INF1002
Programming Fundamentals
Lecture 4: Functions

Zhang Zhengchen

zhengchen.zhang@singaporetech.edu.sg

Early Learning Insight Survey (ELIS)

- To find out how you have been coping with your modules since the start of the trimester
- From 16 Sep 2024 to 29 Sep 2024, 9 PM
- To collect your learning needs early
- Access the survey via
  1. Individualized link in the email sent to all
  2. xSiTe
  3. URL Link - https://singaporetech.bluera.com/singaporetech
  4. QR Code

ELIS Learning Insight Survey AY24/25, Tri 1

Review

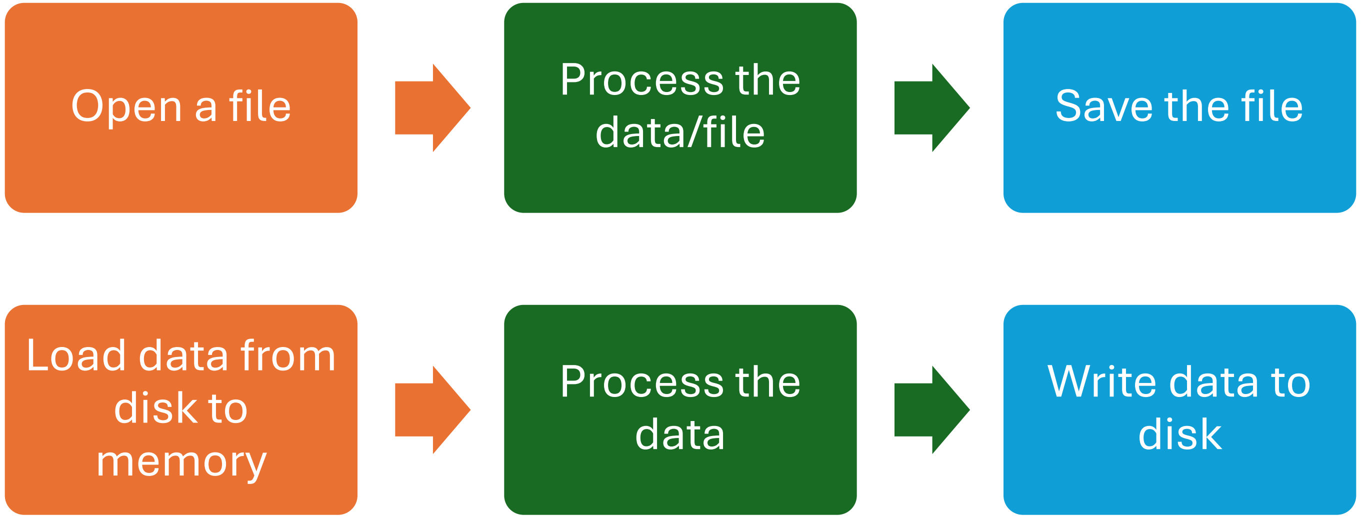
- String, List
  - WADIOOO
  - Sequence
    - Insert, delete, update, query
    - in, not in
    - index, slicing
    - Concatenation and repetition
- **Immutable and mutable**
  - Tuple
  - Shallow Copy and Deep Copy
- Dictionary
  - Key-value pair
- List Comprehension

Outline

- File I/O
  - read
  - write
- Function
  - Default arguments
  - Positional and keyword arguments
  - Will a Variable's Value Change After a Function Call?
  - Variable scope
- Modules
- Higher-order functions

Files I/O

- Why do we need a file?
  - Data used in the program is stored in the memory
  - Still can find the data after reboot the computer/restart the program
  - Keep the data permanent
- File operations

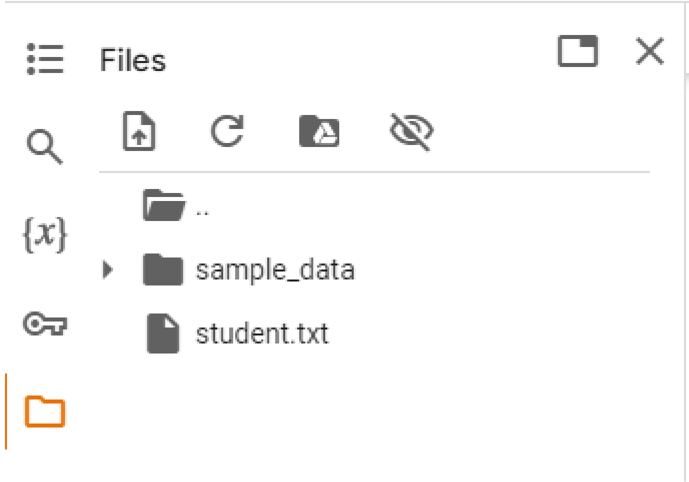


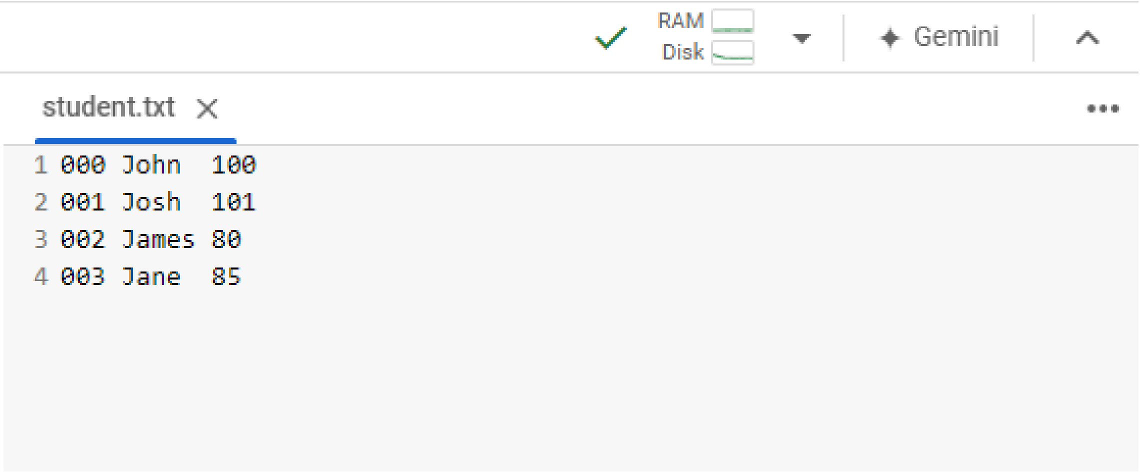
Open a file → Process the data/file → Save the file

Load data from disk to memory → Process the data → Write data to disk

Practice

- Create a new file on Google Colab
- Double click 'student.txt' and edit it, and save it (Ctrl + S)





```
000 John  100
001 Josh  101
002 James 80
003 Jane  85
```

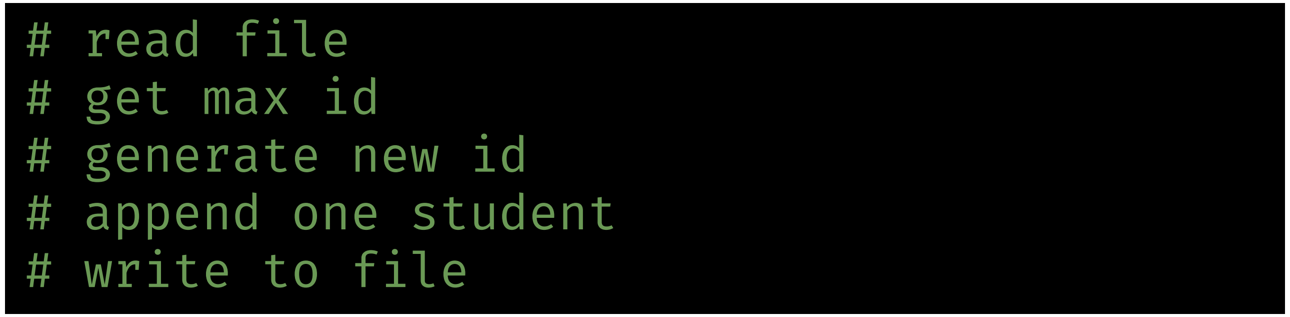
%cd /content/drive/MyDrive

Sample file

```
000 John 100
001 Josh 101
002 James 89
003 Jane 95
```

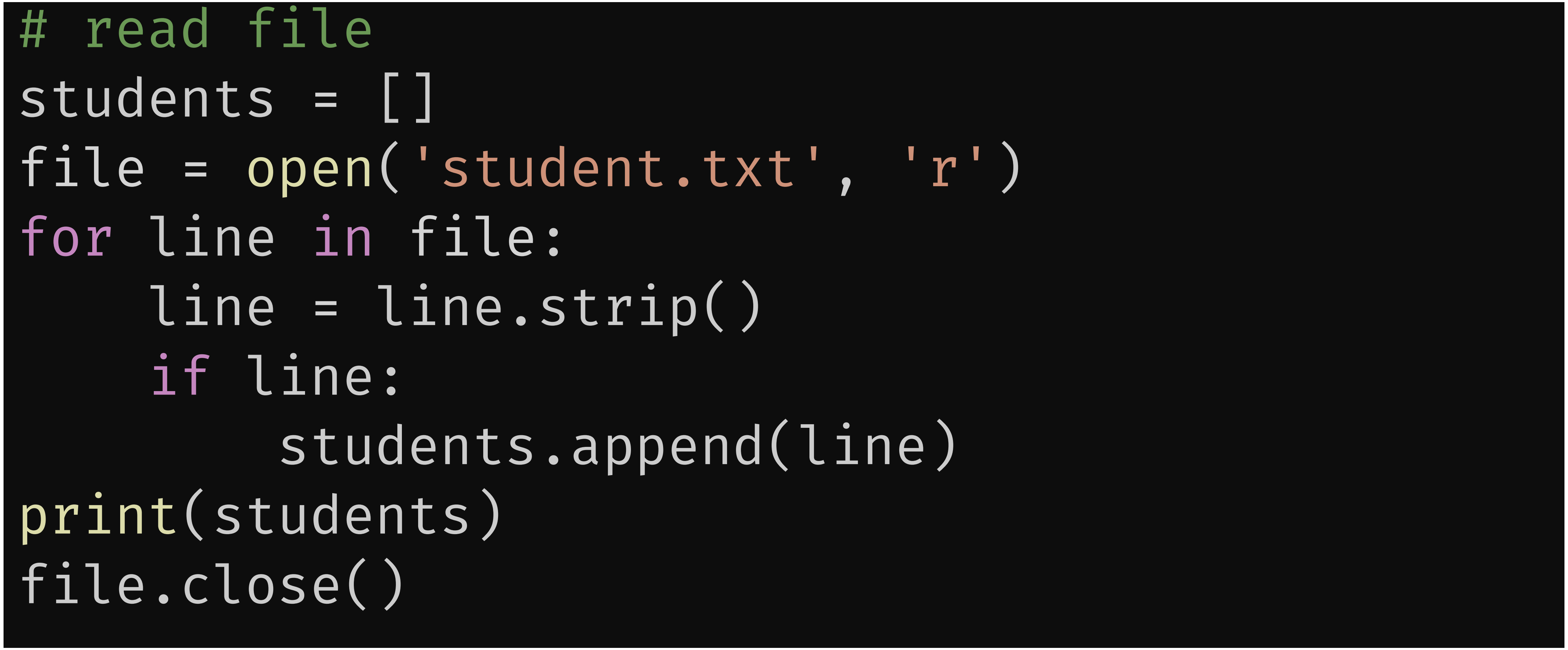
Practice

1. Append a new student: Jack 99
2. The id of the new student is **maximum_id + 1**: 004
3. Save data to the file



```
# read file
# get max id
# generate new id
# append one student
# write to file
```

