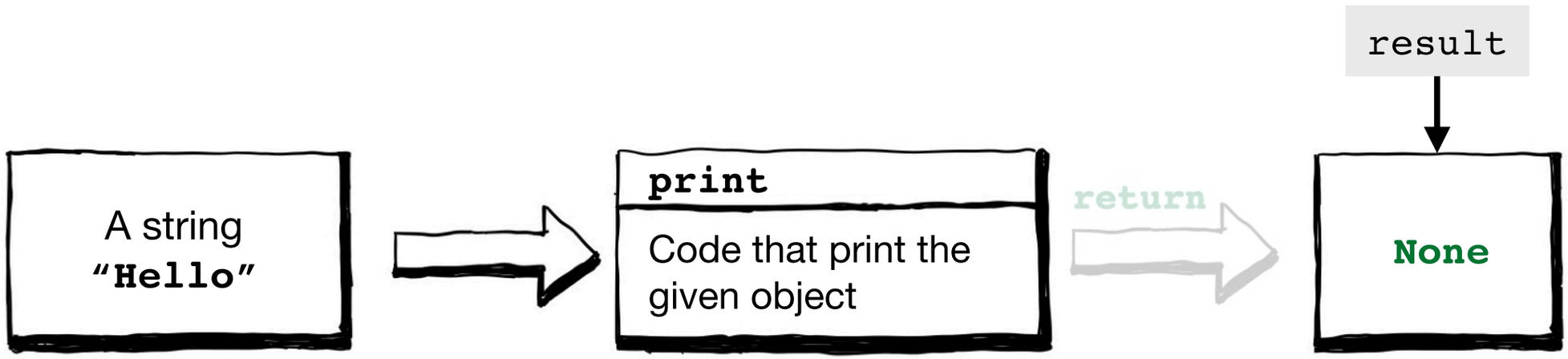



```
print('Hello!')
```

```
Hello!
```

Functions

- Review of some built-in functions
  - Functions that return **no** output

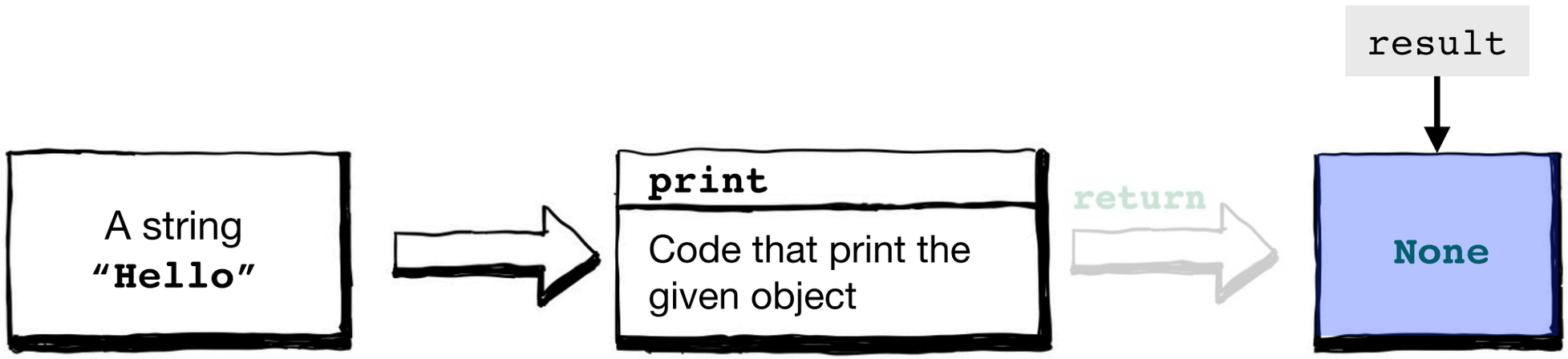



```
result = print('What?')

print(result)
print(type(result))
```

```
What?
None
<class 'NoneType'>
```

Functions

- Review of some built-in functions
  - Functions that return **no** output

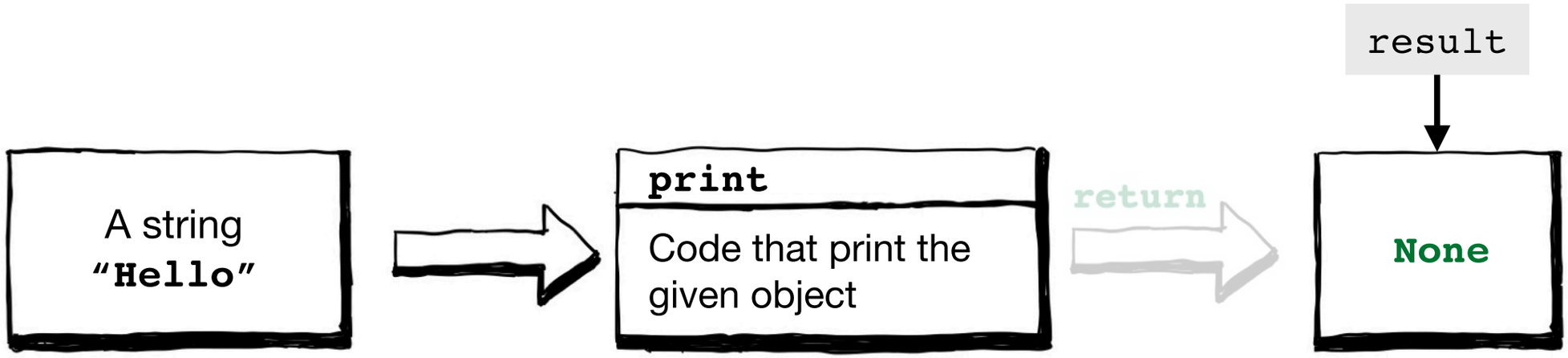



```
result = print('What?')

print(result)
print(type(result))
```

```
What?
None
<class 'NoneType'>
```

Functions

- Review of some built-in functions
  - Functions that return **no** output

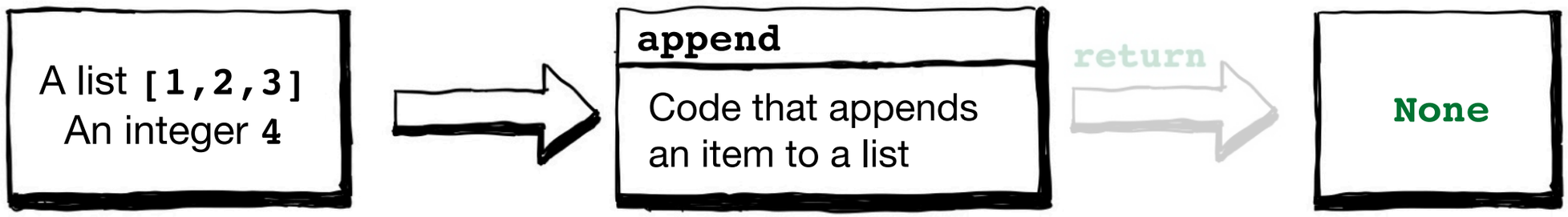



```
result = print('What?')

print(result)
print(type(result))
```

```
What?
None
<class 'NoneType'>
```

Functions

- Review of some built-in functions
  - Functions that return **no** output

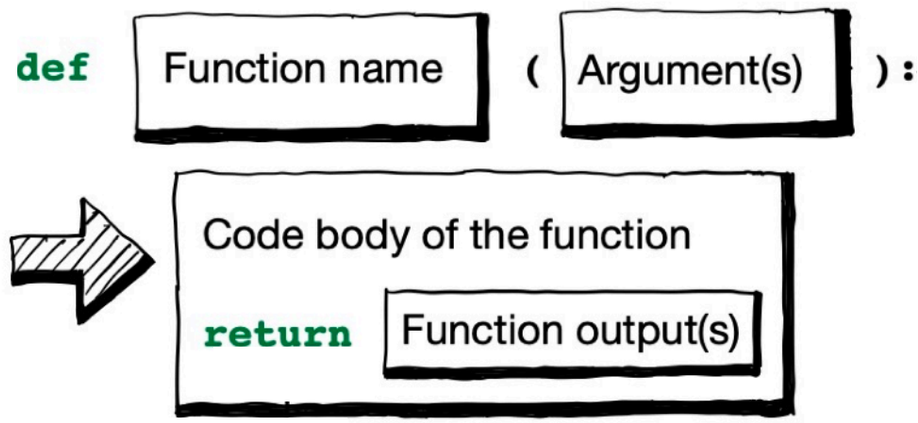



```
x = [1, 2, 3]
x.append(4)
print(x)
```

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

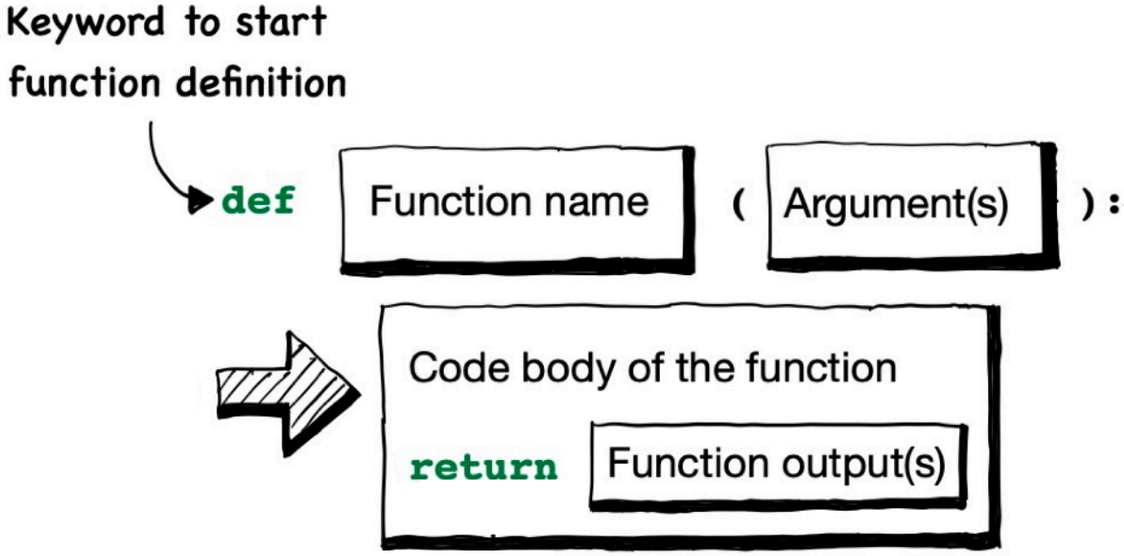
Functions

- Create a function
  - Syntax of function definition



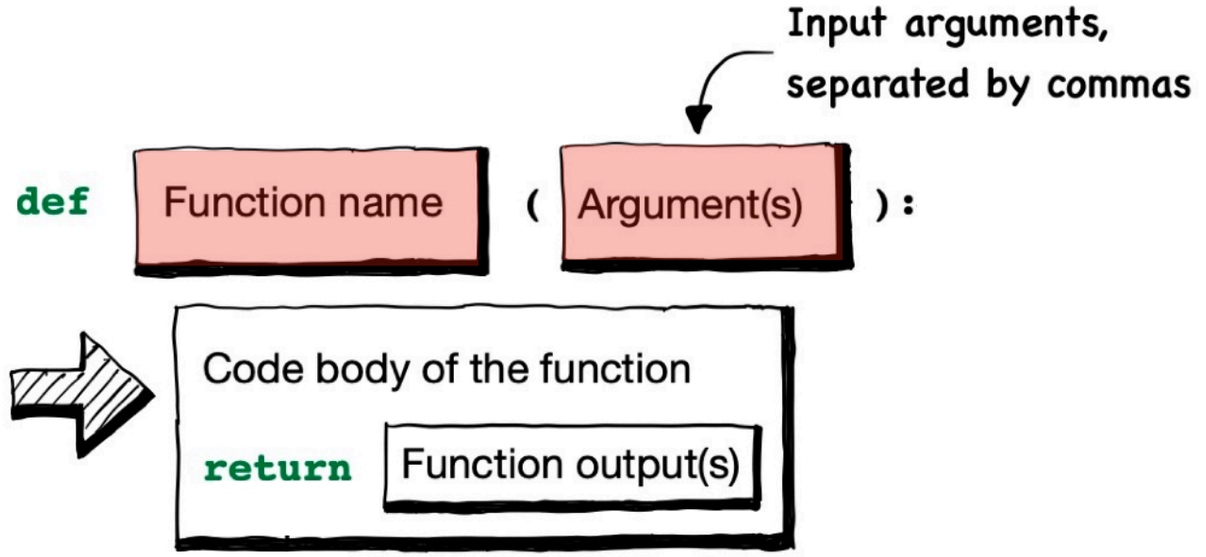
Functions

- Create a function
  - Syntax of function definition
    - ✓ Keyword `def`



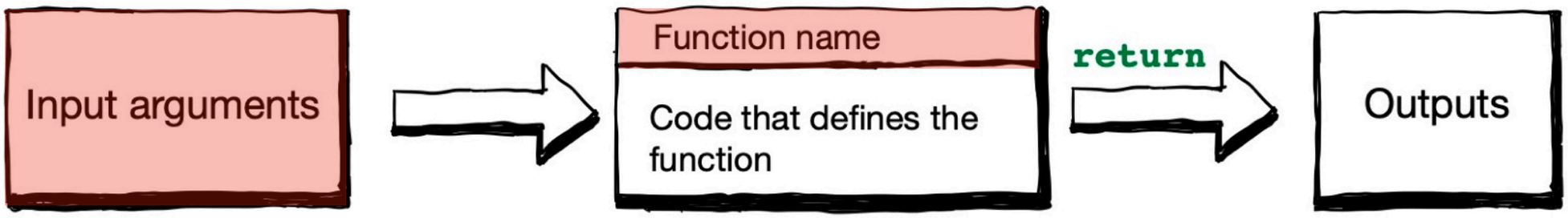
Functions

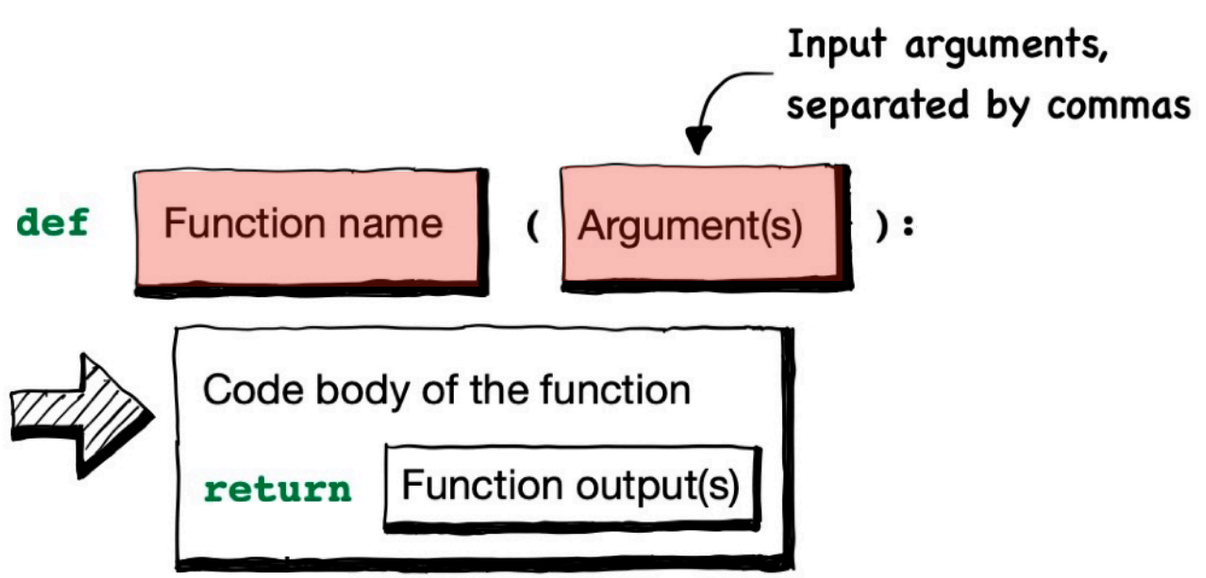
- Create a function
  - Syntax of function definition
    - ✓ A function name followed by input argument(s) enclosed in parentheses