Practice



```python
# read file
students = []
file = open('student.txt', 'r')
for line in file:
    line = line.strip()
    if line:
        students.append(line)
print(students)
file.close()
```

Open a file

- Python's built-in open() function
- Create one file object that can be utilized
- Syntax:
  - `file = open('student.txt', 'r')`
  - file_object= open(file_name[, access_mode] [, encoding])

open()

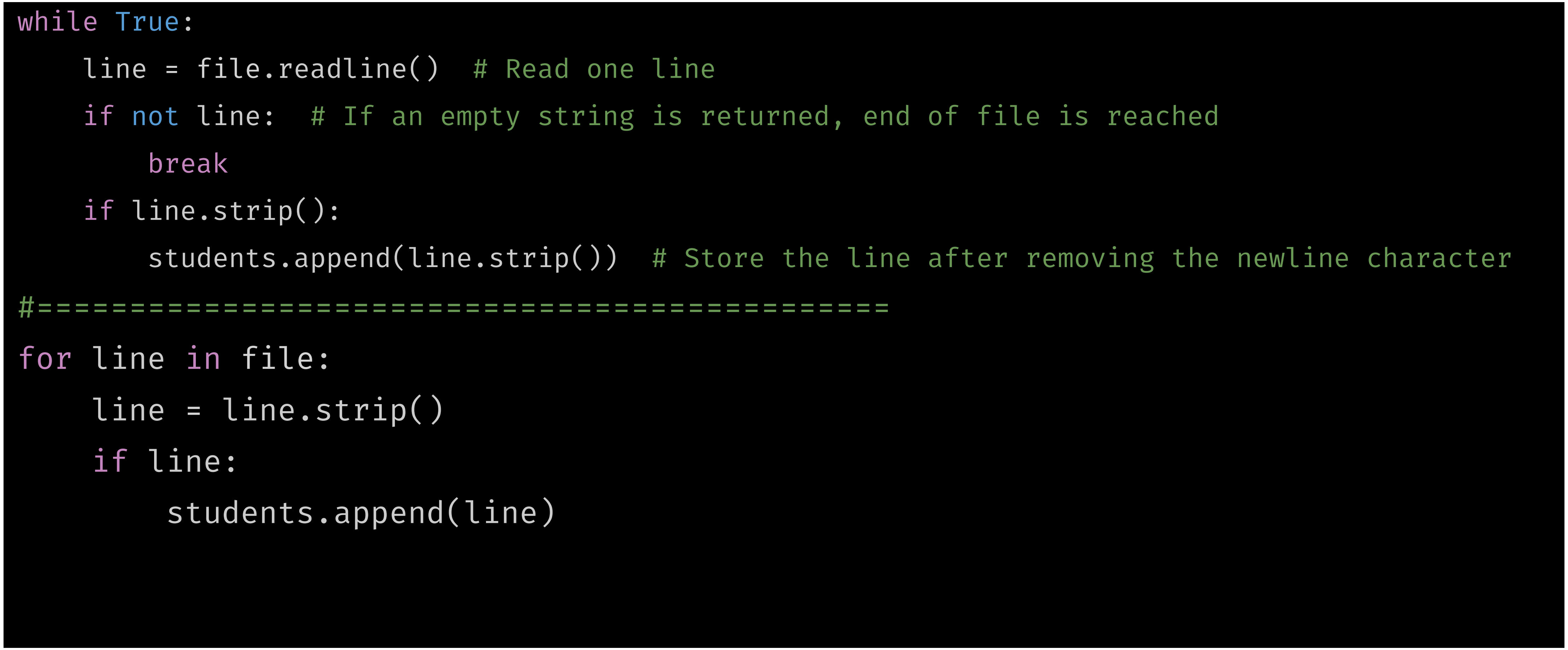
- `open('student.txt', 'r')`
- File_name
  - a string value that contains the name of the file that you want to access
- Access_mode
  - the mode in which the file to be opened, i.e., read, write, append, etc.
  - Optional, the default model is read(r)
- Encoding
  - Default value depends on your operating system
  - utf-8
  - Remember the Unicode thing?

mode

- Open('student_info.txt', mode='r')

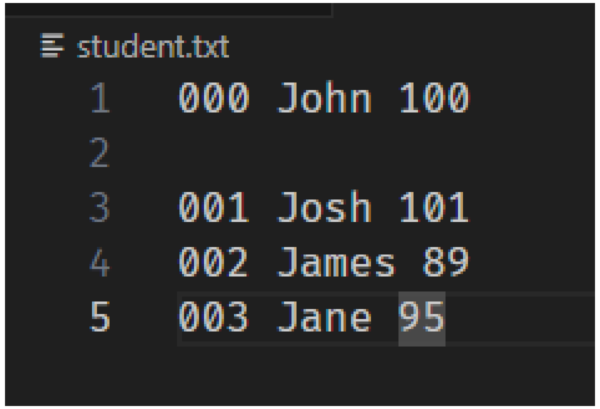
Mode	Description
r	Opens a file for reading only. This is the default mode.
r+	Opens a file for both reading and writing.
w	Opens a file for writing only. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.
w+	Opens a file for both writing and reading. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.
a	Opens a file for appending. If the file does not exist, it creates a new file for writing.
b	rb, wb, ab, etc. Opens the file in binary mode.
…	

readline() vs for line in file



```
while True:
    line = file.readline()  # Read one line
    if not line:  # If an empty string is returned, end of file is reached
        break
    if line.strip():
        students.append(line.strip())  # Store the line after removing the newline character
#==========================================
for line in file:
    line = line.strip()
    if line:
        students.append(line)
```

Practice



student.txt

```
000 John 100

001 Josh 101
002 James 89
003 Jane 95
```

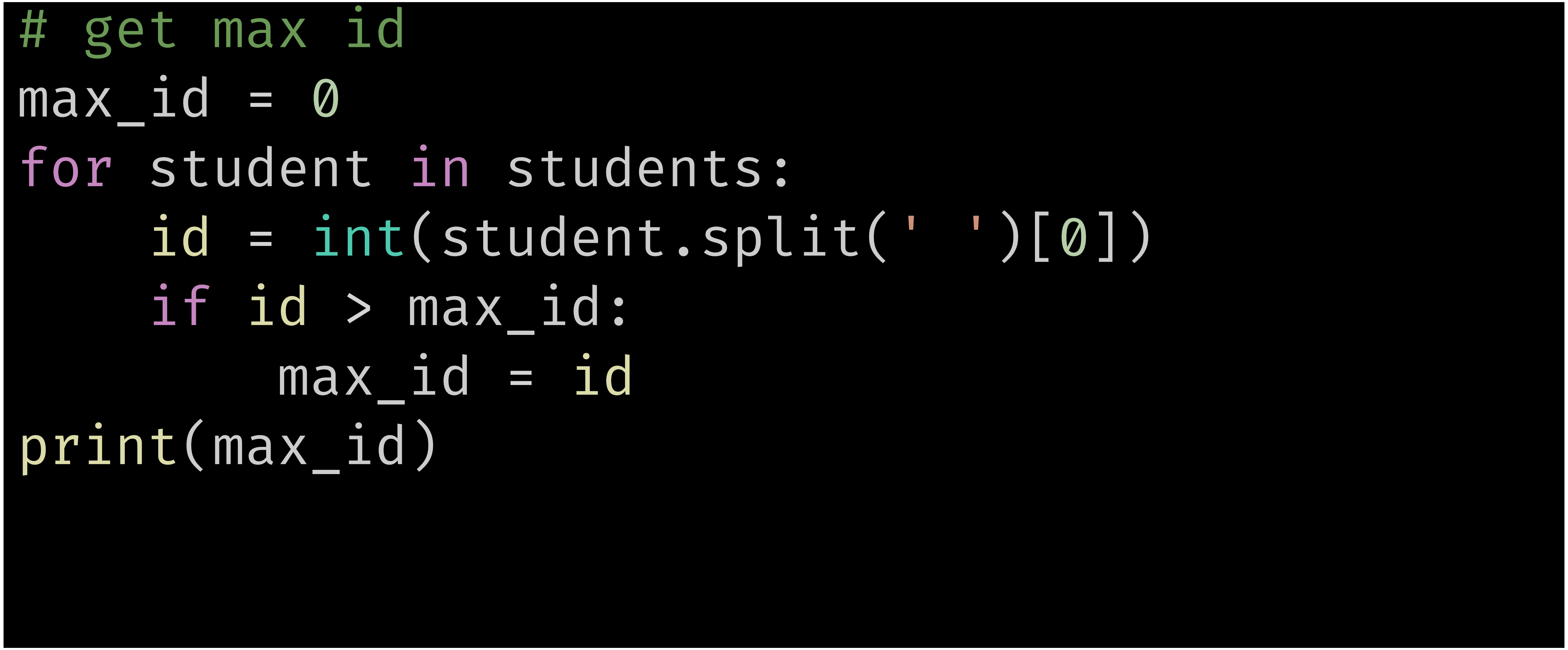
```
#students = file.readlines()
students = [line.strip() for line in file.readlines() if line.strip()]
```

Read a file

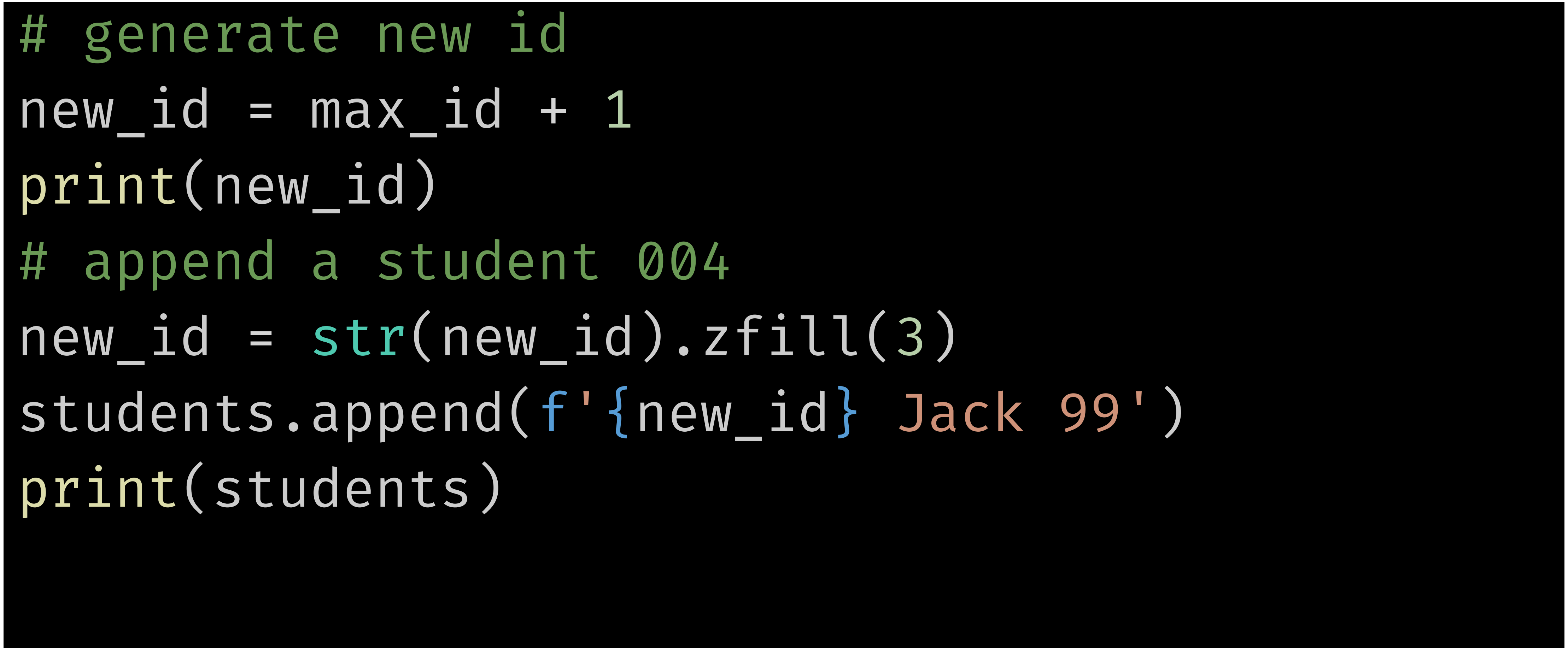
- **readline(*size*)**
  - the *size* parameter specifies the maximum number of characters to read from the line, **strictly limiting the read to that number or until the end of the line is reached.**
- **readlines(*hint*)**
  - returns a list containing each line in the file as a list item.
  - the *hint* parameter suggests the approximate number of characters to read, but it doesn't strictly limit the total number, **ensuring complete lines are still returned**.
- read(size)

```
students = []
file = open('student.txt', 'r')
line = file.readline(3)
students.append(line)
print(students)
file.close()
exit()
```

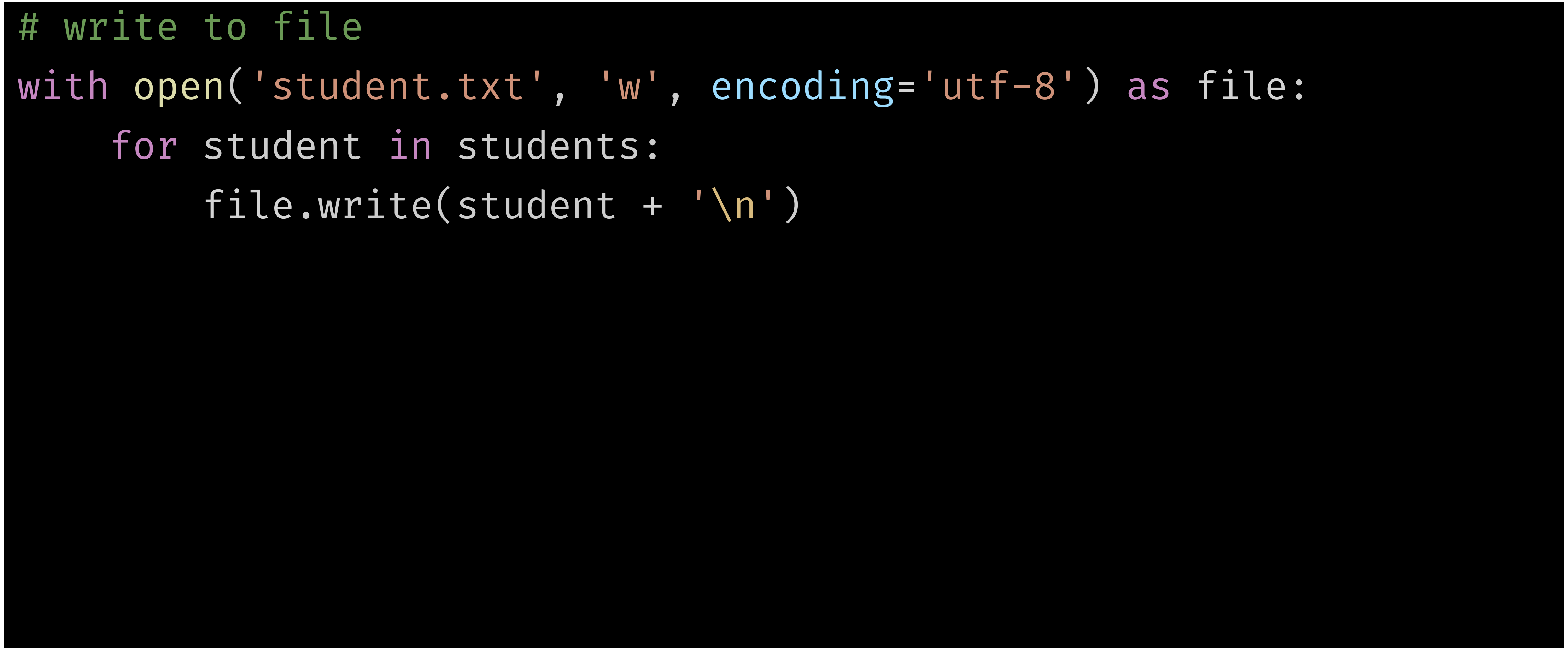
```
students = []
file = open('student.txt', 'r')
students = file.readlines(3)
print(students)
file.close()
```



```
# get max id
max_id = 0
for student in students:
    id = int(student.split(' ')[0])
    if id > max_id:
        max_id = id
print(max_id)
```



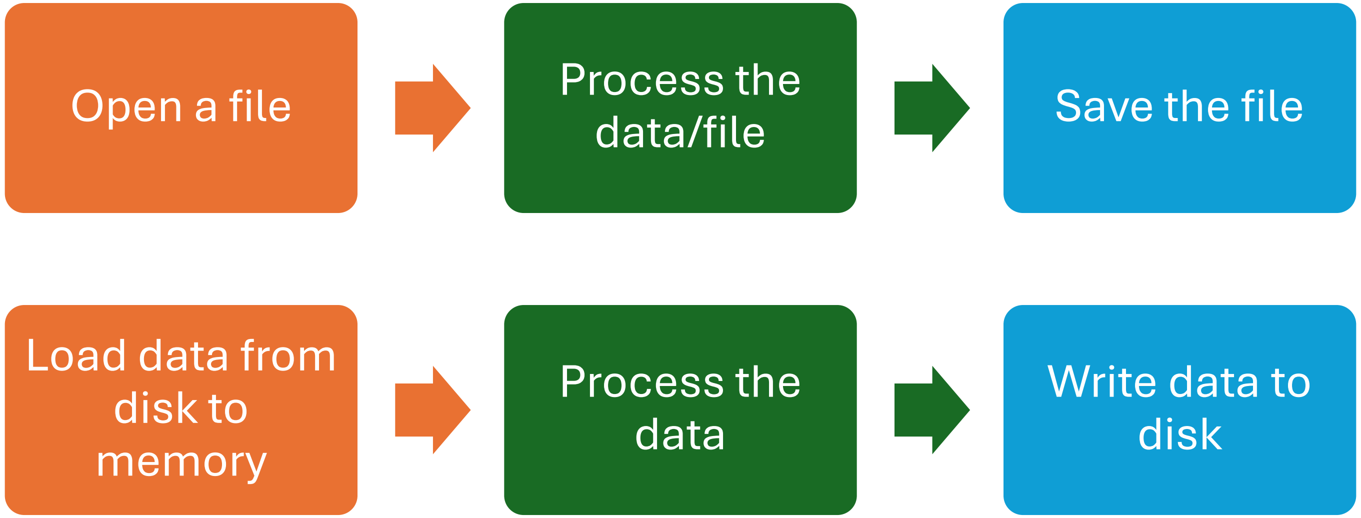
```python
# generate new id
new_id = max_id + 1
print(new_id)
# append a student 004
new_id = str(new_id).zfill(3)
students.append(f'{new_id} Jack 99')
print(students)
```



```python
# write to file
with open('student.txt', 'w', encoding='utf-8') as file:
    for student in students:
        file.write(student + '\n')
```

Write data to a file

- write(str)
- writelines(sequence)
  - Whether the lines include newline characters.



Open a file

Process the data/file

Save the file

Load data from disk to memory

Process the data

Write data to disk

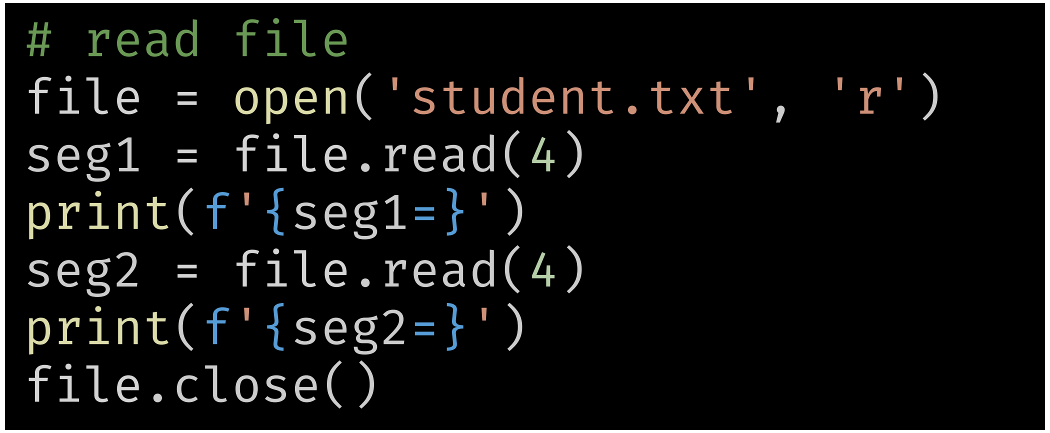
Append mode

- Open('student_info.txt', mode='r')

Mode	Description
r	Opens a file for reading only. This is the default mode.
r+	Opens a file for both reading and writing.
w	Opens a file for writing only. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.
w+	Opens a file for both writing and reading. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.
a	**Opens a file for appending. If the file does not exist, it creates a new file for writing.**
b	rb, wb, ab, etc. Opens the file in binary mode.
…	

File pointer

- Indicate the current focused position of the file
  - open(xxx, mode='r')
  - The file pointer is placed at the beginning of the file.



```
# read file
file = open('student.txt', 'r')
seg1 = file.read(4)
print(f'{seg1=}')
seg2 = file.read(4)
print(f'{seg2=}')
file.close()
```

With open() as file:

- The with statement is a context manager
- Automatically handles file opening and closing
- Ensures the file is properly closed even if an exception occurs
- Restricts the variable scope to the with block

```
file = open('student.txt', 'r')
file.close()

with open('student.txt', 'r') as file:
with open('input.txt', 'r') as infile, open('output.txt', 'w') as outfile:
```

The 'with' statement

- Advantages of the with statement:
  - More concise
  - Automatic resource management, avoiding resource leaks
  - Safer, more readable code
  - Restricts the variable scope, reducing potential bugs
- Disadvantages of the traditional method:
  - Requires explicit file closing
  - Error-prone, can lead to resource leaks
  - Variable remains in scope, which may cause unexpected issues

Review

- Read data from a file
- Operations on the data
- Write to a file
- File Pointer
- 'with' statement

Open a file

- Python’s built-in open() function
- Create one file object that can be utilized
- Syntax:
  - `file = open('student.txt', 'r')`
  - file_object= open(file_name[, access_mode] [, encoding])

Object Oriented Programming

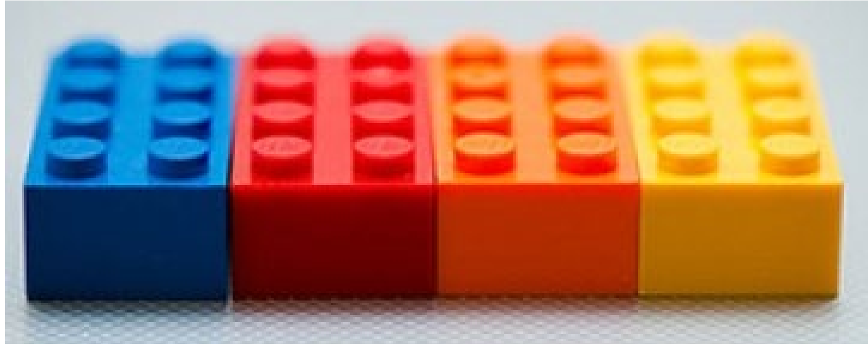
- Class
  - A blueprint for creating objects
  - Defines a set of attributes and methods
  - Groups data and behavior together in a reusable and organized way
- Instance
  - Attributes (or Properties): Variables that store data specific to the object
    - Name, Age
  - Methods (or Functions): Functions that define the behavior of the object
    - Walk, Speak
- Abstraction
- Encapsulation, Inheritance, Polymorphism, ...