Functions

- Create a function



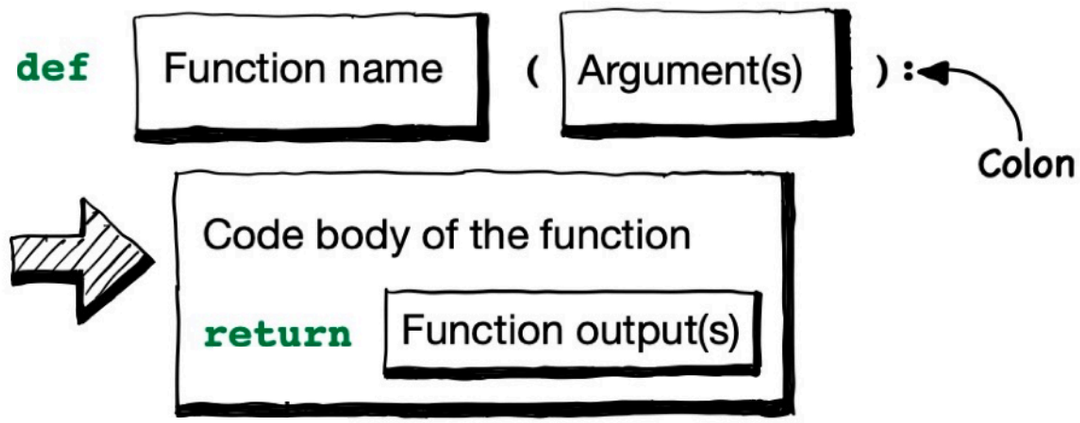


Notes: syntax for names:

- Only one word
- Only consist of letters, numbers, and underscores
- Cannot begin with a number
- Avoid contradictions with Python keywords

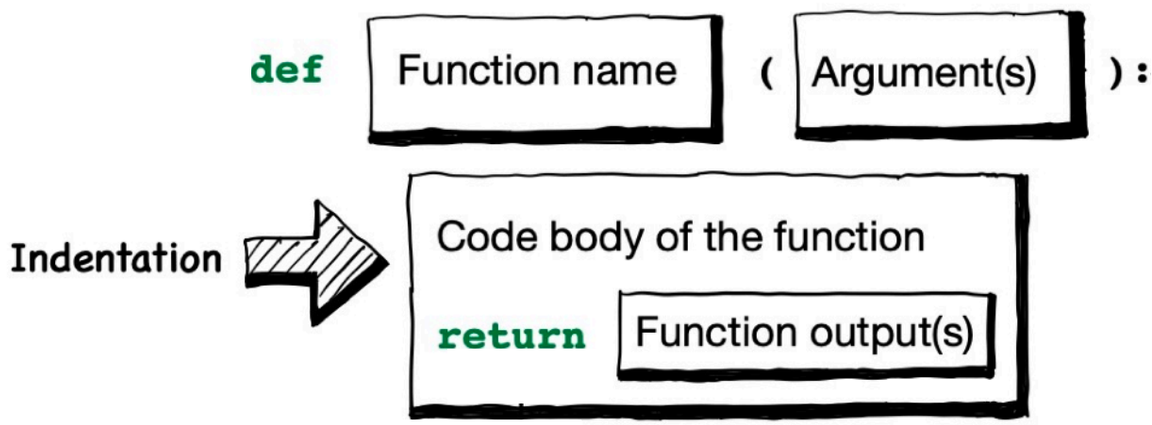
Functions

- Create a function
  - Syntax of function definition
    - ✓ Do not forget the colon



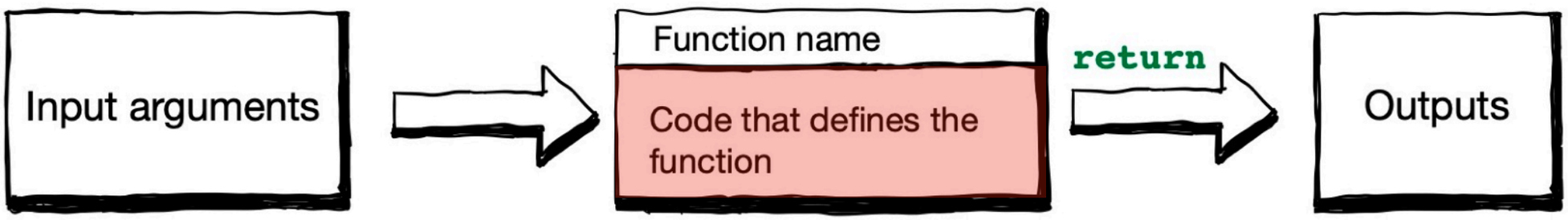
Functions

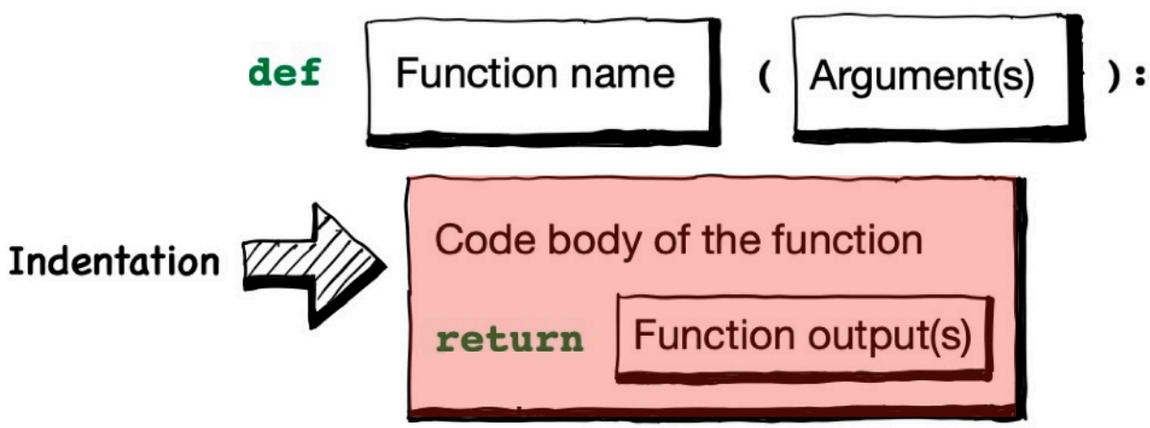
- Create a function
  - Syntax of function definition
    - ✓ Indentations are used to indicate the body of the function



Functions

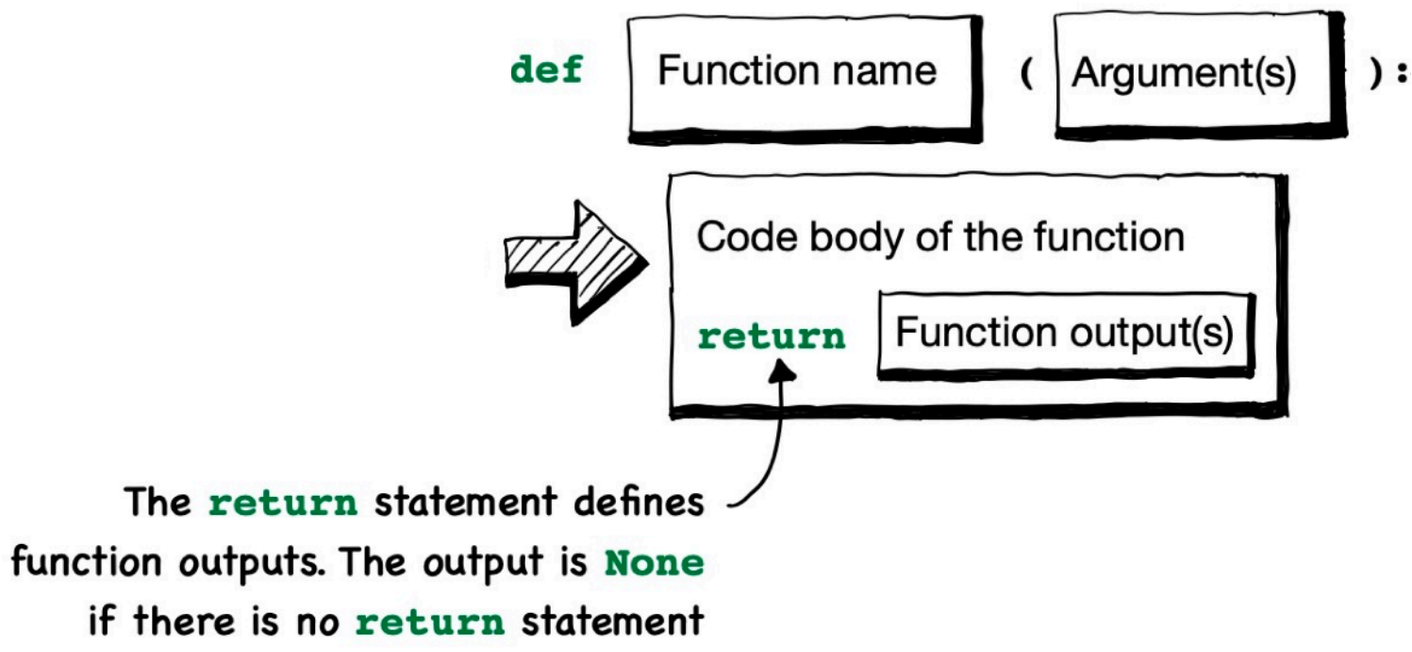
- Create a function





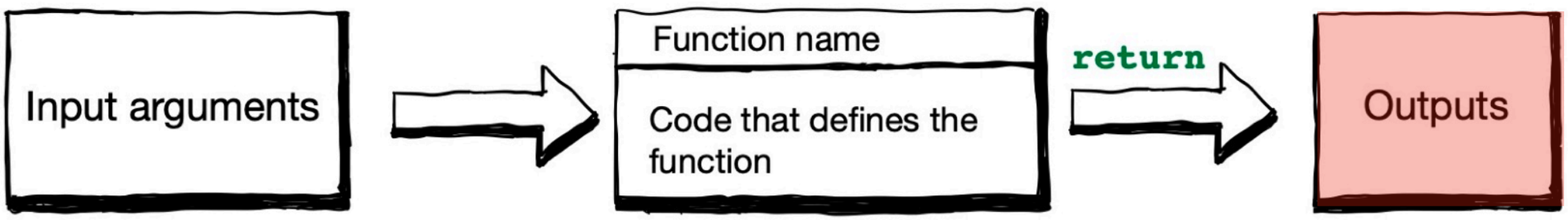
Functions

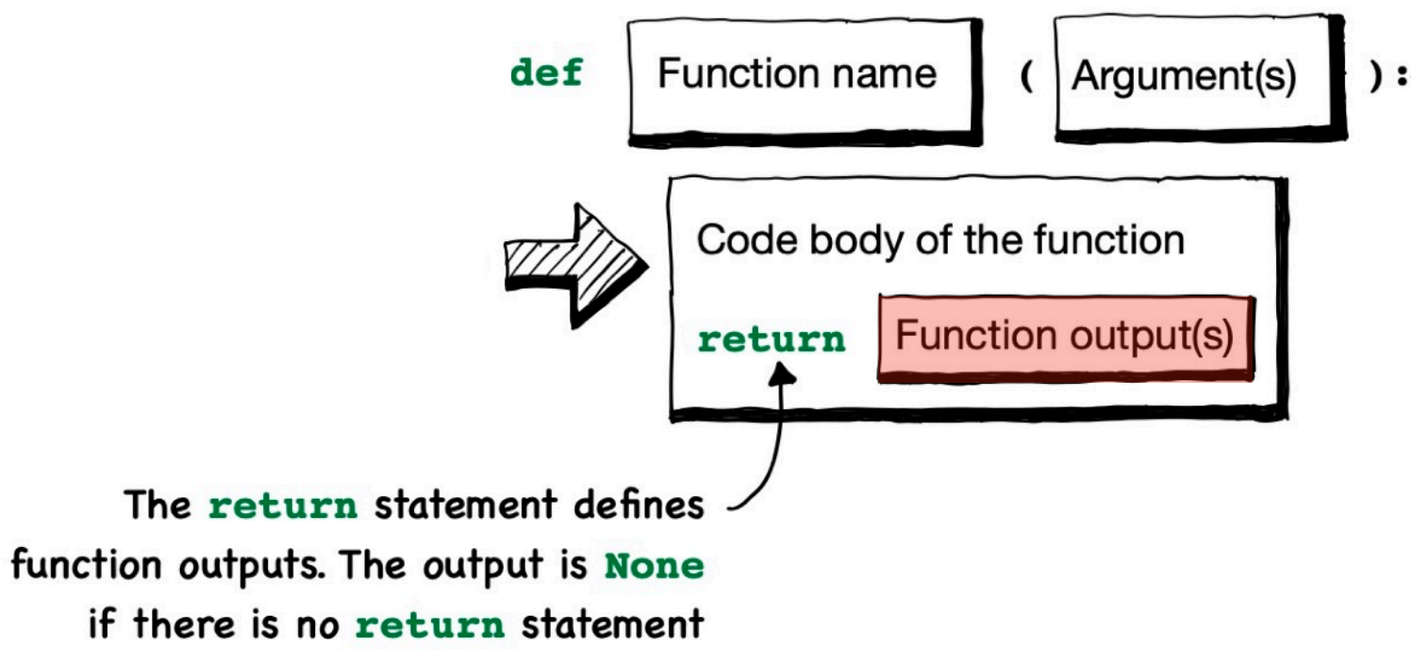
- Create a function
  - Syntax of function definition
    - ✓ Keyword `return`



Functions

- Create a function

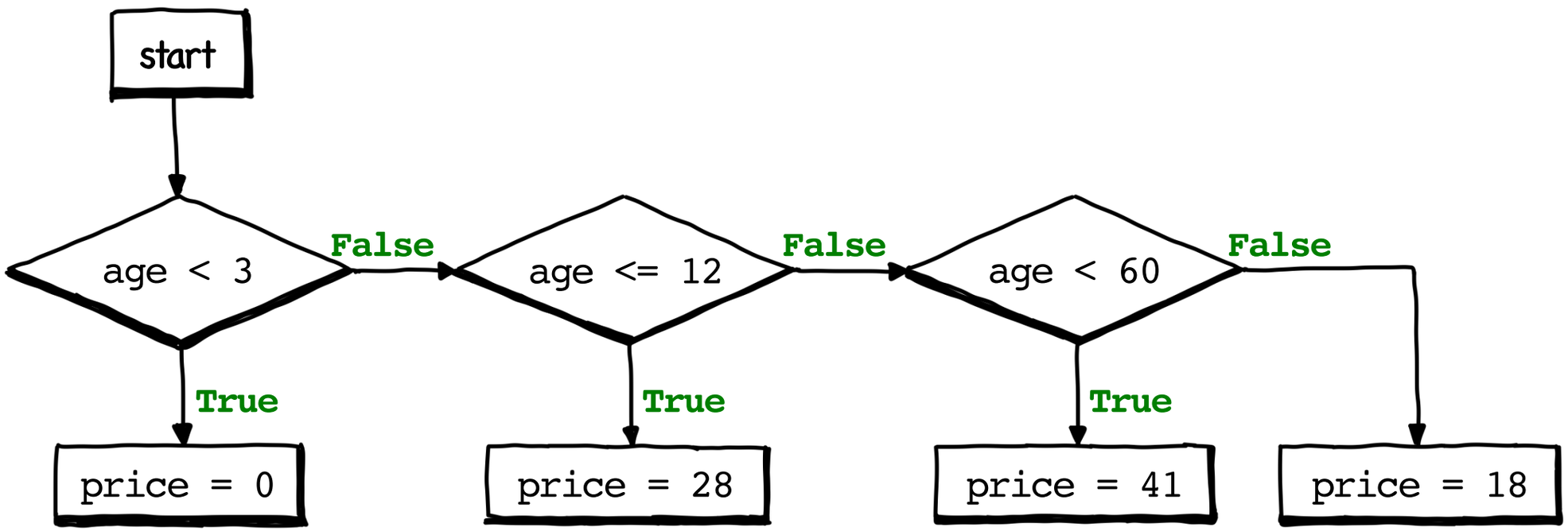




Functions

- Create a function
  - Syntax of function definition

Example 1: A group of four visitors are going to Singapore Zoo. Their ages are 11, 36, 37, and 67, respectively. Determine the ticket prices for these visitors.



Functions

- Create a function
  - Syntax of function definition

Example 1: A group of four visitors are going to Singapore Zoo. Their ages are 11, 36, 37, and 67, respectively. Determine the ticket prices for these visitors.

```
# Visitor 1
age1 = 11
if age1 < 3:
    ticket1 = 0
elif age1 <= 12:
    ticket1 = 28
elif age1 < 60:
    ticket1 = 41
else:
    ticket1 = 18
```

Functions

- Create a function
  - Syntax of function definition

Example 1: A group of four visitors are going to Singapore Zoo. Their ages are 11, 36, 37, and 67, respectively. Determine the ticket prices for these visitors.

```
# Visitor 2
age2 = 36
if age2 < 3:
    ticket2 = 0
elif age2 <= 12:
    ticket2 = 28
elif age2 < 60:
    ticket2 = 41
else:
    ticket2 = 18
```

Functions

- Create a function
  - Syntax of function definition

Example 1: A group of four visitors are going to Singapore Zoo. Their ages are 11, 36, 37, and 67, respectively. Determine the ticket prices for these visitors.

```
# Visitor 3
age3 = 37
if age3 < 3:
    ticket3 = 0
elif age3 <= 12:
    ticket3 = 28
elif age3 < 60:
    ticket3 = 41
else:
    ticket3 = 18
```

Functions

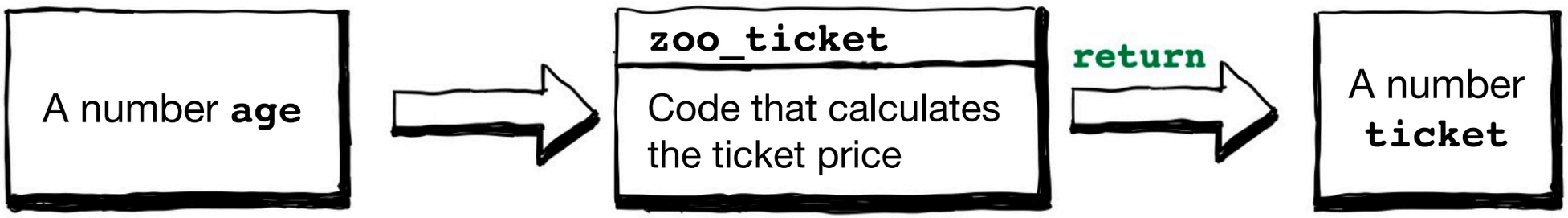
- Create a function
  - Syntax of function definition

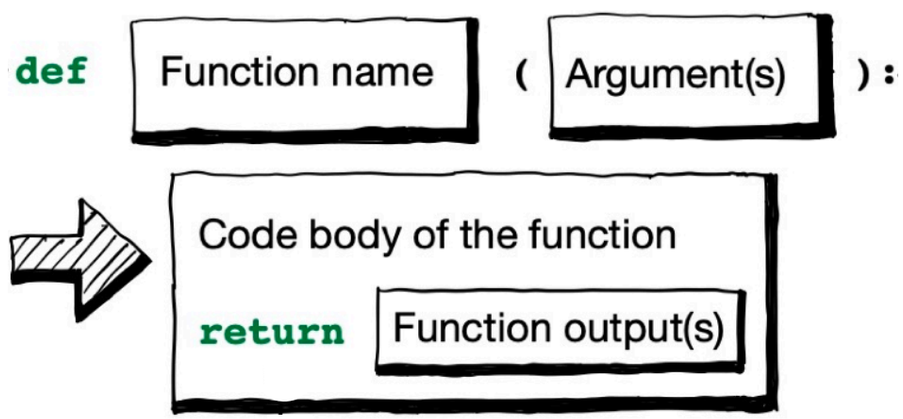
Example 1: A group of four visitors are going to Singapore Zoo. Their ages are 11, 36, 37, and 67, respectively. Determine the ticket prices for these visitors.

```
# Visitor 4
age4 = 67
if age4 < 3:
    ticket4 = 0
elif age4 <= 12:
    ticket4 = 28
elif age4 < 60:
    ticket4 = 41
else:
    ticket4 = 18
```

Functions

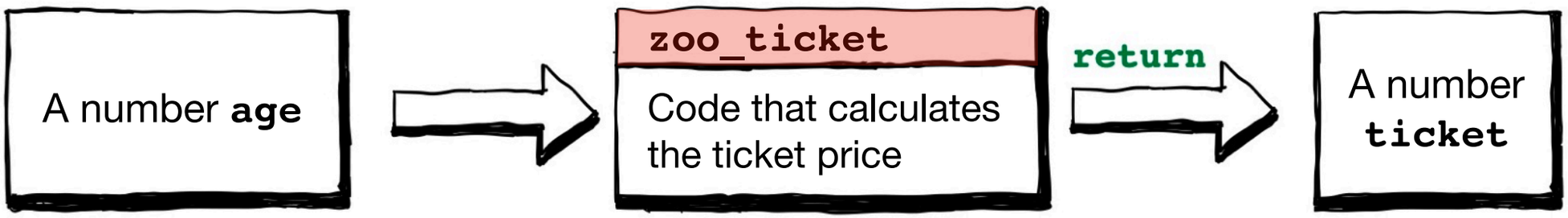
- Create a function
  - Syntax of function definition

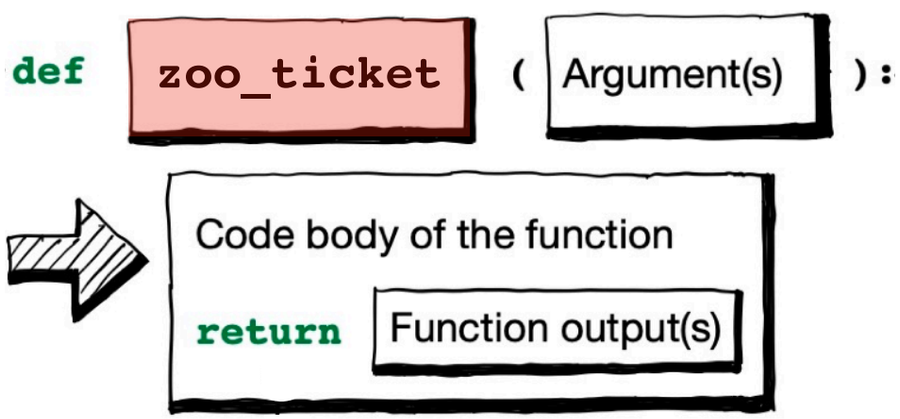




Functions

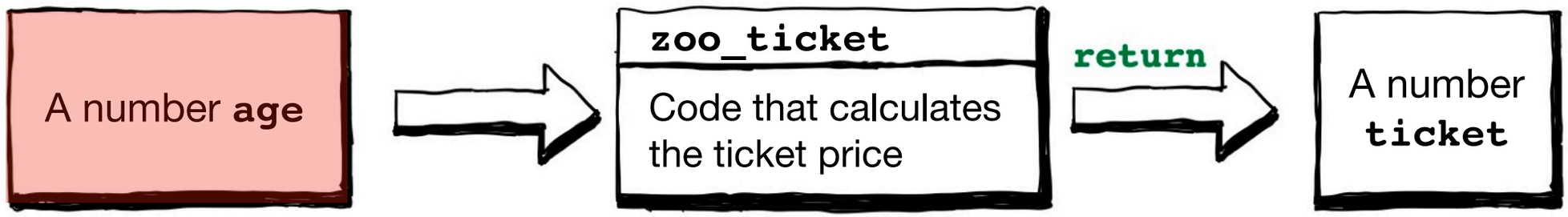
- Create a function
  - Syntax of function definition

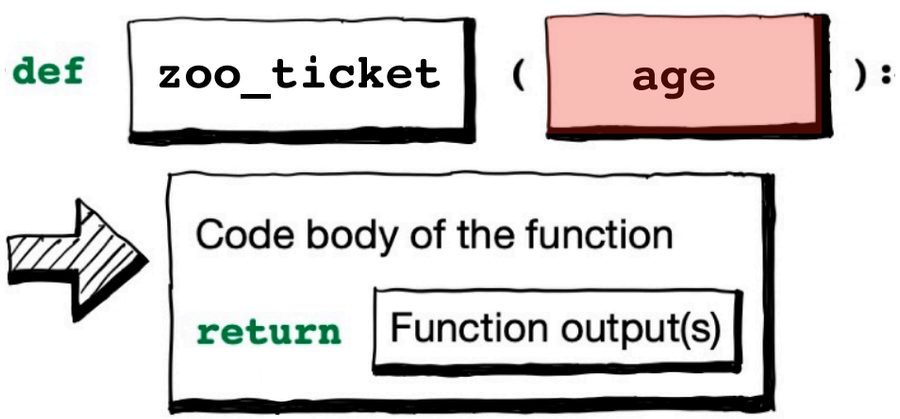




Functions

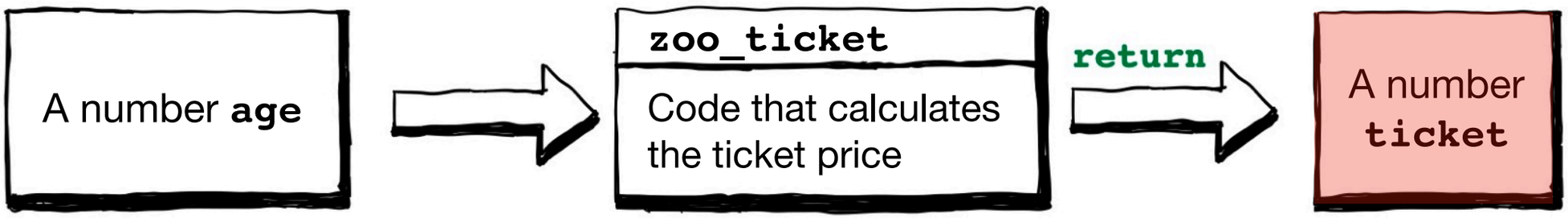
- Create a function
  - Syntax of function definition

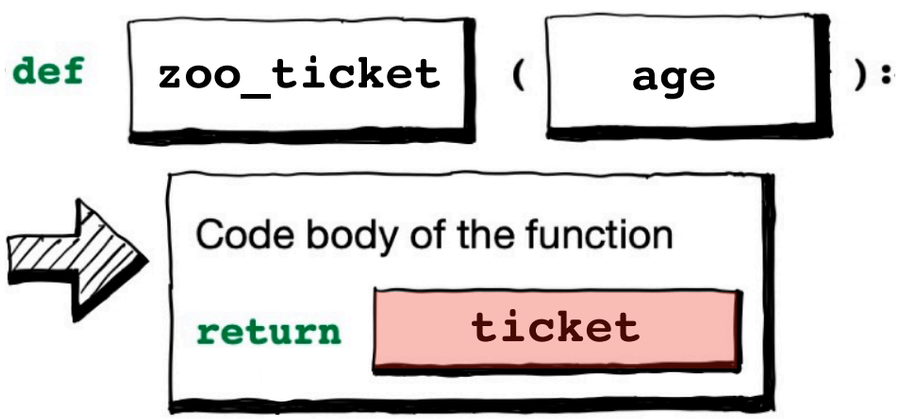




Functions

- Create a function
  - Syntax of function definition

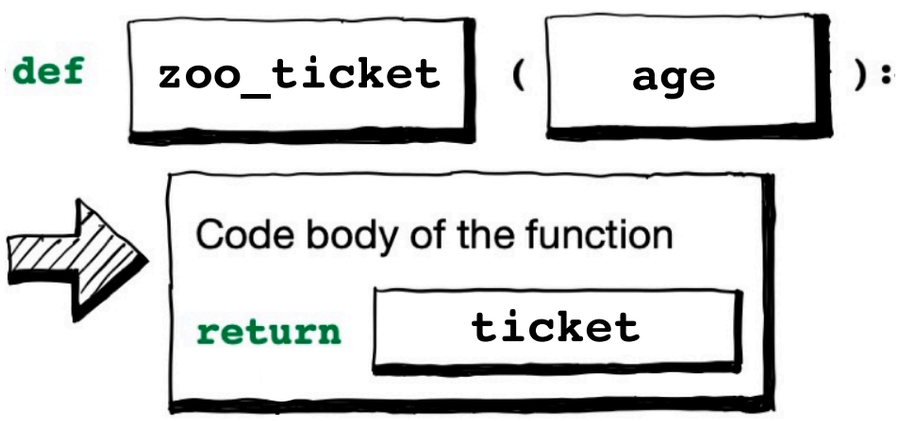




Functions

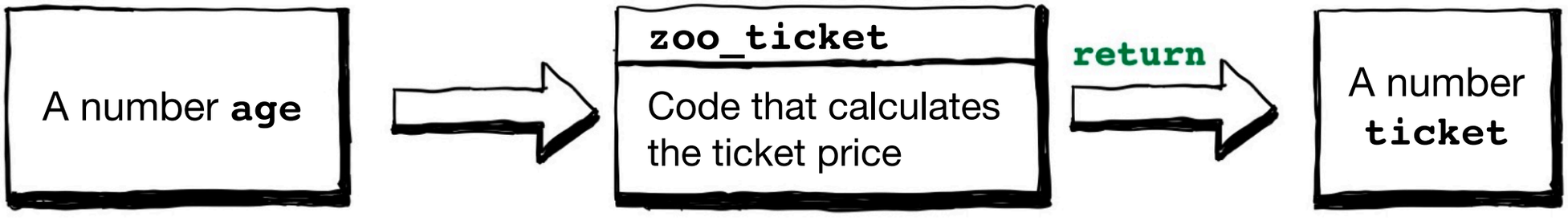
- Create a function
  - Syntax of function definition

```
def zoo_ticket(age):
    """The function zoo_ticket calculates the
    Singapore Zoo ticket prices for visitors"""

    if age < 3:
        ticket = 0
    elif age <= 12:
        ticket = 28
    elif age < 60:
        ticket = 41
    else:
        ticket = 18

    return ticket
```



Functions

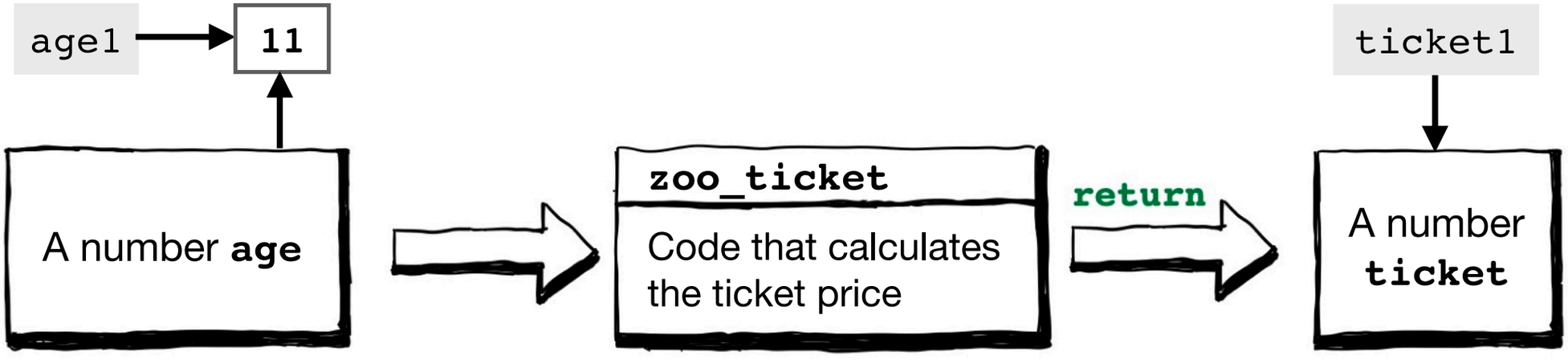
- Create a function




```
age1 = 11                        # Age of the 1st visitor
ticket1 = zoo_ticket(age1)       # Ticket price for the 1st visitor

age2 = 36                        # Age of the 2nd visitor
ticket2 = zoo_ticket(age2)       # Ticket price for the 2nd visitor

age3 = 37                        # Age of the 3rd visitor
ticket3 = zoo_ticket(age3)       # Ticket price for the 3rd visitor

age4 = 67                        # Age of the 4th visitor
ticket4 = zoo_ticket(age4)       # Ticket price for the 4th visitor
```

Functions

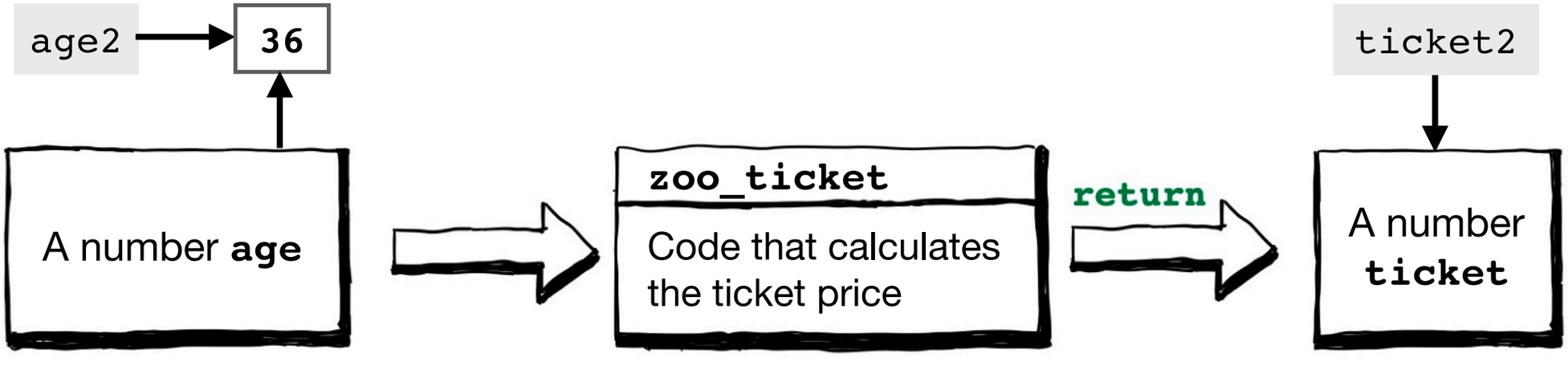
- Create a function




```
age1 = 11                       # Age of the 1st visitor
ticket1 = zoo_ticket(age1)      # Ticket price for the 1st visitor

age2 = 36                       # Age of the 2nd visitor
ticket2 = zoo_ticket(age2)      # Ticket price for the 2nd visitor

age3 = 37                       # Age of the 3rd visitor
ticket3 = zoo_ticket(age3)      # Ticket price for the 3rd visitor

age4 = 67                       # Age of the 4th visitor
ticket4 = zoo_ticket(age4)      # Ticket price for the 4th visitor
```