Outline

- File I/O
  - read
  - write
- **Function**
  - Default arguments
  - Positional and keyword arguments
  - Will a Variable's Value Change After a Function Call?
  - Variable scope
- Modules
- Higher-order functions

Function

- A Python function is a block of organized, reusable code that is used to perform a single, related action.
  - Functions of a smartphone
    - Make a phone call
    - Calculator
    - Play games



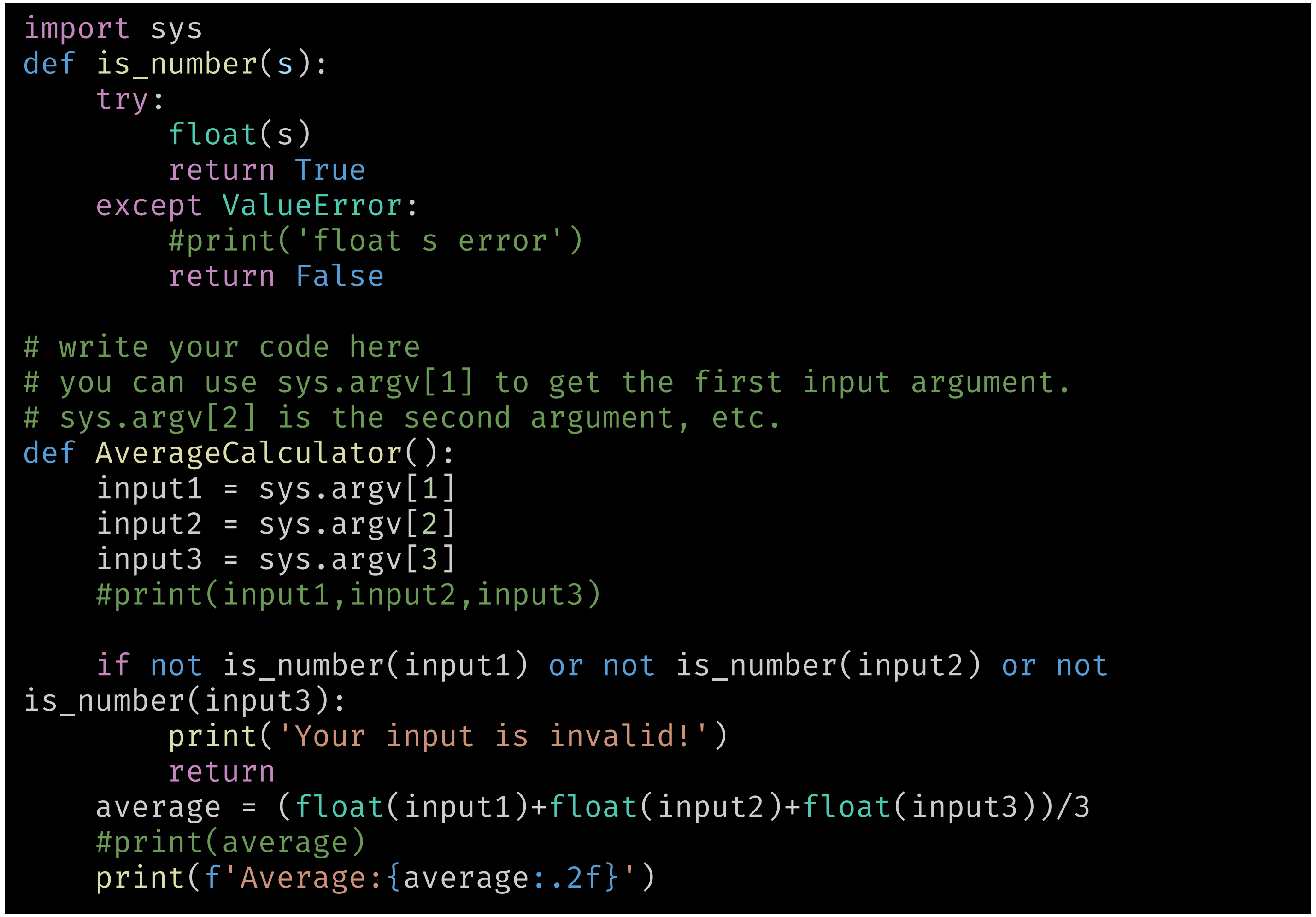
https://hub.jhu.edu/2016/03/07/lego-blocks-build-better-thinkers/

https://www.tutorialspoint.com/python/python_functions.htm

function definition

```
def function_name( arguments ):
    '''function_docstring'''
    function_suite
    return [expression]
```

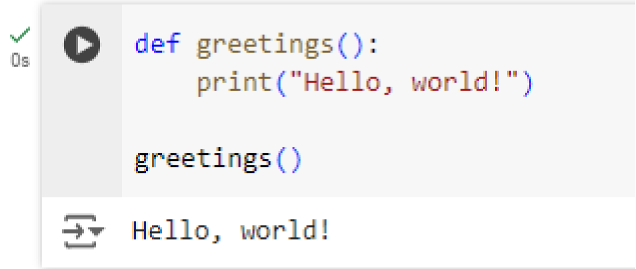
- Function_name
  - The identifier of the function
- Arguents
  - Input
  - Formal Arguments
  - Actual Arguments
- Function_docstring
  - Comment your codes
  - Optional
- Function_suite
  - Block of code
  - indented
- Return value
  - Optional
  - Exits a function
  - Passing back an expression to the caller
  - Output



```
import sys
def is_number(s):
    try:
        float(s)
        return True
    except ValueError:
        #print('float s error')
        return False

# write your code here
# you can use sys.argv[1] to get the first input argument.
# sys.argv[2] is the second argument, etc.
def AverageCalculator():
    input1 = sys.argv[1]
    input2 = sys.argv[2]
    input3 = sys.argv[3]
    #print(input1,input2,input3)

    if not is_number(input1) or not is_number(input2) or not 
is_number(input3):
        print('Your input is invalid!')
        return
    average = (float(input1)+float(input2)+float(input3))/3
    #print(average)
    print(f'Average:{average:.2f}')
```

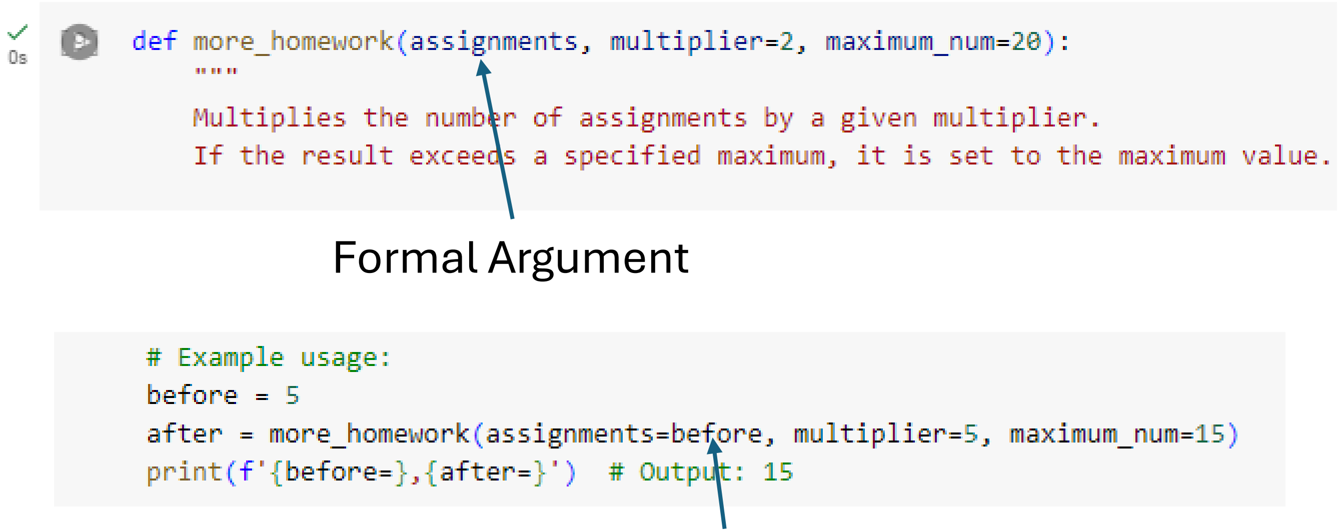
Arguments

- A function can have 0 or multiple arguments
- Formal Arguments
  - A place to get input value
  - A name used inside a function
- Actual Arguments
  - True value
- When you call one function, it must have the same number of arguments as defined



```
def greetings():
    print("Hello, world!")

greetings()
```

```
Hello, world!
```



```
def more_homework(assignments, multiplier=2, maximum_num=20):
    """
    Multiplies the number of assignments by a given multiplier.
    If the result exceeds a specified maximum, it is set to the maximum value.
```

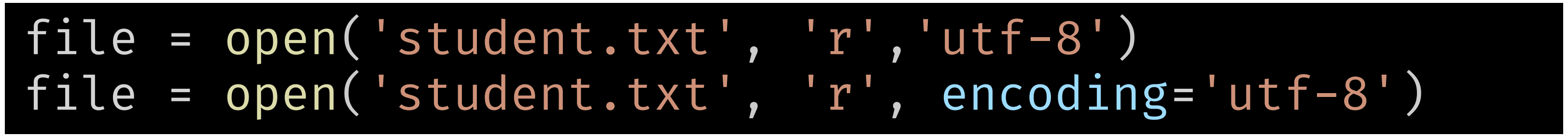
Formal Argument

```
# Example usage:
before = 5
after = more_homework(assignments=before, multiplier=5, maximum_num=15)
print(f'{before=},{after=}')  # Output: 15
```

Actual Argument

Positional and Keyword Arguments

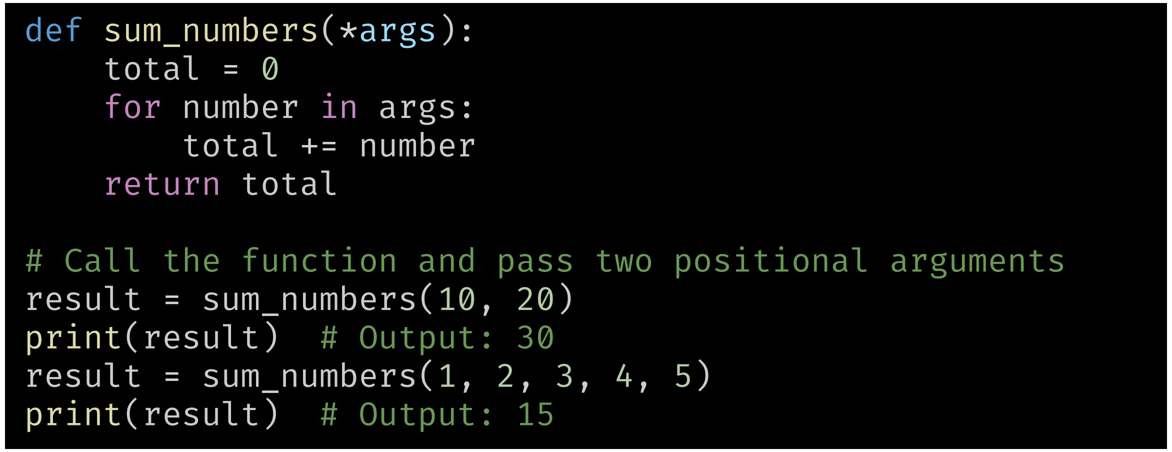
- Positional Arguments
  - Passed to the function in the order in which they are defined.
  - Order matters, arguments are matched by position.
- Keyword Arguments
  - Passed to the function by explicitly naming each parameter.
  - Order doesn't matter, parameters are matched by name.
- Keyword-Only Arguments
  - To enhance the **robustness** and **readability** of the program
  - Especially when there are many parameters



```
file = open('student.txt', 'r','utf-8')
file = open('student.txt', 'r', encoding='utf-8')
```

Positional and Keyword Arguments

- Further Reading
  - *args
    - Allows a function to accept any number of **positional arguments**.
    - Pack positional arguments into a tuple.



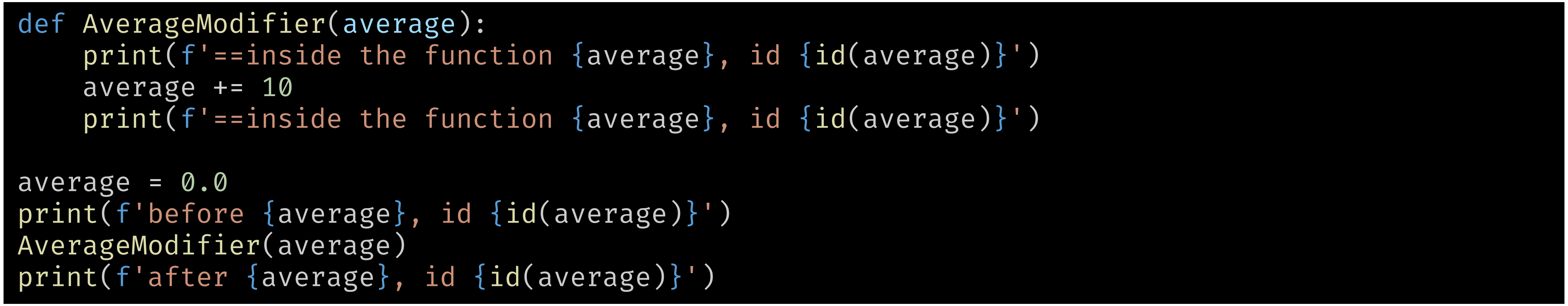
```python
def sum_numbers(*args):
    total = 0
    for number in args:
        total += number
    return total

# Call the function and pass two positional arguments
result = sum_numbers(10, 20)
print(result)  # Output: 30
result = sum_numbers(1, 2, 3, 4, 5)
print(result)  # Output: 15
```

  - **kwargs
    - Allows a function to accept any number of **keyword arguments**.
    - Pack keyword arguments into a dictionary.

https://www.tutorialspoint.com/python/python_arbitrary_arguments.htm

Will a Variable's Value Change After a Function Call?

- Immutable variables
  - str, int, float, tuple
  - Immutable variables are not changed after a function call

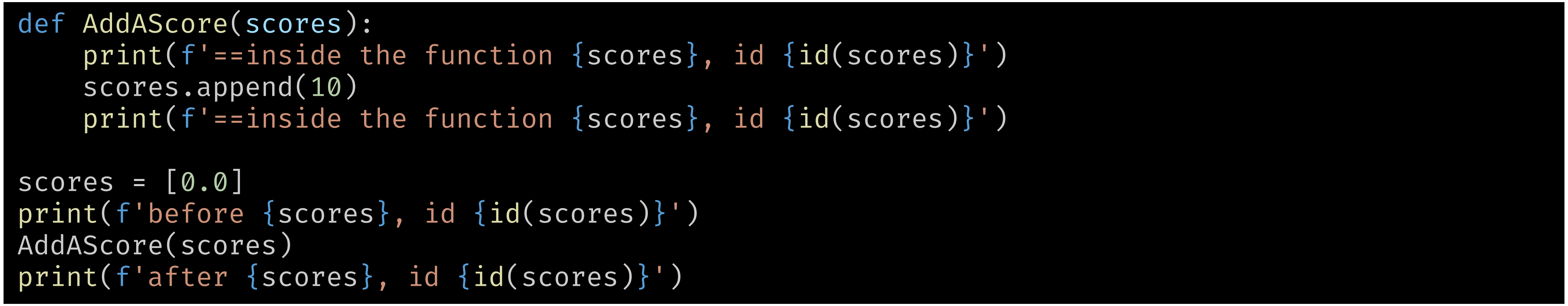


```python
def AverageModifier(average):
    print(f'==inside the function {average}, id {id(average)}')
    average += 10
    print(f'==inside the function {average}, id {id(average)}')

average = 0.0
print(f'before {average}, id {id(average)}')
AverageModifier(average)
print(f'after {average}, id {id(average)}')
```

```
before 0.0, id 2197420871600
==inside the function 0.0, id 2197420871600
==inside the function 10.0, id 2197420578224
after 0.0, id 2197420871600
```

Will a Variable's Value Change After a Function Call?

- Mutable variables
    - list, dictionary
- Mutable variables can be changed as a result of one function call

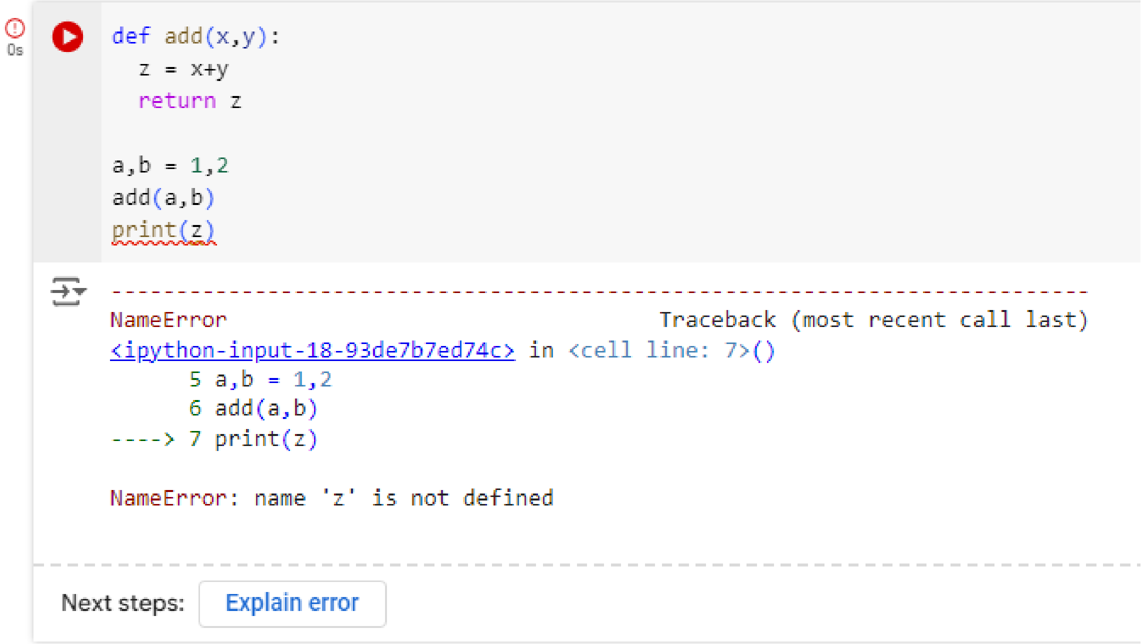


```python
def AddAScore(scores):
    print(f'==inside the function {scores}, id {id(scores)}')
    scores.append(10)
    print(f'==inside the function {scores}, id {id(scores)}')

scores = [0.0]
print(f'before {scores}, id {id(scores)}')
AddAScore(scores)
print(f'after {scores}, id {id(scores)}')
```

```
before [0.0], id 2197429867136
==inside the function [0.0], id 2197429867136
==inside the function [0.0, 10], id 2197429867136
after [0.0, 10], id 2197429867136
```

The scope of the variable

- Local Variable
  - Variables defined within a function can only be accessed and used inside that function.
  - They "come to life" when the function is called and "die" (or go out of scope) when the function execution is completed.

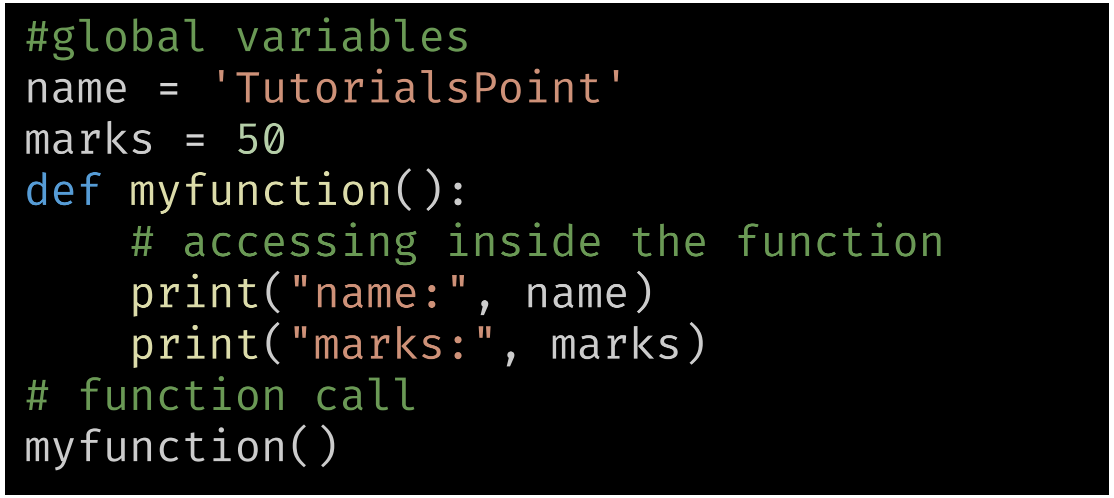


```
def add(x,y):
  z = x+y
  return z

a,b = 1,2
add(a,b)
print(z)
```