Functions

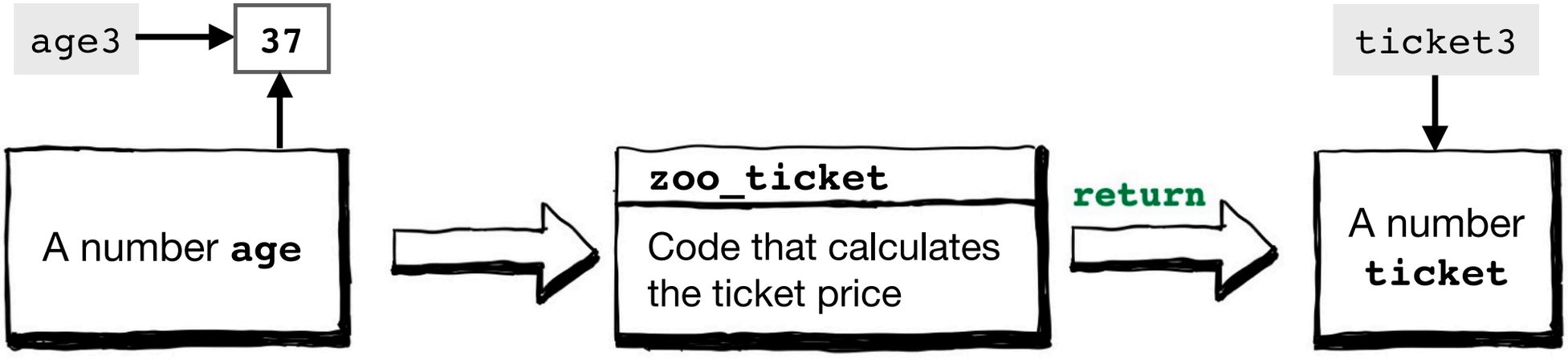
- Create a function




```
age1 = 11                       # Age of the 1st visitor
ticket1 = zoo_ticket(age1)      # Ticket price for the 1st visitor

age2 = 36                       # Age of the 2nd visitor
ticket2 = zoo_ticket(age2)      # Ticket price for the 2nd visitor

age3 = 37                       # Age of the 3rd visitor
ticket3 = zoo_ticket(age3)      # Ticket price for the 3rd visitor

age4 = 67                       # Age of the 4th visitor
ticket4 = zoo_ticket(age4)      # Ticket price for the 4th visitor
```

Functions

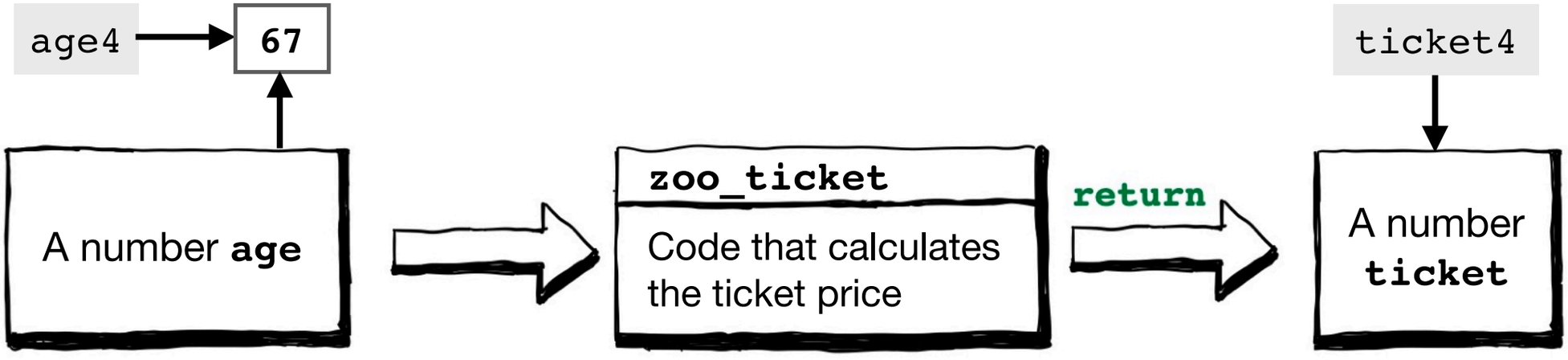
- Create a function




```
age1 = 11                       # Age of the 1st visitor
ticket1 = zoo_ticket(age1)      # Ticket price for the 1st visitor

age2 = 36                       # Age of the 2nd visitor
ticket2 = zoo_ticket(age2)      # Ticket price for the 2nd visitor

age3 = 37                       # Age of the 3rd visitor
ticket3 = zoo_ticket(age3)      # Ticket price for the 3rd visitor

age4 = 67                       # Age of the 4th visitor
ticket4 = zoo_ticket(age4)      # Ticket price for the 4th visitor
```

Functions

- Create a function




```
age1 = 11                       # Age of the 1st visitor
ticket1 = zoo_ticket(age1)      # Ticket price for the 1st visitor

age2 = 36                       # Age of the 2nd visitor
ticket2 = zoo_ticket(age2)      # Ticket price for the 2nd visitor

age3 = 37                       # Age of the 3rd visitor
ticket3 = zoo_ticket(age3)      # Ticket price for the 3rd visitor

age4 = 67                       # Age of the 4th visitor
ticket4 = zoo_ticket(age4)      # Ticket price for the 4th visitor
```

Functions

- Create a function
  - Syntax of function definition

```
def zoo_ticket(age):
    """The function zoo_ticket calculates the
    Singapore Zoo ticket prices for visitors"""

    if age < 3:
        ticket = 0
    elif age <= 12:
        ticket = 28
    elif age < 60:
        ticket = 41
    else:
        ticket = 18

    return ticket
```

Notes: A function terminates at **`return`**. After hitting the **`return`**, the remaining code in the function will not be executed.

Functions

- Create a function
  - Syntax of function definition

```
def zoo_ticket_new(age):
    """
    The function zoo_ticket_new is equivalent
    to the function zoo_ticket
    """
```

```
if age < 3:
    return 0
```

If the age is lower than 3, return 0 as the output

```
if age <= 12:
    return 28

if age < 60:
    return 41

return 18
```

This code block is skipped as the function stops at the previous return statement

Functions

- Create a function
  - Syntax of function definition

```
def zoo_ticket_new(age):
    """
    The function zoo_ticket_new is equivalent
    to the function zoo_ticket
    """
```

```
if age < 3:
    return 0

if age <= 12:
    return 28
```

If the age is between 3 and 12, return 28 as the output

```
if age < 60:
    return 41

return 18
```

The code block is skipped as the function stops at the previous return statement

Functions

- Create a function
  - Syntax of function definition

```
def zoo_ticket_new(age):
    """
    The function zoo_ticket_new is equivalent
    to the function zoo_ticket
    """
```

```
if age < 3:
    return 0

if age <= 12:
    return 28

if age < 60:
    return 41
```

If the age is larger than 12 and lower than 60, return 41 as the output

```
return 18
```

Skip the last statement as the function stops at the previous return statement

Functions

- Create a function
  - Syntax of function definition

```
def zoo_ticket_new(age):
    """
    The function zoo_ticket_new is equivalent
    to the function zoo_ticket
    """

    if age < 3:
        return 0

    if age <= 12:
        return 28

    if age < 60:
        return 41

    return 18
```

If the age is larger than or equal to 60, skip all previous return statements and return 18

Functions

- Create a function
  - Syntax of function definition

```
def zoo_ticket(age):
    """The function zoo_ticket calculates the
    Singapore Zoo ticket prices for visitors"""

    if age < 3:
        ticket = 0
    elif age <= 12:
        ticket = 28
    elif age < 60:
        ticket = 41
    else:
        ticket = 18

    return ticket
```

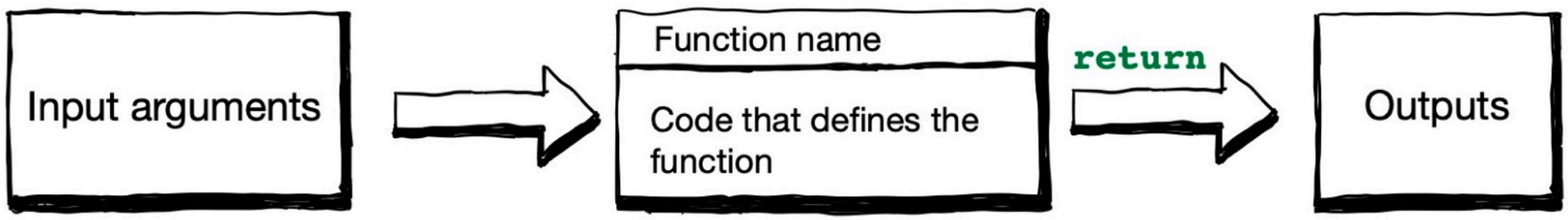
Coding Style: At the beginning of each function, there is a string, called a **docstring**, used to explain how the function behaves. Style of the docstring can be found in **PEP 257 Docstring Conventions**.

Functions

- Create a function
  - Benefits of using functions
    - ✓ Reuse code
    - ✓ Easy to test and debug (separately)
    - ✓ Higher readability

Functions

- Function arguments and outputs



Functions

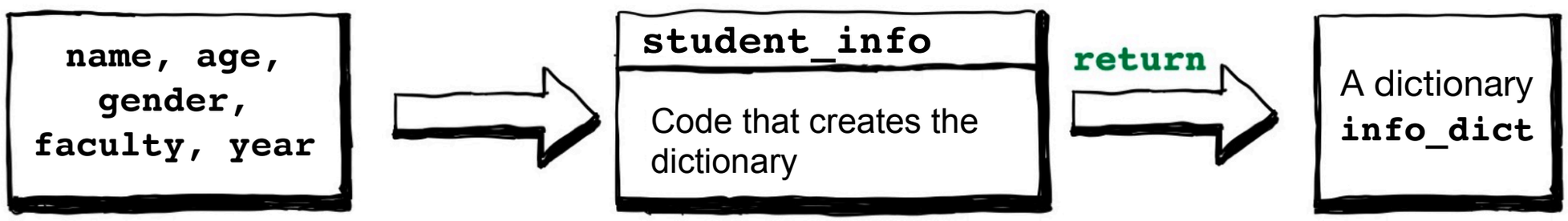
- Function arguments and outputs

Example 2: Write a function to summarize the information of students registered for Python BA course. The input arguments are student's name, age, gender, faculty, and year of enrollment. The output is a dictionary.

```
def student_info(name, age, gender, faculty, year):
    "This function summarizes student info. into a dictionary"

    info_dict = {'name': name,
                 'age': age,
                 'gender': gender,
                 'faculty': faculty,
                 'year': year}

    return info_dict
```

Functions

- Function arguments and outputs




```
def student_info(name, age, gender, faculty, year):
    "This function summarizes student info. into a dictionary"

    info_dict = {'name': name,
                 'age': age,
                 'gender': gender,
                 'faculty': faculty,
                 'year': year}

    return info_dict
```

Functions

- Function arguments and outputs
  - Positional arguments

```
student = student_info('Jane Doe', 20, 'F', 'Business', 2019)
```

```
def student_info(name, age, gender, faculty, year):
    "This function summarizes student info. into a dictionary"

    info_dict = {'name': name,
                 'age': age,
                 'gender': gender,
                 'faculty': faculty,
                 'year': year}

    return info_dict
```

Functions

- Function arguments and outputs
  - Keyword arguments

```
student = student_info('Jane Doe', 20,
                       year=2019, faculty='Business', gender='F')
```

```
def student_info(name, age, gender, faculty, year):
    "This function summarizes student info. into a dictionary"

    info_dict = {'name': name,
                 'age': age,
                 'gender': gender,
                 'faculty': faculty,
                 'year': year}

    return info_dict
```

Keyword arguments

Functions

- Function arguments and outputs
  - Keyword arguments

```
student = student_info('Jane Doe', 20,
                       year=2019, faculty='Business', gender='F')
```

Positional arguments

```
def student_info(name, age, gender, faculty, year):
    "This function summarizes student info. into a dictionary"

    info_dict = {'name': name,
                 'age': age,
                 'gender': gender,
                 'faculty': faculty,
                 'year': year}

    return info_dict
```

Functions

- Function arguments and outputs
  - Keyword arguments

```
student = student_info('Jane Doe', 20,
                       year=2019, faculty='Business', gender='F')
```

```
def student_info(name, age, gender, faculty, year):
    "This function summarizes student info. into a dictionary"

    info_dict = {'name': name,
                 'age': age,
                 'gender': gender,
                 'faculty': faculty,
                 'year': year}

    return info_dict
```

Notes: In Python programming, all positional arguments must be specified before keyword arguments, otherwise there will be an error message.

Functions

- Function arguments and outputs
  - Keyword arguments

```
student = student_info('Jane Doe', 20,
                       year=2019, faculty='Business', gender='F')
```

```
def student_info(name, age, gender, faculty, year):
    "This function summarizes student info. into a dictionary"

    info_dict = {'name': name,
                 'age': age,
                 'gender': gender,
                 'faculty': faculty,
                 'year': year}

    return info_dict
```

Coding Style: Do not use spaces around the equal sign while specifying keyword arguments.

Functions

- Function arguments and outputs
  - Default values

```
def student_info(name, age, gender, faculty='Business', year=2019):
    "This function summarizes student info. into a dictionary"

    info_dict = {'name': name,
                 'age': age,
                 'gender': gender,
                 'faculty': faculty,
                 'year': year}

    return info_dict
```

Define default values of function arguments

Functions

- Function arguments and outputs
  - Default values

```
def student_info(name, age, gender, faculty='Business', year=2019):
    "This function summarizes student info. into a dictionary"

    info_dict = {'name': name,
                 'age': age,
                 'gender': gender,
                 'faculty': faculty,
                 'year': year}

    return info_dict
```

```
{'name': 'Jane Doe',
 'age': 20,
 'gender': 'F',
 'faculty': 'Business',
 'year': 2019}
```

```
student = student_info('Jane Doe', 20, 'F')
```

Unspecified arguments take default values

Functions

- Function arguments and outputs
  - Default values

```
def student_info(name, age, gender, faculty='Business', year=2019):
    "This function summarizes student info. into a dictionary"

    info_dict = {'name': name,
                 'age': age,
                 'gender': gender,
                 'faculty': faculty,
                 'year': year}

    return info_dict
```

```
{'name': 'John Doe',
 'age': 20,
 'gender': 'M',
 'faculty': 'Business',
 'year': 2020}
```

```
student = student_info('John Doe', 20, 'M', year=2020)
```

Unspecified argument takes the default value

Functions

- Function arguments and outputs
  - Accessing argument information using the `help()` function
    - ✓ Order of arguments
    - ✓ Keyword names
    - ✓ Default values

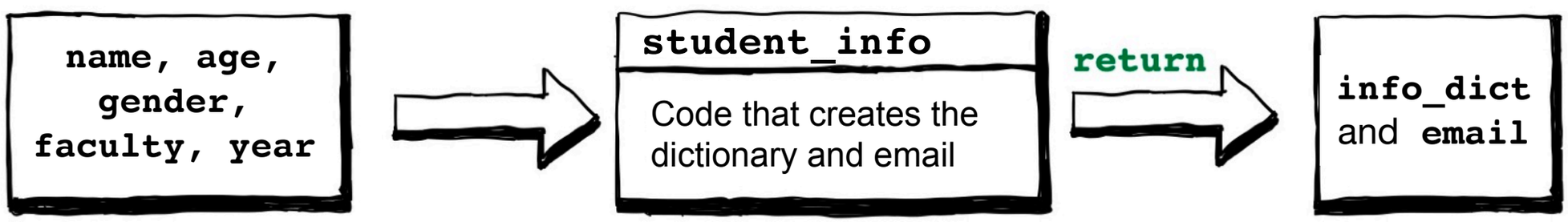
```
help(student_info)
```

```
Help on function student_info in module __main__:

student_info(name, age, gender, faculty='Business', year=2019)
    This function summarizes student information into a dictionary
```

Functions

- Function arguments and outputs
  - Multiple outputs

Example 3: Besides the dictionary of student's information, the **`student_info()`** function is modified to also returns the email of the student, which is in the format of "<first name>_<last name>_<faculty>@u.nus.edu.sg", with all letters to be lower case.



Functions

- Function arguments and outputs
  - Multiple outputs

```
def student_info(name, age, gender, faculty='Business', year=2019):
    """
    This function summarizes student info. into a dictionary and
    return the generated email for the student.
    """

    info_dict = {'name': name,
                 'age': age,
                 'gender': gender,
                 'faculty': faculty,
                 'year': year}

    lower_name = name.lower().replace(' ', '_')
    lower_faculty = faculty.lower()
    email = '{}_{}@u.nus.edu.sg'.format(lower_name, lower_faculty)

    return info_dict, email
```

What is the data type?

Functions

- Function arguments and outputs
  - Multiple outputs

```
def student_info(name, age, gender, faculty='Business', year=2019):
    """
    This function summarizes student info. into a dictionary and
    return the generated email for the student.
    """

    info_dict = {'name': name,
                 'age': age,
                 'gender': gender,
                 'faculty': faculty,
                 'year': year}

    lower_name = name.lower().replace(' ', '_')
    lower_faculty = faculty.lower()
    email = '{}_{}@u.nus.edu.sg'.format(lower_name, lower_faculty)

    return info_dict, email
```

A tuple!!

Functions

- Function arguments and outputs
  - Multiple outputs

```
student1, email1 = student_info('Jane Doe', 20, 'F')
student1
```

```
{'name': 'Jane Doe',
 'age': 20,
 'gender': 'F',
 'faculty': 'Business',
 'year': 2019}
```

```
email1
```

```
'jane_doe_business@u.nus.edu.sg'
```

Functions

- Function arguments and outputs
  - Multiple outputs

```
student2, email2 = student_info('Sam Lee', 21, 'M', 'Engineering')
student2
```

```
{'name': 'Sam Lee',
 'age': 21,
 'gender': 'M',
 'faculty': 'Engineering',
 'year': 2019}
```

```
email2
```

```
'sam_lee_engineering@u.nus.edu.sg'
```

Functions

- Scopes and namespaces
  - A namespace is a collection of names
  - The same name may be used to represent different objects in different scopes

Functions

- Scopes and namespaces
  - A namespace is a collection of names
  - The same name may be used to represent different objects in different scopes
  - Scopes
    - ✓ Local: input arguments and names declared (by assignment statements) in a function and are effective only within the function.
    - ✓ Global: names declared outside of function definitions.
    - ✓ Built-in: names predefined in Python.

Functions

- Scopes and namespaces

> **Example 4:** A group of four visitors are going to Singapore Zoo. Their ages are given in a list **`age`**. Use the function **`zoo_ticket()`** defined above to create a list named **`ticket`**, which provides the corresponding ticket prices.

Functions

- Scopes and namespaces

```
age = [11, 36, 37, 67]
```

Global scope

```
def zoo_ticket(age):
    """The function zoo_ticket calculates the
    Singapore Zoo ticket prices for visitors"""

    if age < 3:
        ticket = 0
    elif age <= 12:
        ticket = 28
    elif age < 60:
        ticket = 41
    else:
        ticket = 18

    return ticket
```

Local scope

```
ticket = [zoo_ticket(item) for item in age]
```

Global scope

Functions

- Scopes and namespaces

```
age = [11, 36, 37, 67]
```

Global name "age" is a list

Global scope

```
def zoo_ticket(age):
    """The function zoo_ticket calculates the
    Singapore Zoo ticket prices for visitors"""

    if age < 3:
        ticket = 0
    elif age <= 12:
        ticket = 28
    elif age < 60:
        ticket = 41
    else:
        ticket = 18

    return ticket
```

Local name "age" is an integer

Local scope

```
ticket = [zoo_ticket(item) for item in age]
```

Global name "age" is a list

Global scope

Functions

- Scopes and namespaces

```
age = [11, 36, 37, 67]
```

Global scope

```
def zoo_ticket(age):
    """The function zoo_ticket calculates the
    Singapore Zoo ticket prices for visitors"""

    if age < 3:
        ticket = 0
    elif age <= 12:
        ticket = 28
    elif age < 60:
        ticket = 41
    else:
        ticket = 18

    return ticket
```

Local scope

Local name "ticket" is an integer

```
ticket = [zoo_ticket(item) for item in age]
```

Global scope

Global name "ticket" is a list

Functions

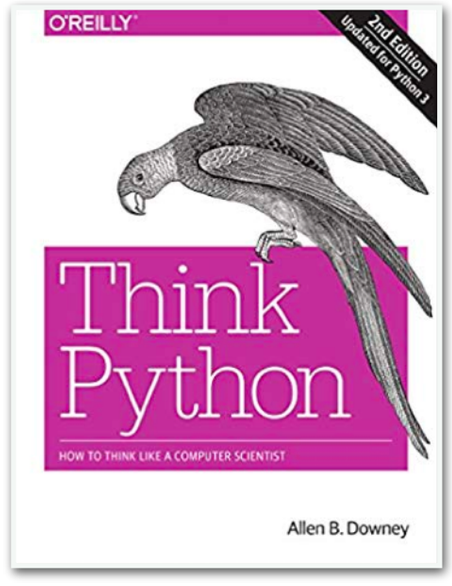
- Leap of faith

Leap of Faith

Following the flow of execution is one way to read programs, but it can quickly become labyrinthine. An alternative is what I call the "leap of faith." When you come to a function call, instead of following the flow of execution, you *assume* that the function works correctly and returns the right result.

… …

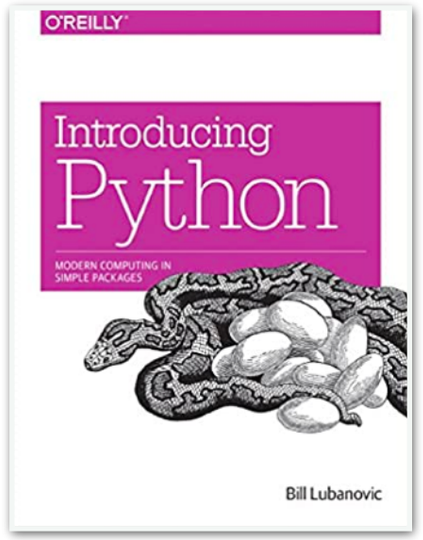
Once we have convinced ourselves that this function is correct—by examining the code and testing—we can use the function without looking at the body again. — **Think Python**



Modules

- Introduction to modules
  - A module is a “.py" file that defines functions, classes, variables, or simply contains some runnable code.

The text of this book is organized in a hierarchy: words, sentences, paragraphs, and chapters. Otherwise, it would be unreadable after a page or two. Code has a roughly similar bottom-up organization: data types are like words, statements are like sentences, functions are like paragraphs, and modules are like chapters. To continue the analogy, in this book, when I say that something will be explained in Chapter 8, in programming, that’s like referring to code in another module.— **Introducing Python**



Modules

- Introduction to modules
  - Benefits of using modules
    - ✓ Code reuse
    - ✓ System namespace partitioning
    - ✓ Implementing shared services or data

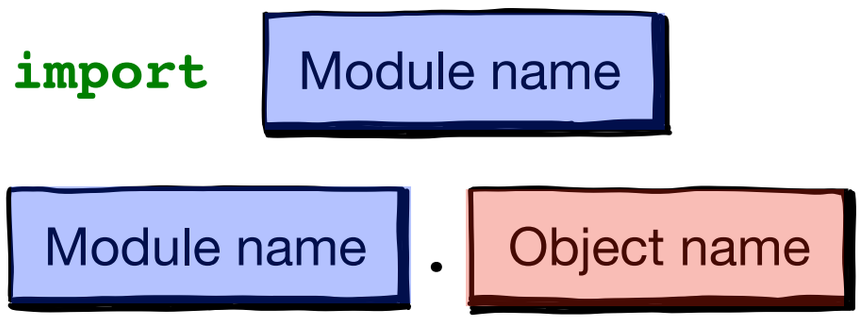
Modules

- Syntax of importing modules

> **Example 5:** Use functions **`mean()`** and **`variance()`** from the module **`statistics`** to calculate the sample average and the sample variance of numbers in a list.

Modules

- Syntax of importing modules

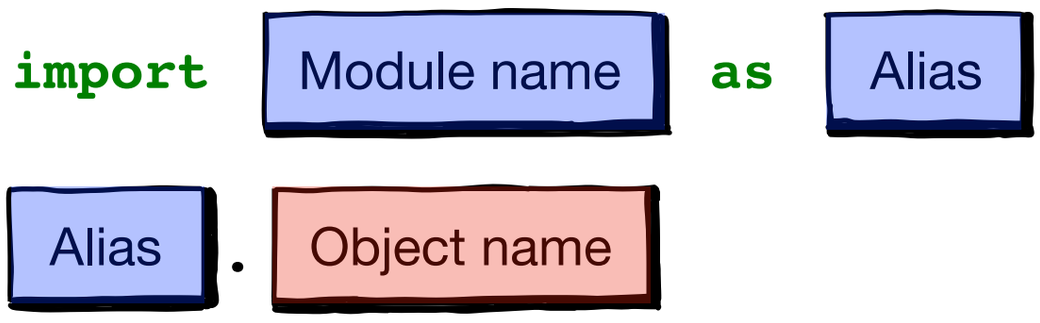



```
import statistics

numbers = [0.5, 1.2, 3.8, 2.4, 4.1, 3.5]

print(statistics.mean(numbers))
print(statistics.variance(numbers))
```

```
2.583333333333333
2.1816666666666666
```

Modules

- Syntax of importing modules

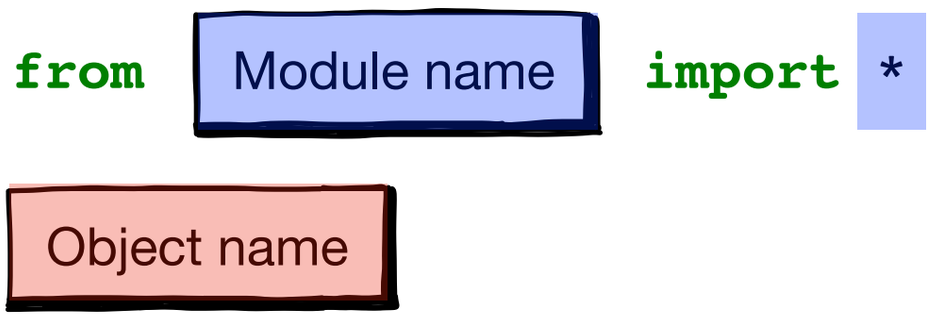



```
import statistics as st

numbers = [0.5, 1.2, 3.8, 2.4, 4.1, 3.5]

print(st.mean(numbers))
print(st.variance(numbers))
```

```
2.583333333333333
2.1816666666666666
```

Modules

- Syntax of importing modules



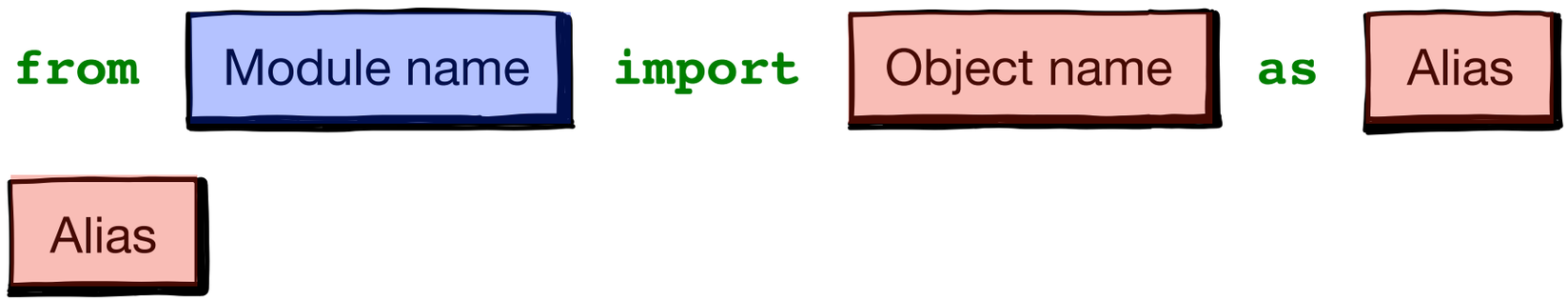

```
from statistics import *

numbers = [0.5, 1.2, 3.8, 2.4, 4.1, 3.5]

print(mean(numbers))
print(variance(numbers))
```

```
2.583333333333333
2.1816666666666666
```

Coding Style: It is usually not recommended to import everything from a module to omit the module name, as the code above.

Modules

- Syntax of importing modules



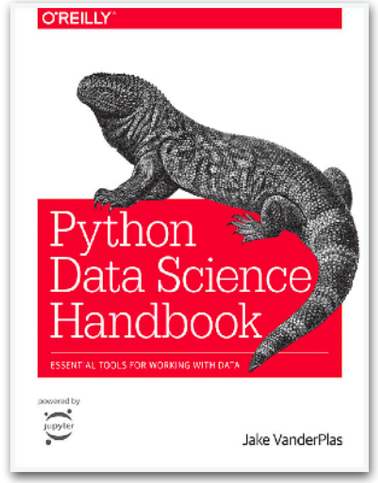

```
from statistics import mean as me
from statistics import variance as var

numbers = [0.5, 1.2, 3.8, 2.4, 4.1, 3.5]

print(me(numbers))
print(var(numbers))
```

```
2.583333333333333
2.1816666666666666
```

Packages

- Introduction to packages

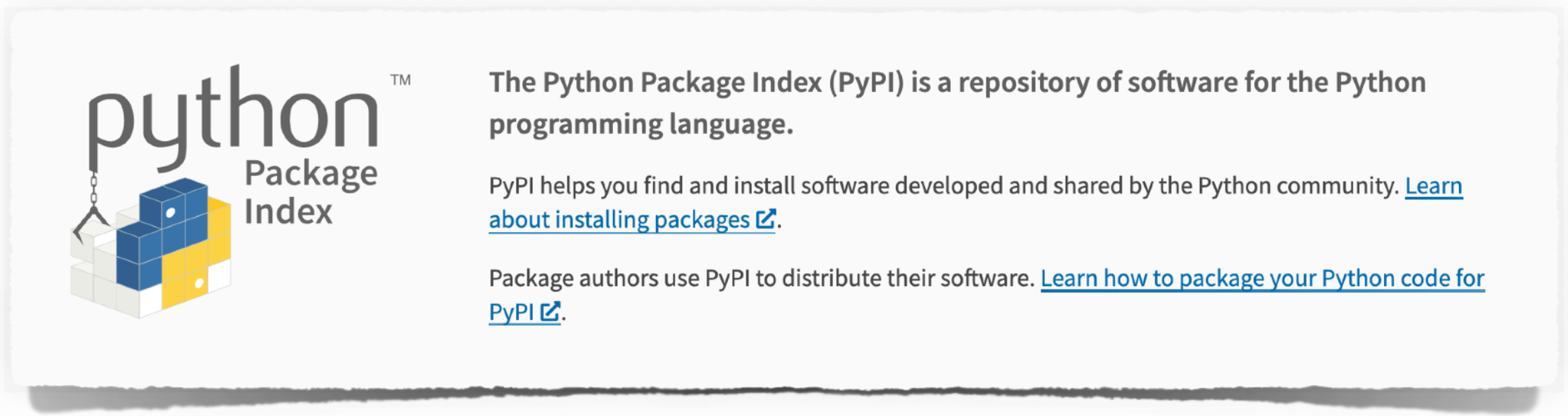
The usefulness of Python for data science stems primarily from the large and active ecosystem of third-party packages: NumPy for manipulation of homogeneous array-based data, Pandas for manipulation of heterogeneous and labeled data, SciPy for common scientific computing tasks, Matplotlib for publication-quality visualizations, IPython for interactive execution and sharing of code, Scikit-Learn for machine learning, and many more tools that will be mentioned in the following pages.