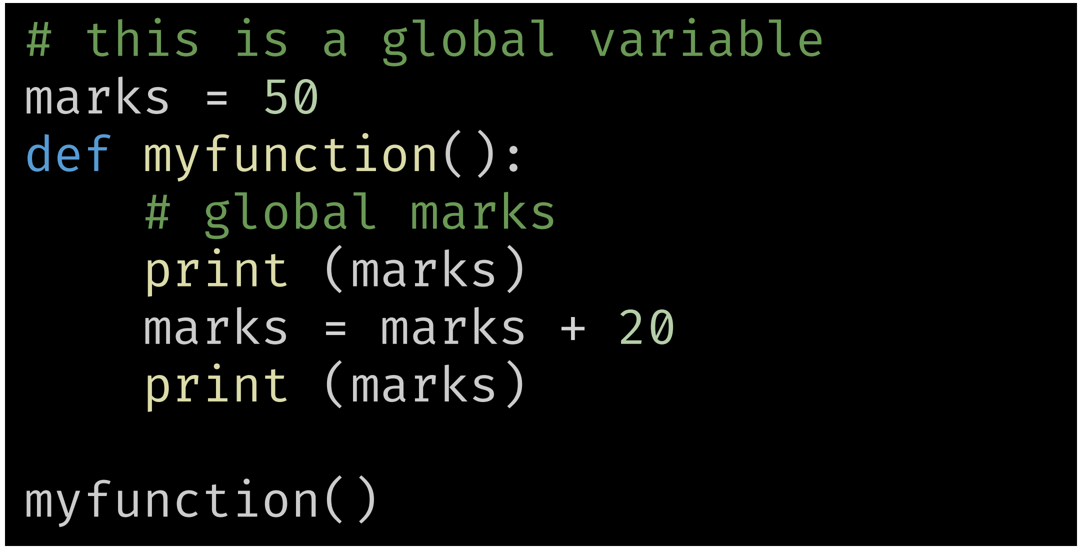
```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-18-93de7b7ed74c> in <cell line: 7>()
      5 a,b = 1,2
      6 add(a,b)
----> 7 print(z)

NameError: name 'z' is not defined
```

The scope of the variable

- Global Variable
- Nonlocal Variables



```python
#global variables
name = 'TutorialsPoint'
marks = 50
def myfunction():
    # accessing inside the function
    print("name:", name)
    print("marks:", marks)
# function call
myfunction()
```



```python
# this is a global variable
marks = 50
def myfunction():
    # global marks
    print (marks)
    marks = marks + 20
    print (marks)

myfunction()
```

UnboundLocalError: local variable 'marks' referenced before assignment

https://www.tutorialspoint.com/python/python_variables_scope.htm

Review

```
def function_name( arguments ):
    statements
```

- Arguments
  - Default arguments
  - Positional and Keyword Arguments
- Immutable variables' value cannot be changed
- Scope of variables
  - Local Variable
  - Global Variable

Outline

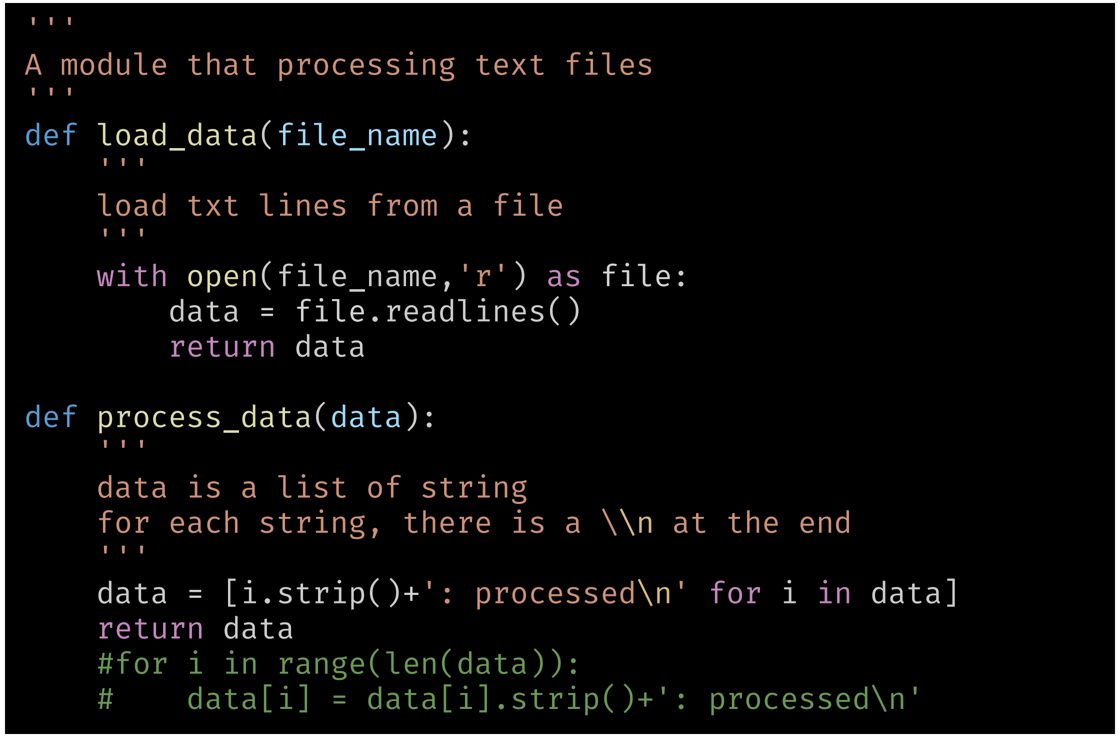
- File I/O
  - read
  - write
- Function
  - Default arguments
  - Positional and keyword arguments
  - Will a Variable's Value Change After a Function Call?
  - Variable scope
- **Modules**
- Higher-order functions

Modules

- Function is a container of bit of codes
- Module
  - Put related functions together.
  - A module is a file containing definition of functions, classes, variables, constants or any other Python object.
  - Python has the import keyword to load a module.

Create your own module

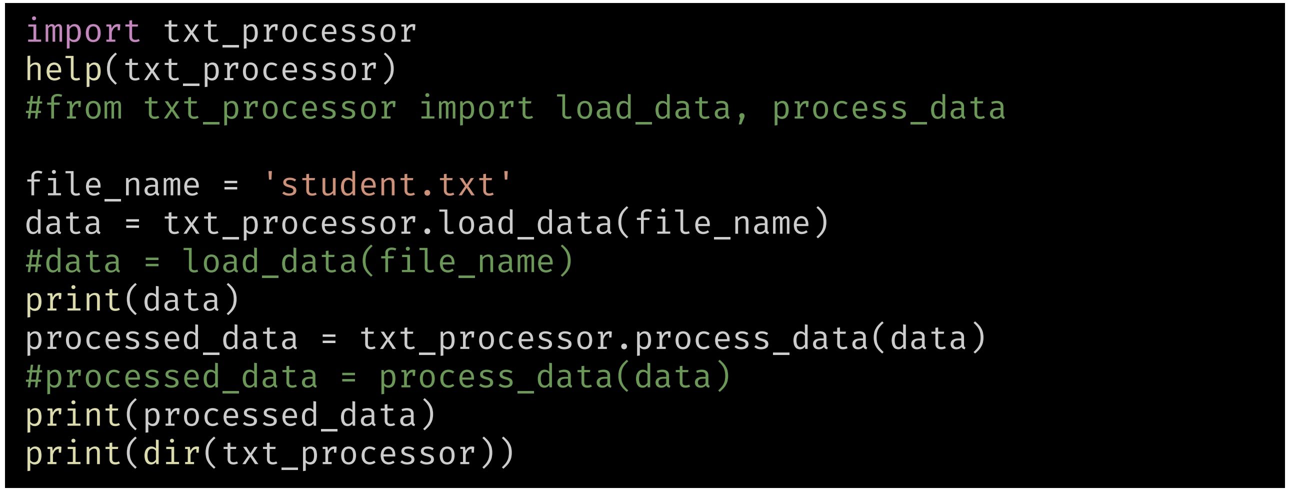
- A module is a simple python file
  - Like your regular scripts
- Create one module
  - Write one or more functions in a text file
  - Save it with a .py extension
  - Save in the same directory as other scripts
  - Give one descriptive name



```
'''
A module that processing text files
'''
def load_data(file_name):
    '''
    load txt lines from a file
    '''
    with open(file_name,'r') as file:
        data = file.readlines()
        return data

def process_data(data):
    '''
    data is a list of string
    for each string, there is a \\n at the end
    '''
    data = [i.strip()+': processed\n' for i in data]
    return data
    #for i in range(len(data)):
    #    data[i] = data[i].strip()+': processed\n'
```

Use a module

- import module_name [as simple_name]
- Without .py extension
- Use a function
    - module_name.fun_name()

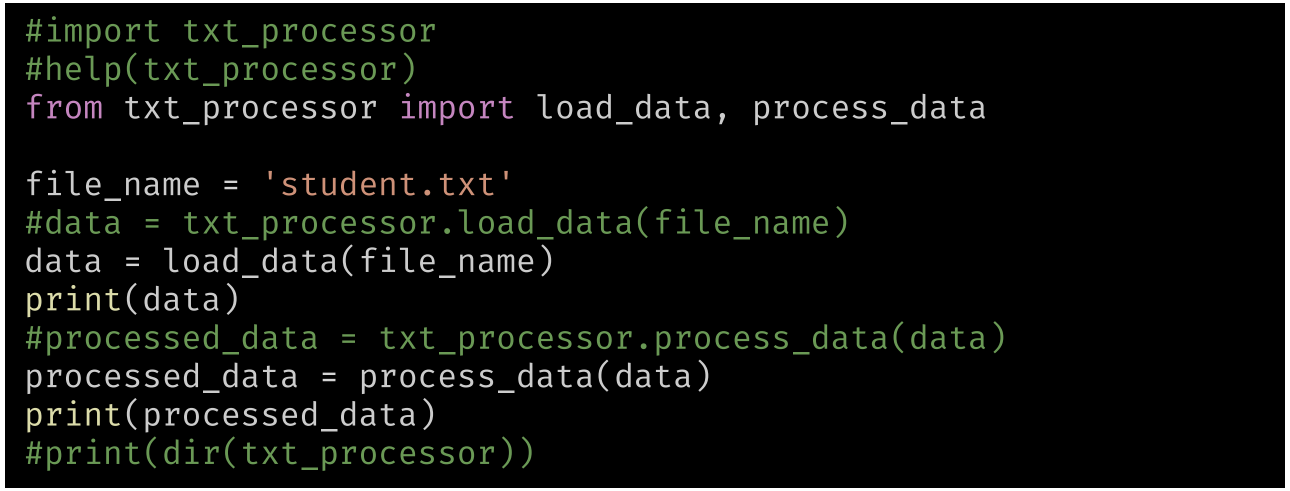


```python
import txt_processor
help(txt_processor)
#from txt_processor import load_data, process_data

file_name = 'student.txt'
data = txt_processor.load_data(file_name)
#data = load_data(file_name)
print(data)
processed_data = txt_processor.process_data(data)
#processed_data = process_data(data)
print(processed_data)
print(dir(txt_processor))
```

```
Help on module txt_processor:

NAME
    txt_processor - A module that processing text files

FUNCTIONS
    load_data(file_name)
        load txt lines from a file

    process_data(data)
        data is a list of string
        for each string, there is a \n at the end

FILE
    ---\txt_processor.py


['John\n', 'Josh\n', 'James\n', 'Jane']
['John: processed\n', 'Josh: processed\n', 'James: processed\n', 'Jane: processed\n']
['__builtins__', '__cached__', '__doc__', '__file__', '__loader__', '__name__', '__package__', '__spec__',
'load_data', 'process_data']
```

Use a module

- from module_name import (* or function_names)
  - `from txt_processor import load_data, process_data`
  - Allow you to select specific functions from the module
  - When to use from keyword
    - For a module that hundreds of functions
    - It saves loading time
- In your program, can directly use the function name



```
#import txt_processor
#help(txt_processor)
from txt_processor import load_data, process_data

file_name = 'student.txt'
#data = txt_processor.load_data(file_name)
data = load_data(file_name)
print(data)
#processed_data = txt_processor.process_data(data)
processed_data = process_data(data)
print(processed_data)
#print(dir(txt_processor))
```