Packages

- Introduction to packages
  - A collection of modules and supporting files
  - Install a third-party package
    - ✓ Run the following command in Jupyter Notebook code cells

```
!pip install package_name
```



Packages

- Syntax of importing packages
  - Similar syntax as import modules
  - Use `.` operator to indicate the directory hierarchy

```
import pandas as pd
import matplotlib.pyplot as plt

from numpy import random
from scipy.stats import norm
from scipy.stats import binom
```

Coding Style:

- **Import different modules and packages in separate lines.**
- **Place import statements at the top of the code file**

Packages

- Package information
  - Package documentation

Packages

- Package information
  - Package documentation
  - Run `help()` to display the docstring
  - Google keywords and examples

Matplotlib for Data Visualization

The beautiful data visualization - David McCandless

Matplotlib for Data Visualization

- Functions for data visualization
  - Import Matplotlib tools

```
import matplotlib.pyplot as plt
```

  - Other data visualization packages
    - ✓ Plotly
    - ✓ Seaborn

Matplotlib for Data Visualization

- Functions for data visualization

Example 6: The dictionary **`rates`** provides the rates of return of the stock prices of Facebook and Google during the first six months of 2021. The other dictionary **`volumes`** provides the volumes of trading of these two stocks over the same time horizon. Visualize the trend of the return rates of these two stocks.

```
rates = {'FB': [-5.43, -0.27, 14.33, 10.37, 1.12, 5.77],
         'GOOG': [4.79, 10.96, 1.56, 16.51, 0.06, 3.93]}

volumes = {'FB': [452.63, 317.78, 497.42, 421.26, 353.42, 344.81],
           'GOOG': [33.08, 29.43, 34.79, 30.72, 25.46, 27.43]}

months = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun']
```

Matplotlib for Data Visualization

- Functions for data visualization

> **Example 6:** The dictionary **`rates`** provides the rates of return of the stock prices of Facebook and Google during the first six months of 2021. The other dictionary **`volumes`** provides the volumes of trading of these two stocks over the same time horizon. Visualize the trend of the return rates of these two stocks.

- Line plot

✓ The `help()` information of the `plot()` function

Sequences of data items

```
... ...
    Parameters
    ----------
    x, y : array-like or scalar
        The horizontal / vertical coordinates of the data points.
        *x* values are optional and default to ``range(len(y))``.

        Commonly, these parameters are 1D arrays.
... ...
```

Matplotlib for Data Visualization

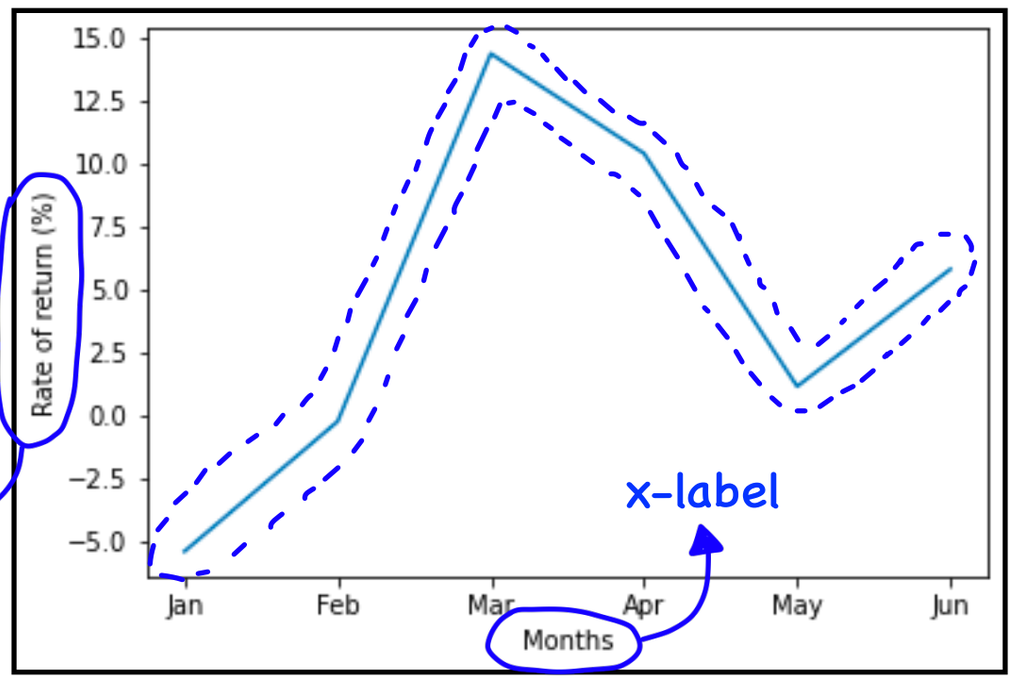
- Functions for data visualization
  - Line plot

```
rate = rates[name]

plt.plot(months, rate)
plt.xlabel('Months')
plt.ylabel('Rate of return (%)')
plt.show()
```

x-data

y-data

y-label



rate	-5.43	-0.27	14.33	10.37	1.12	5.77
months	'Jan'	'Feb'	'Mar'	'Apr'	'May'	'Jun'

Matplotlib for Data Visualization

- Functions for data visualization
  - Bar chart

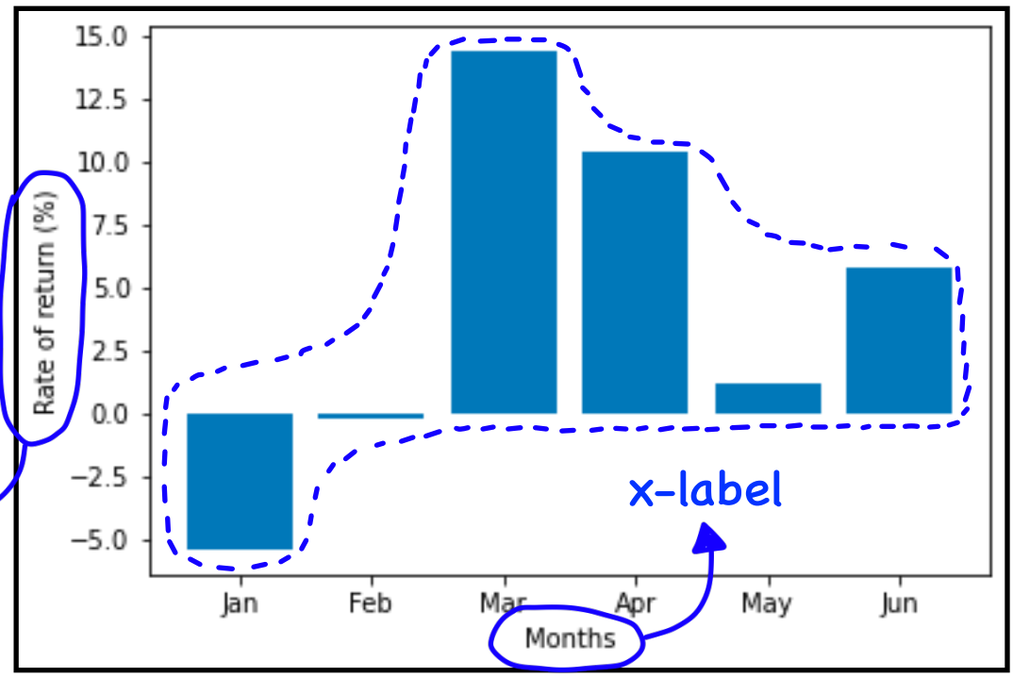
```
rate = rates[name]

plt.bar(months, rate)
plt.xlabel('Months')
plt.ylabel('Rate of return (%)')
plt.show()
```

x-data

y-data

y-label

x-label



rate	-5.43	-0.27	14.33	10.37	1.12	5.77
months	'Jan'	'Feb'	'Mar'	'Apr'	'May'	'Jun'

Matplotlib for Data Visualization

- Functions for data visualization

> **Example 6:** The dictionary **`rates`** provides the rates of return of the stock prices of Facebook and Google during the first six months of 2021. The other dictionary **`volumes`** provides the volumes of trading of these two stocks over the same time horizon. Visualize the trend of the return rates of these two stocks.

- Scatter plot

✓ The `help()` information of the `scatter()` function

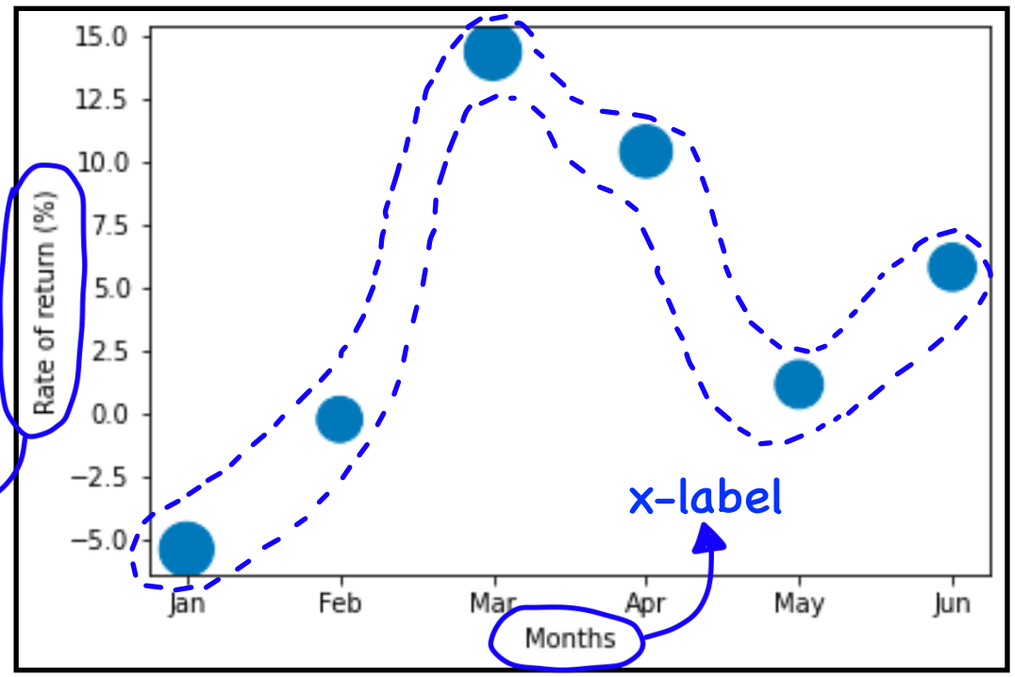
```
... ...
    Parameters
    ----------
    x, y : float or array-like, shape (n, )
        The data positions.

    s : float or array-like, shape (n, ), optional
        The marker size in points**2.
        Default is ``rcParams['lines.markersize'] ** 2``.
... ...
```

Matplotlib for Data Visualization

- Functions for data visualization
  - Scatter plot

```
rate = rates[name]
volume = volumes[name]

plt.scatter(months, rate, volume)
plt.xlabel('Months')
plt.ylabel('Rate of return (%)')
plt.show()
```

x-data
y-data
Marker sizes
y-label

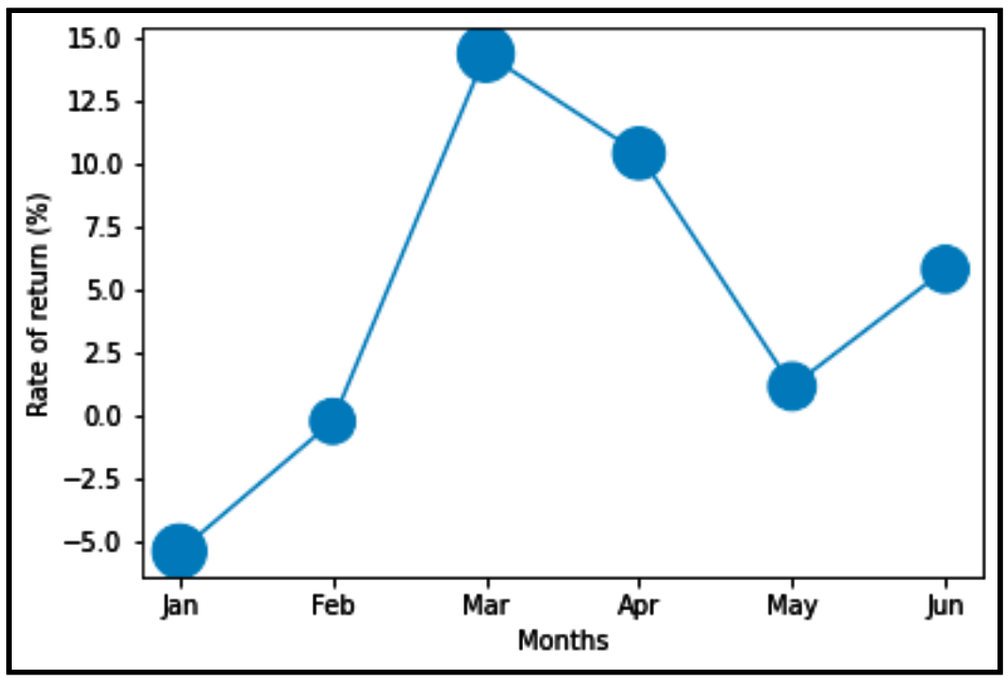


rate	-5.43	-0.27	14.33	10.37	1.12	5.77
months	'Jan'	'Feb'	'Mar'	'Apr'	'May'	'Jun'
volume	452.63	317.78	497.42	421.26	353.42	344.81

Matplotlib for Data Visualization

- Functions for data visualization
  - Multiple plot components

```
rate = rates[name]
volume = volumes[name]

plt.plot(months, rate)
plt.scatter(months, rate, volume)
plt.xlabel('Months')
plt.ylabel('Rate of return (%)')
plt.show()
```



rate	-5.43	-0.27	14.33	10.37	1.12	5.77
months	'Jan'	'Feb'	'Mar'	'Apr'	'May'	'Jun'
volume	452.63	317.78	497.42	421.26	353.42	344.81

Matplotlib for Data Visualization

- Configuration of plots and figures
  - Specify plot features by keyword arguments

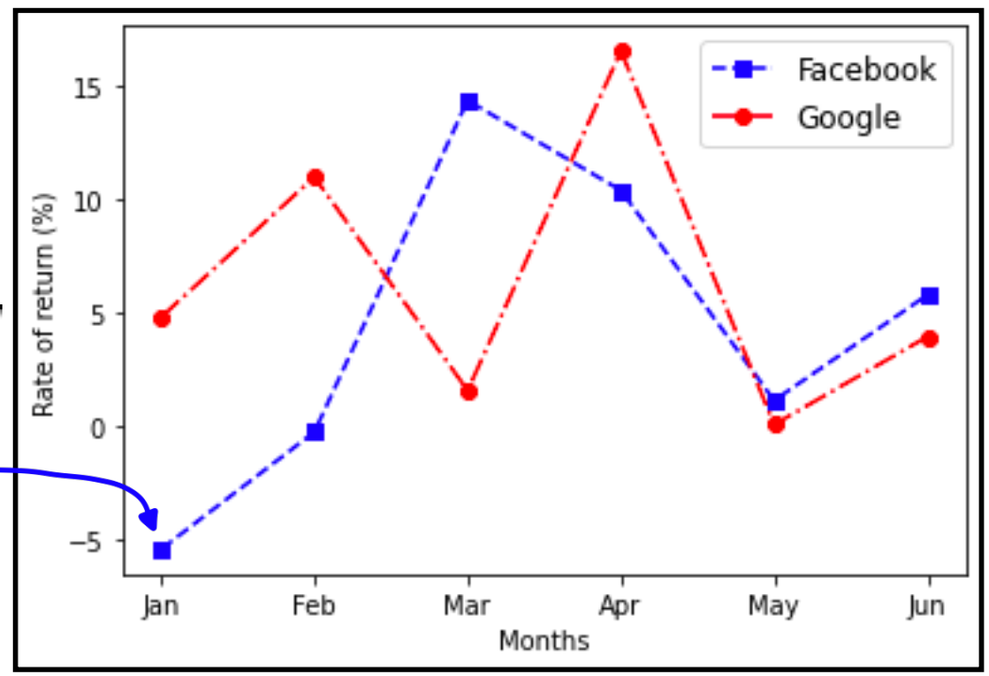
```
plt.plot(months, rates['FB'],
         color='b', linestyle='--', linewidth=1.5, marker='s',
         label='Facebook')
plt.plot(months, rates['GOOG'],
         color='r', linestyle='-.', linewidth=1.5, marker='o',
         label='Google')
plt.xlabel('Months')
plt.ylabel('Rate of return (%)')
plt.legend(fontsize=12)
plt.show()
```

Matplotlib for Data Visualization

- Configuration of plots and figures
  - Specify plot features by keyword arguments

```
plt.plot(months, rates['FB'],
         color='b', linestyle='--', linewidth=1.5, marker='s',
         label='Facebook')
plt.plot(months, rates['GOOG'],
         color='r', linestyle='-.', linewidth=1.5, marker='o',
         label='Google')
plt.xlabel('Months')
plt.ylabel('Rate of return (%)')
plt.legend(fontsize=12)
plt.show()
```

Blue dash line, with the width to be 1.5, and square markers added

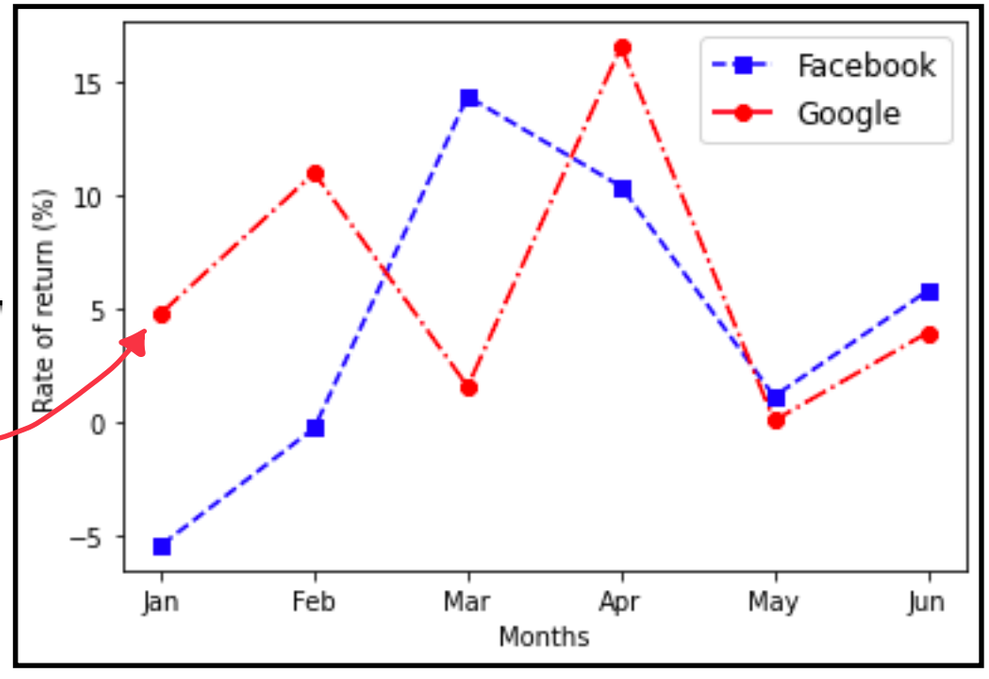


Matplotlib for Data Visualization

- Configuration of plots and figures
  - Specify plot features by keyword arguments

```
plt.plot(months, rates['FB'],
         color='b', linestyle='--', linewidth=1.5, marker='s',
         label='Facebook')
plt.plot(months, rates['GOOG'],
         color='r', linestyle='-.', linewidth=1.5, marker='o',
         label='Google')
plt.xlabel('Months')
plt.ylabel('Rate of return (%)')
plt.legend(fontsize=12)
plt.show()
```

Red dash-dot line, with the width to be 1.5, and circle markers added

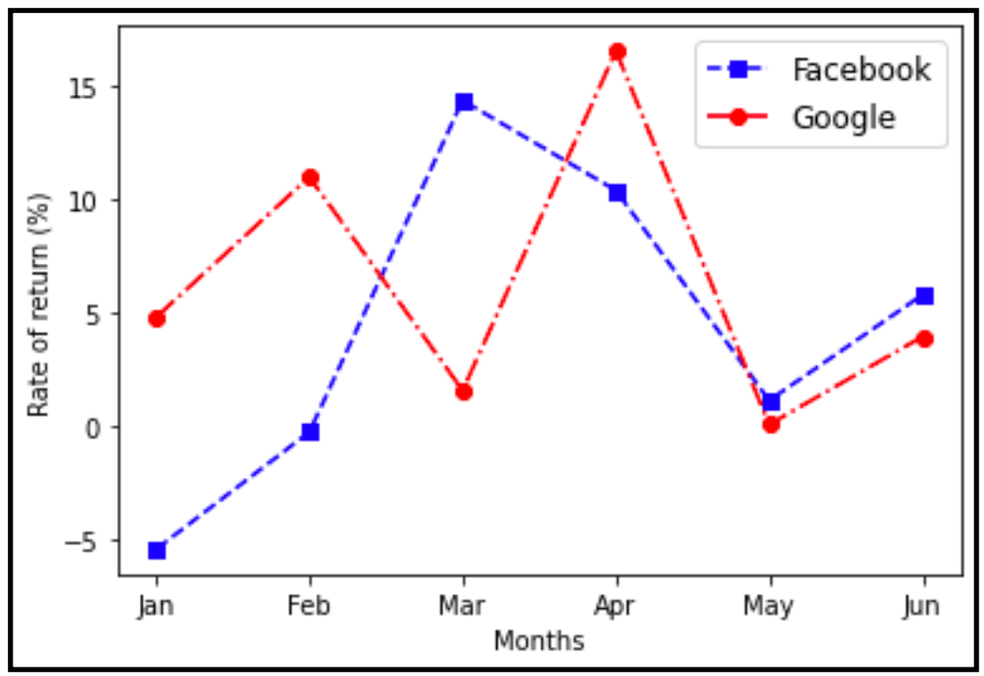


Matplotlib for Data Visualization

- Configuration of plots and figures
  - Specify plot features by keyword arguments

```
plt.plot(months, rates['FB'],
         color='b', linestyle='--', linewidth=1.5, marker='s',
         label='Facebook')
plt.plot(months, rates['GOOG'],
         color='r', linestyle='-.', linewidth=1.5, marker='o',
         label='Google')
plt.xlabel('Months')
plt.ylabel('Rate of return (%)')
plt.legend(fontsize=12)
plt.show()
```

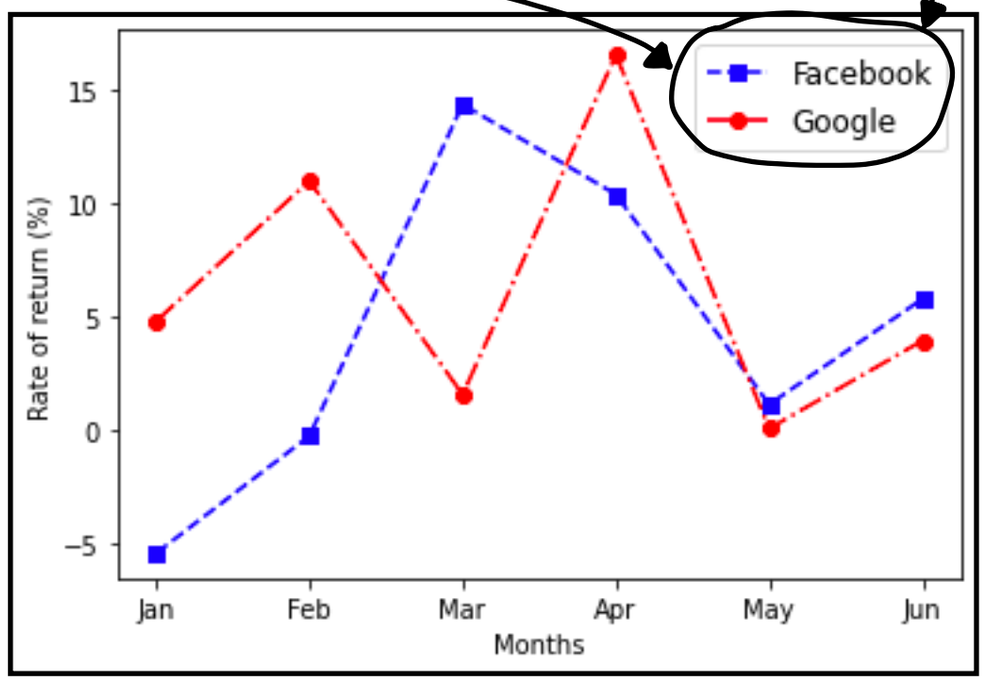
Create the plot legend



Matplotlib for Data Visualization

- Configuration of plots and figures
  - Specify plot features by keyword arguments

```
plt.plot(months, rates['FB'],
         color='b', linestyle='--', linewidth=1.5, marker='s',
         label='Facebook')
plt.plot(months, rates['GOOG'],
         color='r', linestyle='-.', linewidth=1.5, marker='o',
         label='Google')
plt.xlabel('Months')
plt.ylabel('Rate of return (%)')
plt.legend(fontsize=12)
plt.show()
```



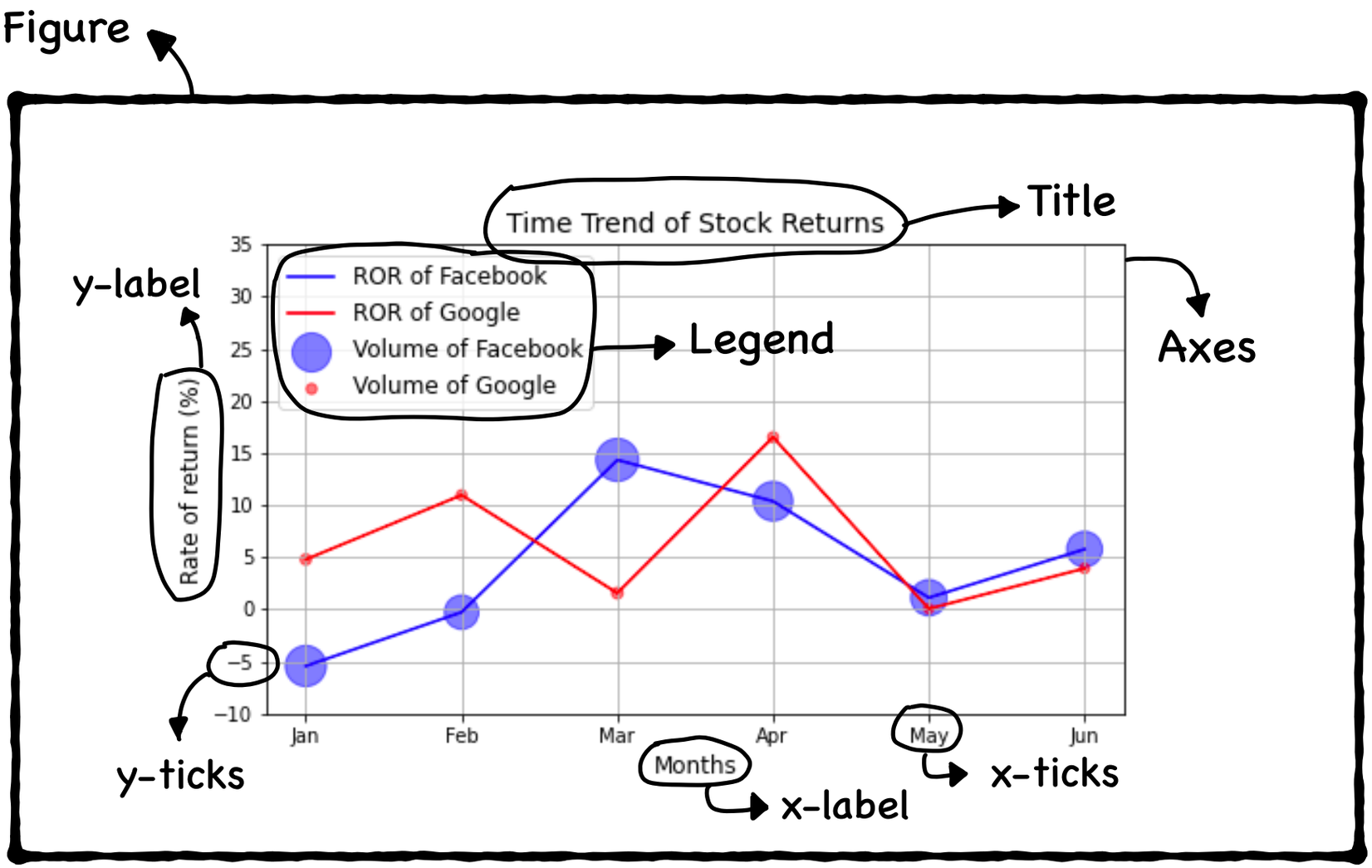
Matplotlib for Data Visualization

- Configuration of plots and figures
  - Specify plot features by keyword arguments

Plot feature	Argument keyword	Plot function
Color	`color`	`plot()`, `bar()`, `scatter()`
Opacity	`alpha`	`plot()`, `bar()`, `scatter()`
Marker shape	`marker`	`plot()`, `scatter()`
Line style	`linestyle`	`plot()`
Line width	`linewidth`	`plot()`
Bar width	`width`	`bar()`

Matplotlib for Data Visualization

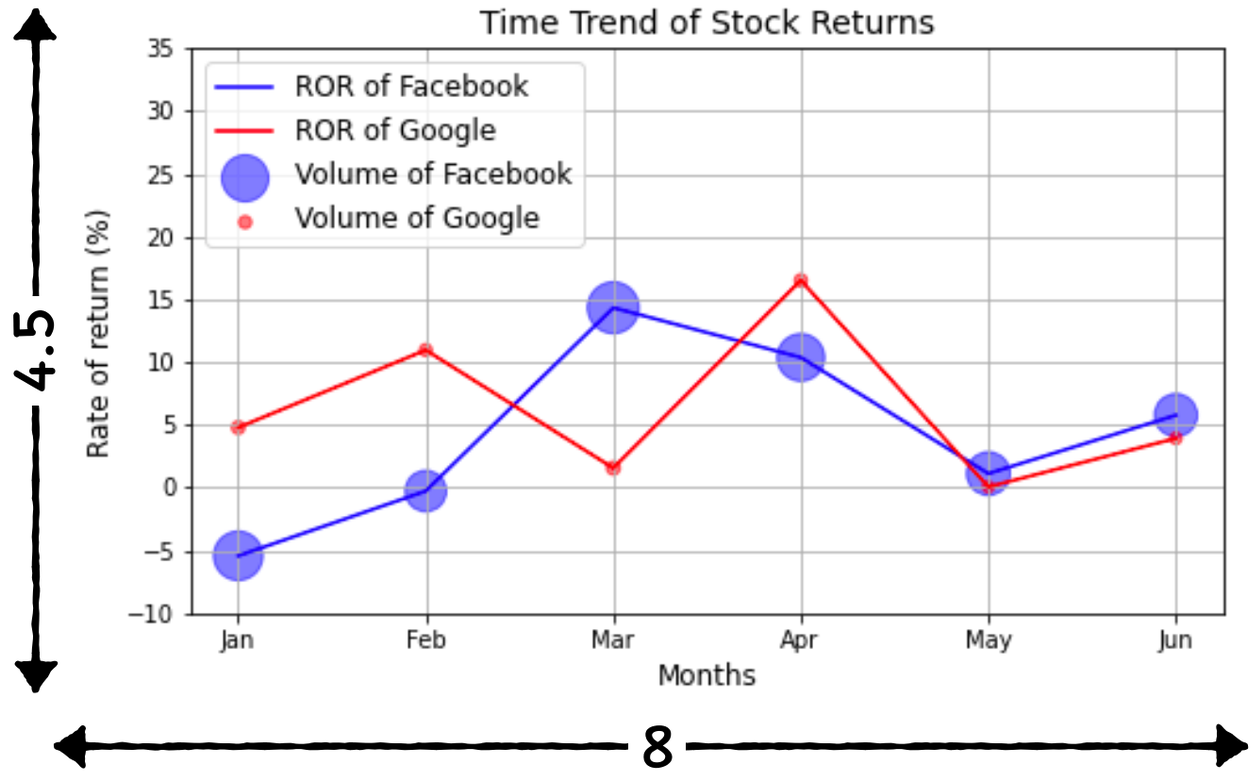
- Configuration of plots and figures
  - Features of figures



Matplotlib for Data Visualization

- Configuration of plots and figures
  - Specify figure features by keyword arguments