Use a module

Comments to describe the program
Comments to temporarily remove part of code

ction_names)

ns from the module

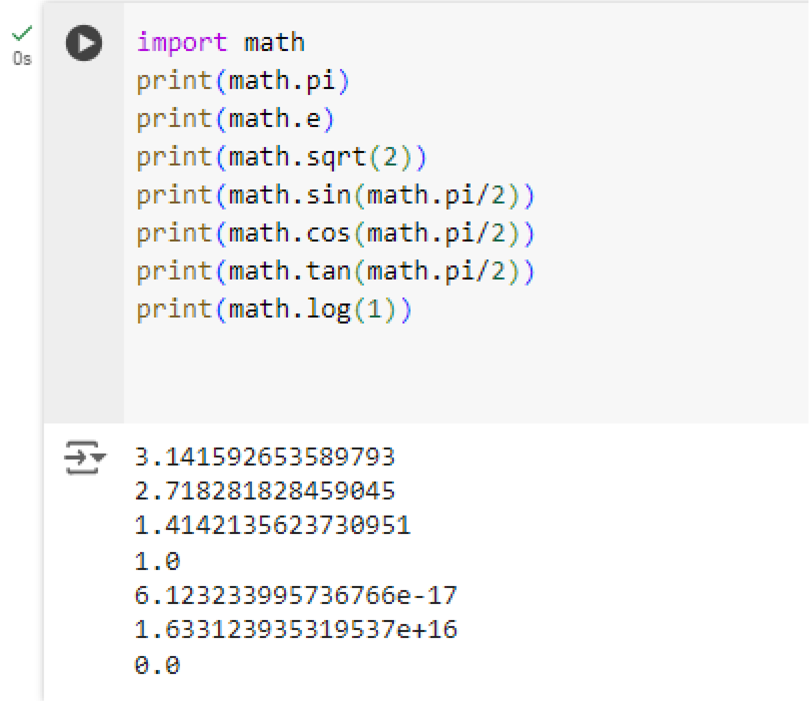
ıctions

e function name

```
r

mport load_data, process_data

.txt'
r.load_data(file_name)
e_name)

t_processor.process_data(data)
processed_data = process_data(data)
print(processed_data)
#print(dir(txt_processor))
```

Python built-in modules

- Plenty of built in modules – Python really excels
- Popular modules
  - math
  - datetime
  - random
  - os
  - urllib2



```python
import math
print(math.pi)
print(math.e)
print(math.sqrt(2))
print(math.sin(math.pi/2))
print(math.cos(math.pi/2))
print(math.tan(math.pi/2))
print(math.log(1))
```

```
3.141592653589793
2.718281828459045
1.4142135623730951
1.0
6.123233995736766e-17
1.633123935319537e+16
0.0
```

File Path

- os.path.join()
- os.path.split()
- os.path.splitext()
- os.path.exists()
- os.mkdir()

Library, Package, Module

```
Library
│
└── Package
    ├── Module (module1.py)
    ├── Module (module2.py)
    └── Subpackage
        ├── Module (module3.py)
        └── Module (module4.py)
```

Popular Python Libraries for Data Science Data Mining

- Scrapy
  - Help to build crawling programs that can retrieve structured data from web
  - A great tool for scraping data used in like ML models
- BeautifulSoup
  - Another library for web crawling and data scraping
  - If you want to collect data that's available on some website but not via a proper CSV or API, this library helps you scrape it and arrange it into the format you need
- NumPy
  - A perfect tool for scientific computing and performing basic and advanced array operations
  - Handy features performing operations on n-arrays and matrics
  - Performs math operations on array
- SciPy
  - Include modules for linear algebra, integration, optimization and statistics
  - Its main functionality was built upon NumPy
  - Works great for all kinds of scientific programming projects (science, mathematics and engineering)

Popular Python Libraries for Data Science Data Mining

- Pandas
  - Help work with “labeled” and “relational” data intuitively
  - Based on two main data structures
    - Series : one-dimensional like a list of items
    - Data frames: two-dimensional like a table with multiple columns
  - A Must-have for data wrangling, manipulation and visualization
- Matplotlib
  - Standard library for visualization like two-dimensional diagrams and graphs (histograms, scatterplots, non-cartesion coordates graphs)
  - Provides an object-oriented API for embedding plots into applications
- Seaborn
  - Based on Matplotlib as useful ML tool for visualizing statistical models –heatmaps and other types of visualizations that summarize data and depict the overall distributions
  - Get to benefit from an extensive gallery of visualization including complex ones like time series, joint plots and violin diagrams

Popular Python Libraries for Data Science Data Mining

- PyTorch
  - Developed by Facebook's AI Research lab (FAIR)
  - Deep Neural Networks: Built on a tape-based autograd system
  - Flexibility: Suitable for research and prototyping
  - Performance: High performance with easy integration into existing workflows
  - Community: Strong support and extensive documentation
- Transformers
  - Developed by: Hugging Face
  - Pre-trained Models: Access to hundreds of models in over 100 languages (e.g., BERT, GPT-2, T5, RoBERTa)
  - Easy Integration: Compatible with PyTorch and TensorFlow
  - Pipeline API: Simplifies application of models to tasks like text classification, named entity recognition, question answering, and text generation
  - Makes advanced NLP models accessible and easy to deploy

Popular Python Libraries for Data Science Data Mining

- SciKit-Learn
  - An industry-standard for data science project
  - A group of packages in the SciPy stack that were created for specific functionalities like image processing
  - Uses math operations of SciPy to explosea concise interface to ML algorithms
  - HanldesML and data mining tasks such as Clustering, regression, model selection, dimensionality reduction and classification
- TensorFlow
  - A framework for ML and DL developed at Good Brain
  - Best tool for tasks like object identification, speech recognition and many others
  - Works with artificial neural networks to handle multiple datasets

Popular Python Libraries for Data Science Data Mining

- You are free to ask ChatGPT
- You are encouraged to read more documents
- Know what you are going to do
  - Focus on your task

Review

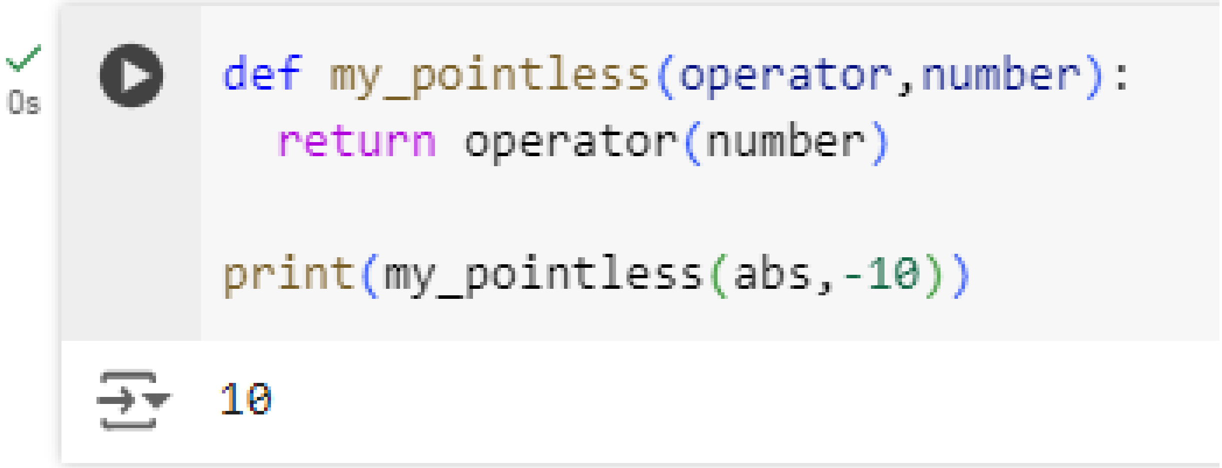
- import a_module as module
- Benefits of Modules
  - Code organization: Group related functionality together for better structure.
  - Code reuse: Write once, use in multiple programs.
  - Namespace isolation: Prevent naming conflicts across different parts of the project.
  - Maintainability: Easier to update and modify specific parts of the code.
- Further Reading
  - __init__.py

Outline

- File I/O
  - read
  - write
- Function
  - Default arguments
  - Positional and keyword arguments
  - Will a Variable's Value Change After a Function Call?
  - Variable scope
- Modules
- **Higher-order functions**

Higher Order Functions

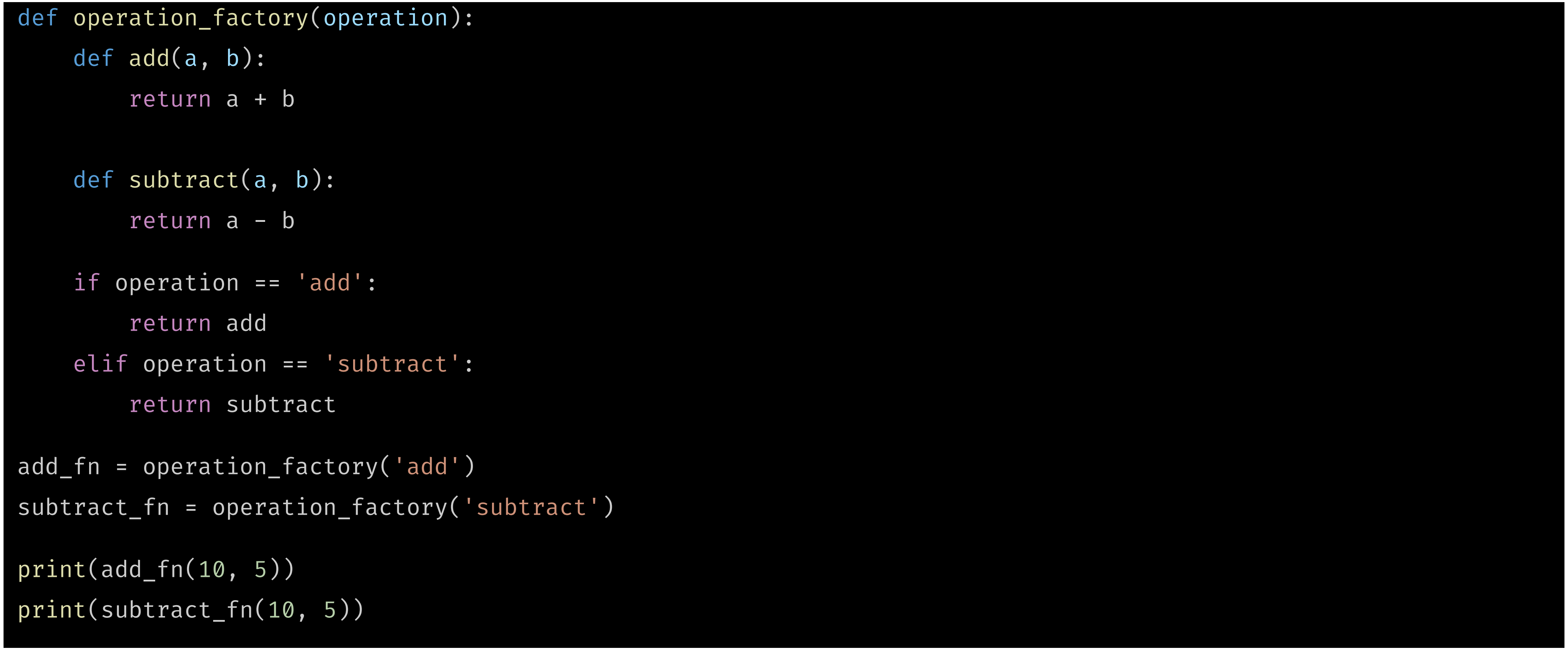
- Functions that take a function as an argument or return a function
  - A function can be assigned as the value of a variable
  - Can also be passed and returned just like any other reference variables
  - A high order functions can be stored in the form of lists, hash tables, etc.



```
def my_pointless(operator,number):
  return operator(number)

print(my_pointless(abs,-10))
```

```
10
```