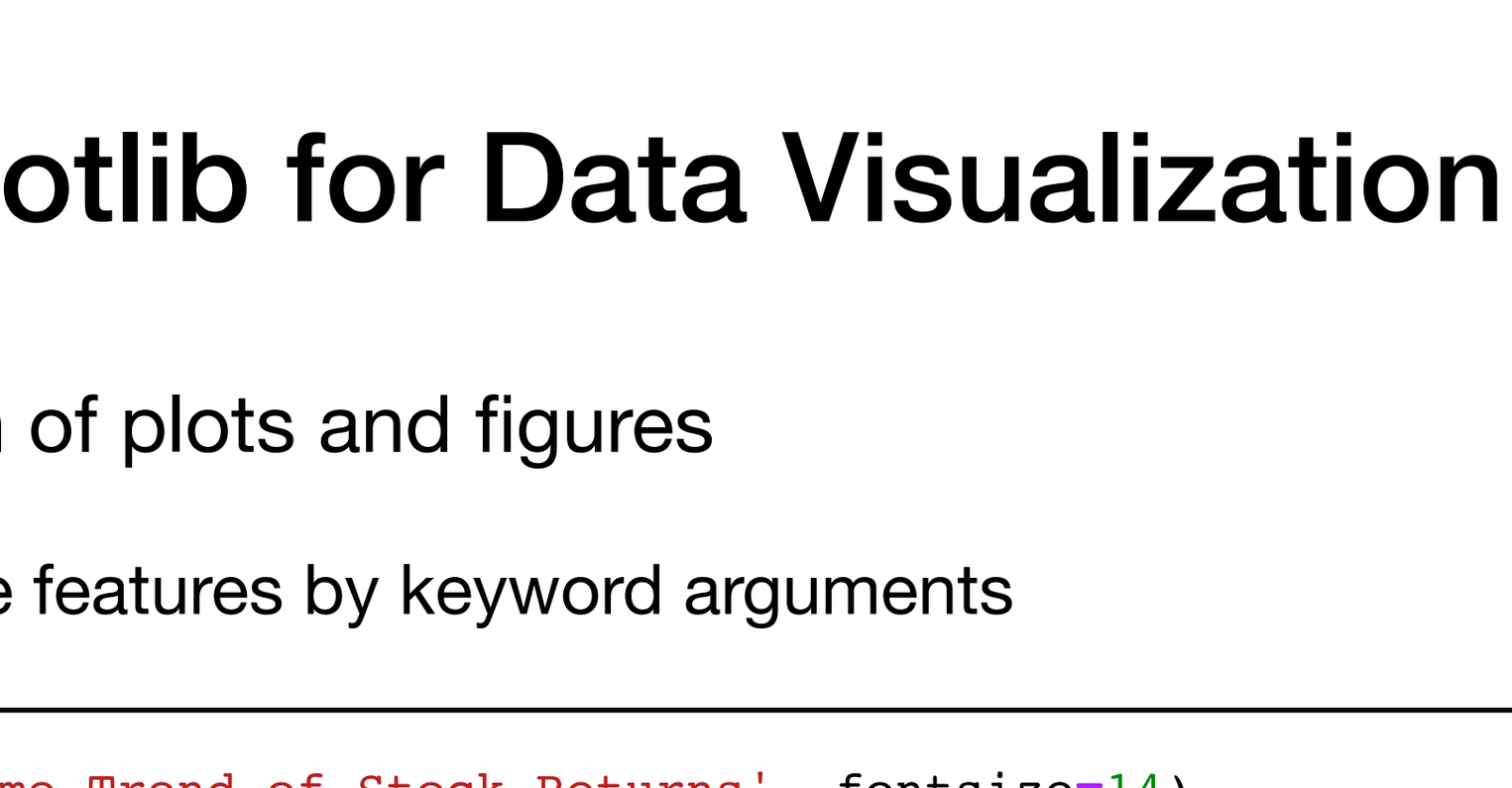
```
plt.figure(figsize=(8, 4.5))
```



Matplotlib for Data Visualization

- Configuration of plots and figures
  - Specify figure features by keyword arguments

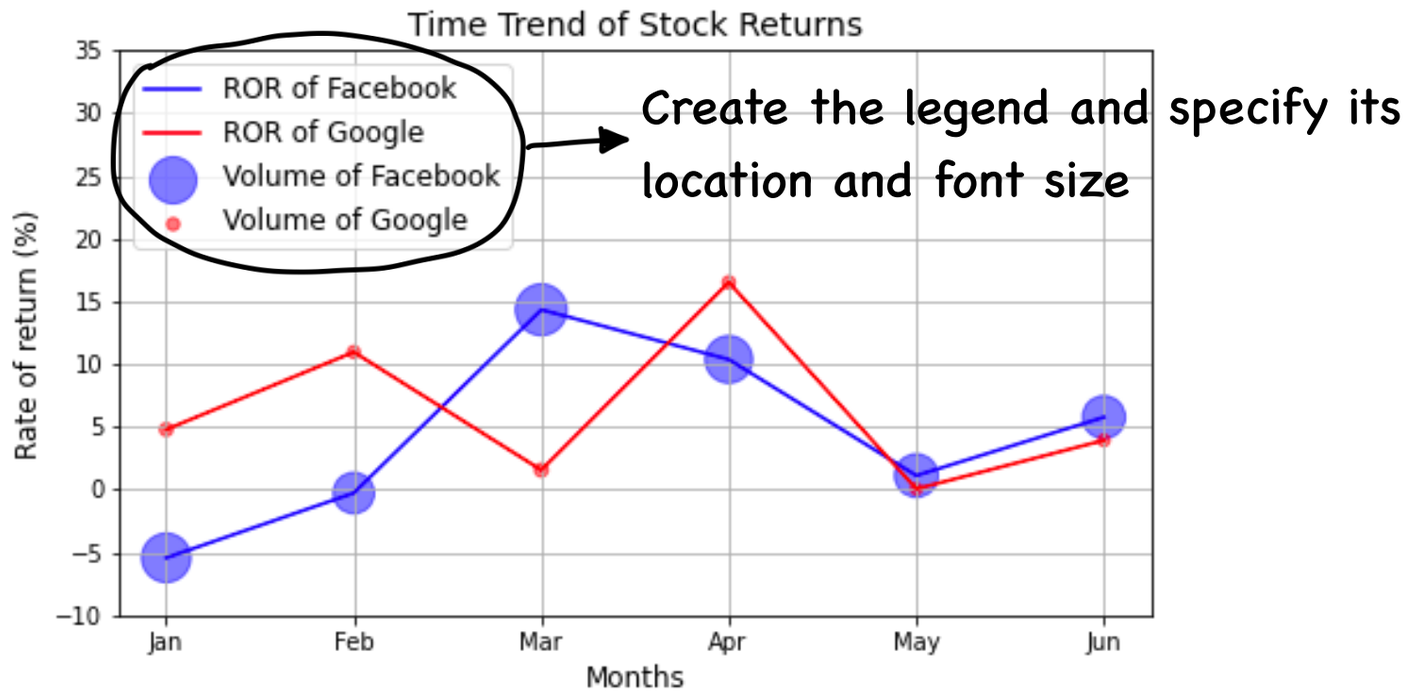
```
plt.title('Time Trend of Stock Returns', fontsize=14)
```



Matplotlib for Data Visualization

- Configuration of plots and figures
  - Specify figure features by keyword arguments

```
plt.legend(loc='upper left', fontsize=12)
```

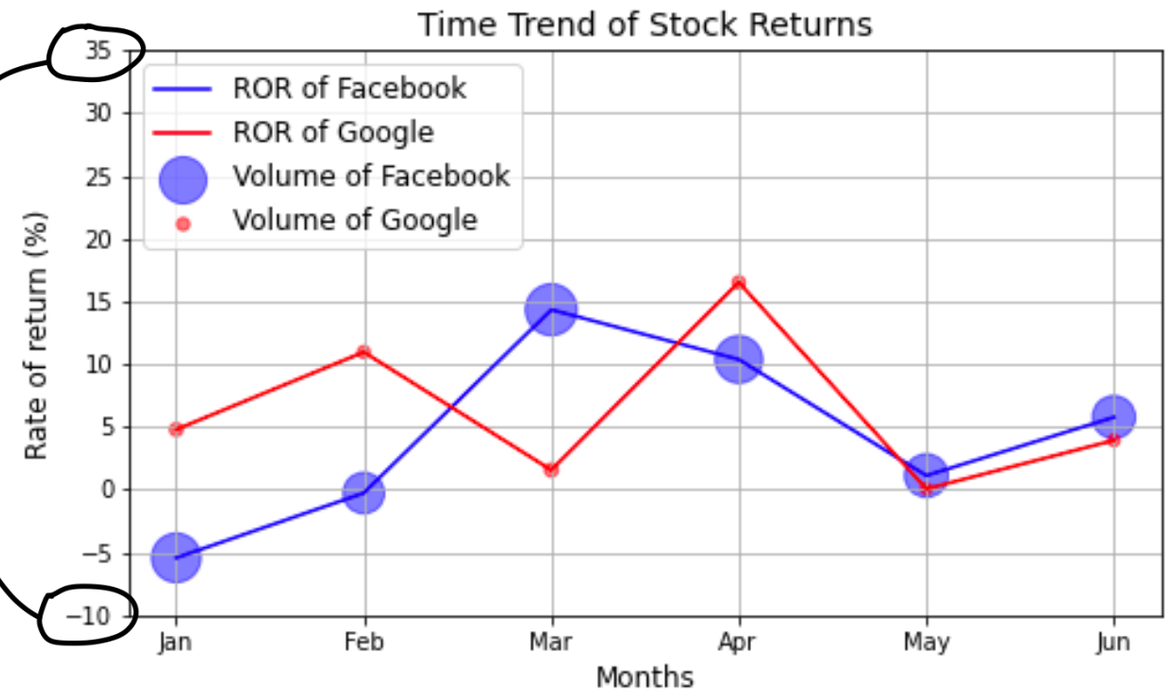


Matplotlib for Data Visualization

- Configuration of plots and figures
  - Specify figure features by keyword arguments

```
plt.ylim([-10, 35])
```

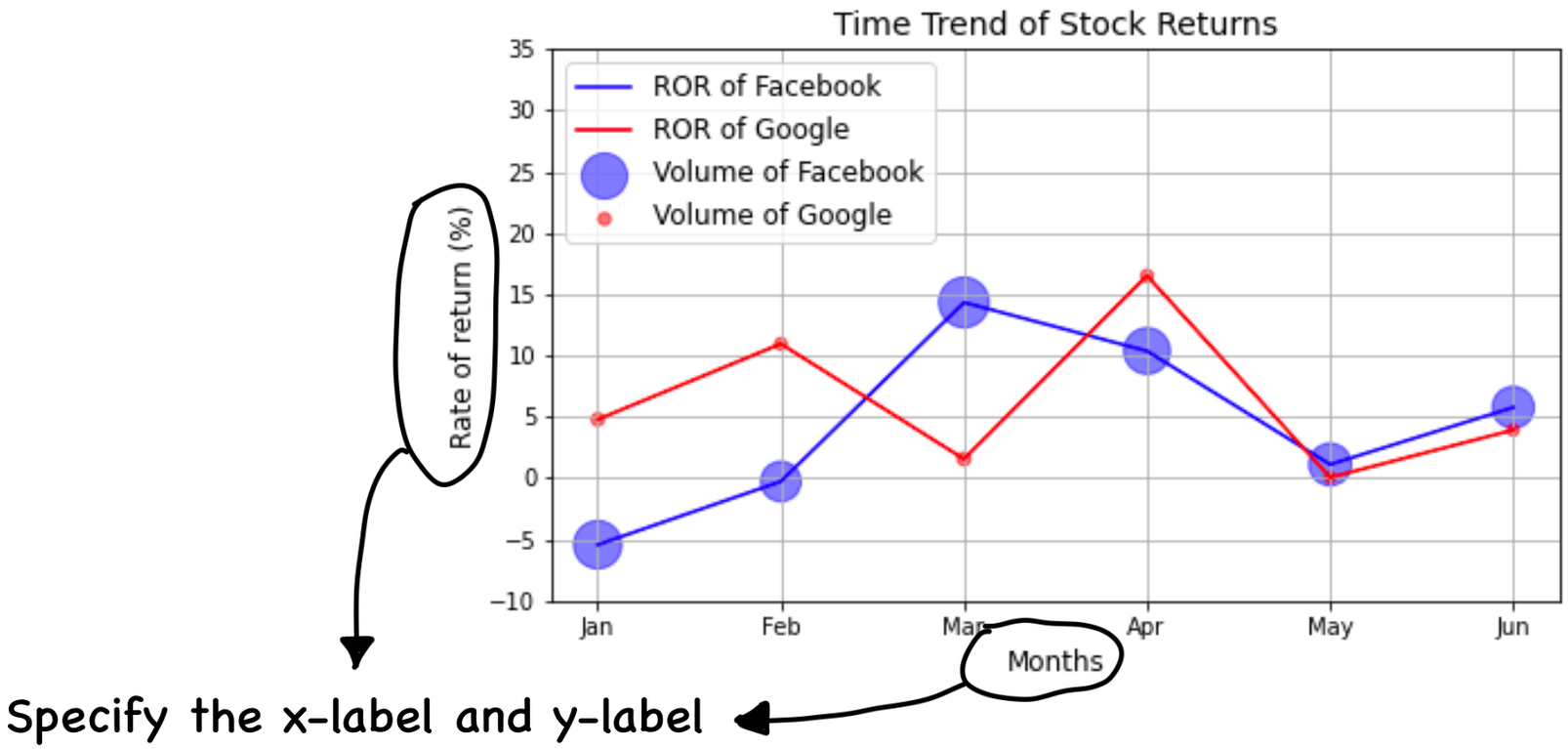
Specify the y-limits of the axes



Matplotlib for Data Visualization

- Configuration of plots and figures
  - Specify figure features by keyword arguments

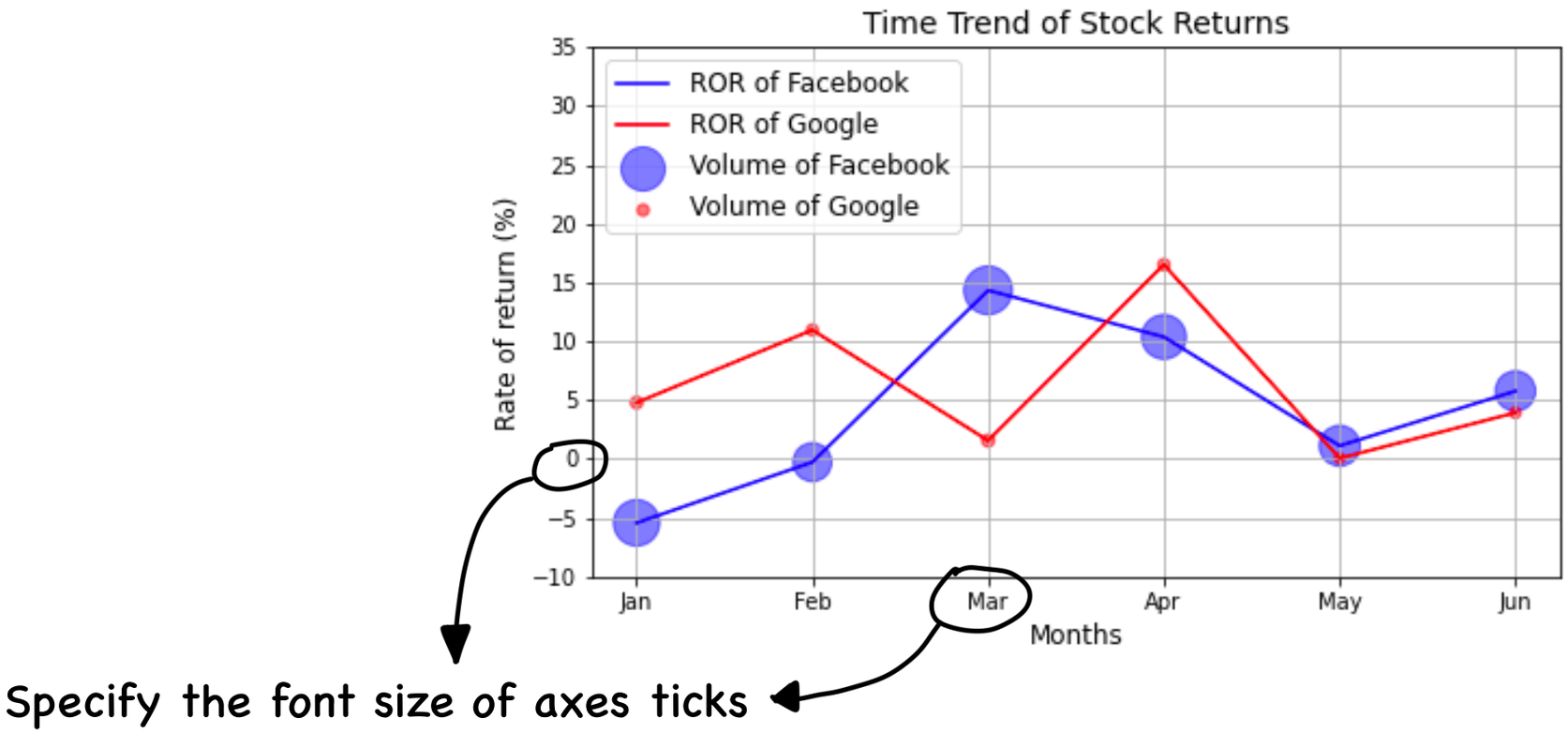
```
plt.xlabel('Months', fontsize=12)
plt.ylabel('Rate of return (%)', fontsize=12)
```



Matplotlib for Data Visualization

- Configuration of plots and figures
  - Specify figure features by keyword arguments

```
plt.xticks(fontsize=10)
plt.yticks(fontsize=10)
```



Matplotlib for Data Visualization

- Configuration of plots and figures
  - Specify figure features by keyword arguments

```
plt.grid()
```