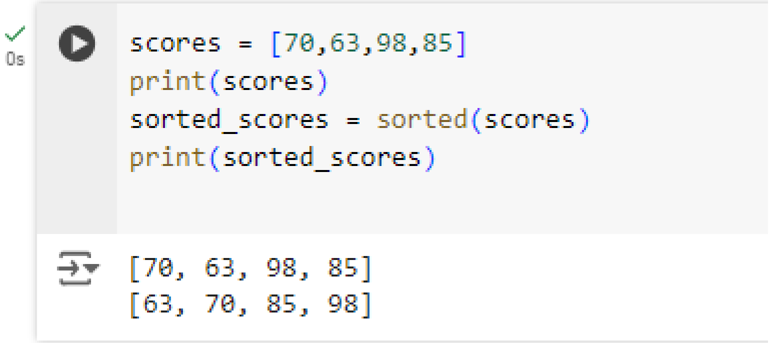
Functions can be returned



```python
def operation_factory(operation):
    def add(a, b):
        return a + b

    def subtract(a, b):
        return a - b

    if operation == 'add':
        return add
    elif operation == 'subtract':
        return subtract

add_fn = operation_factory('add')
subtract_fn = operation_factory('subtract')

print(add_fn(10, 5))
print(subtract_fn(10, 5))
```

Built-in higher order functions

- sorted()
  - Sorting items such as numbers or strings



```
scores = [70,63,98,85]
print(scores)
sorted_scores = sorted(scores)
print(sorted_scores)
```

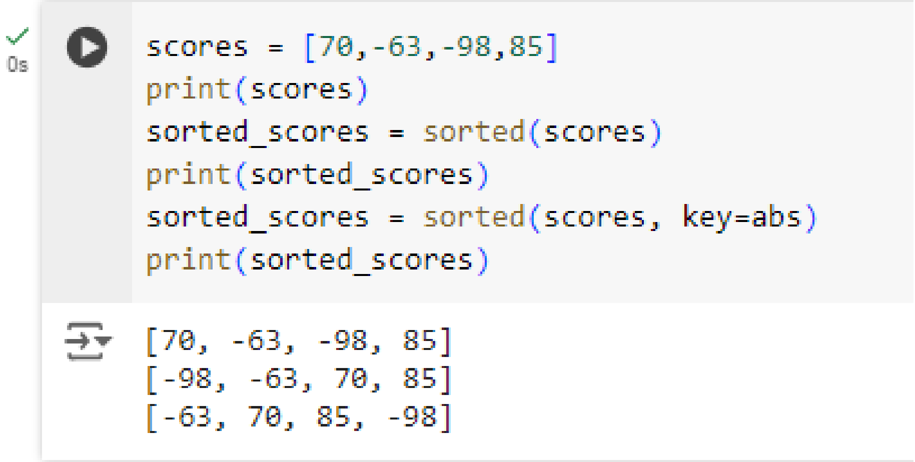
```
[70, 63, 98, 85]
[63, 70, 85, 98]
```

  - Sorting algorithms

sorted – 'higher order'

- Sort a list of numbers by absolute value instead by their nature value
  - Customize how sorting works
  - The sorted function takes a parameter named 'key'
  - The key needs to be a simple function that takes a single value and tells python the value to use in sorting it

  - Built-in function abs() can be used to get the absolute value
  - We can specify the abs() as the key argument

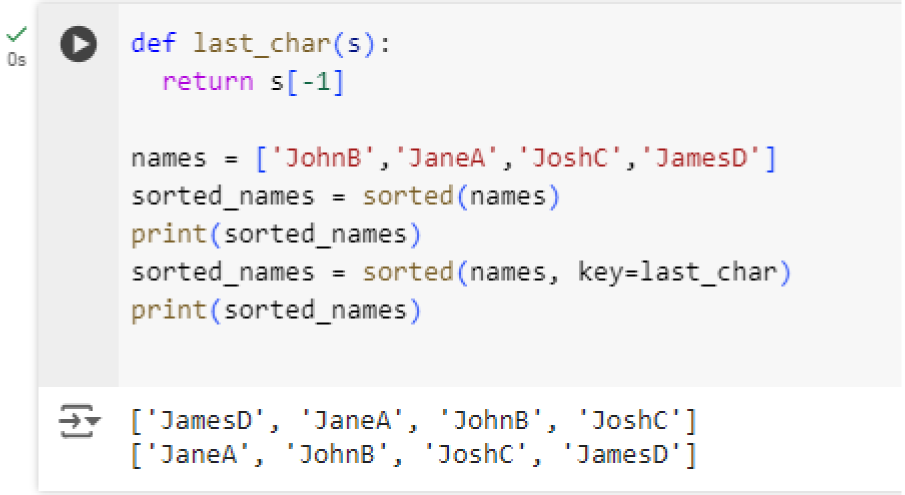


```
scores = [70,-63,-98,85]
print(scores)
sorted_scores = sorted(scores)
print(sorted_scores)
sorted_scores = sorted(scores, key=abs)
print(sorted_scores)
```

```
[70, -63, -98, 85]
[-98, -63, 70, 85]
[-63, 70, 85, -98]
```

sorted – 'higher order'

- Specify your own function to sort the list
- Example:
  - Given a list of strings, sort based on their last character
  - Write one function to get the last character of the one string
  - Specify this function as the key of the sorted function
- Bonus: built-ins max and min can be used in the same way as sorted

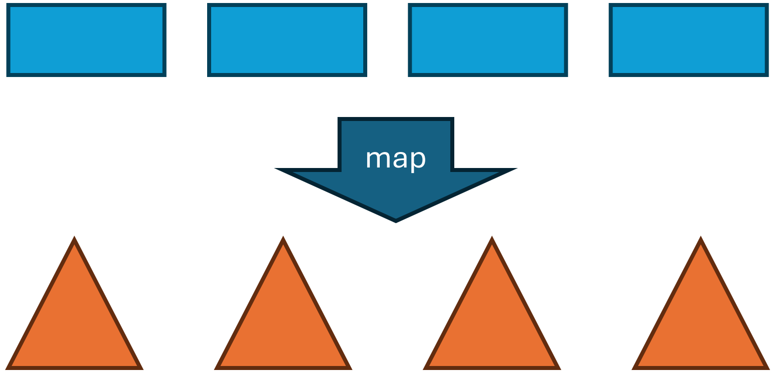


```
def last_char(s):
  return s[-1]

names = ['JohnB','JaneA','JoshC','JamesD']
sorted_names = sorted(names)
print(sorted_names)
sorted_names = sorted(names, key=last_char)
print(sorted_names)
```

```
['JamesD', 'JaneA', 'JohnB', 'JoshC']
['JaneA', 'JohnB', 'JoshC', 'JamesD']
```

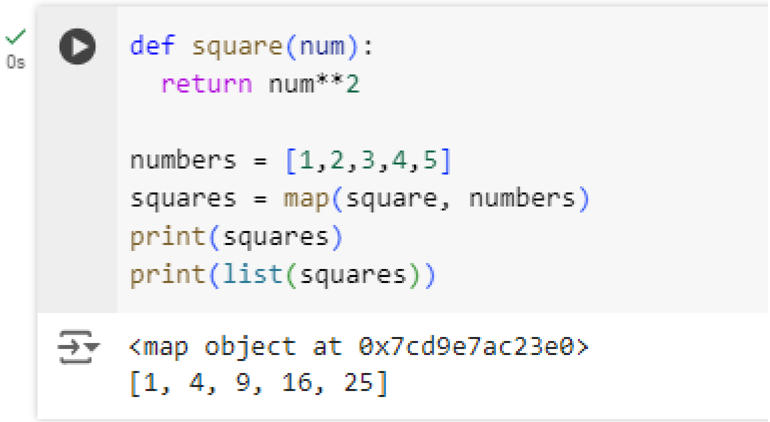
Built-in higher order functions – map

- map(func, sequence)
  - One of the arguments is a function
  - The function is applied to each item in the sequence
  - Return a sequence with different values
  - Same order, same length, mapped via a function



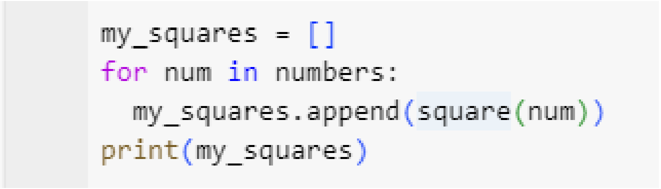
map

- Example
  - Given a list of numbers, calculate the squares of each item
  - Define one function square(x) to calculate the square number
  - Use the map function to calculate the squares of the list



```
def square(num):
  return num**2

numbers = [1,2,3,4,5]
squares = map(square, numbers)
print(squares)
print(list(squares))
```

```
<map object at 0x7cd9e7ac23e0>
[1, 4, 9, 16, 25]
```



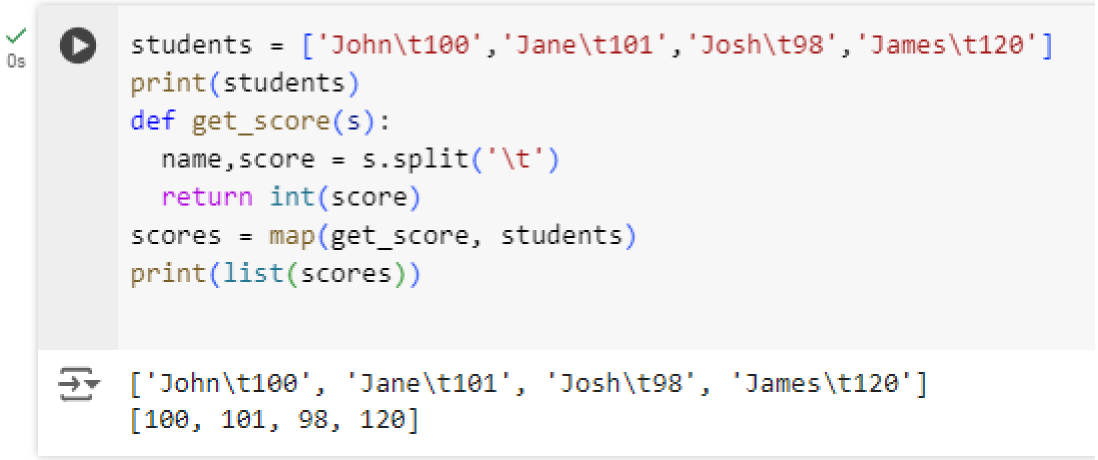
```
my_squares = []
for num in numbers:
  my_squares.append(square(num))
print(my_squares)
```

map

- map can also use built-in functions like abs
- Practice
  - Suppose you have a list of student names and scores
  - ['John\t100', 'Josh\t90','Jane\t101','James\t92']
  - Get the scores only

map

- Practice
  - Suppose you have a list of student names and scores
  - ['John\t100', 'Jane\t101', 'Josh\t98', 'James\t120']
  - Get the scores only

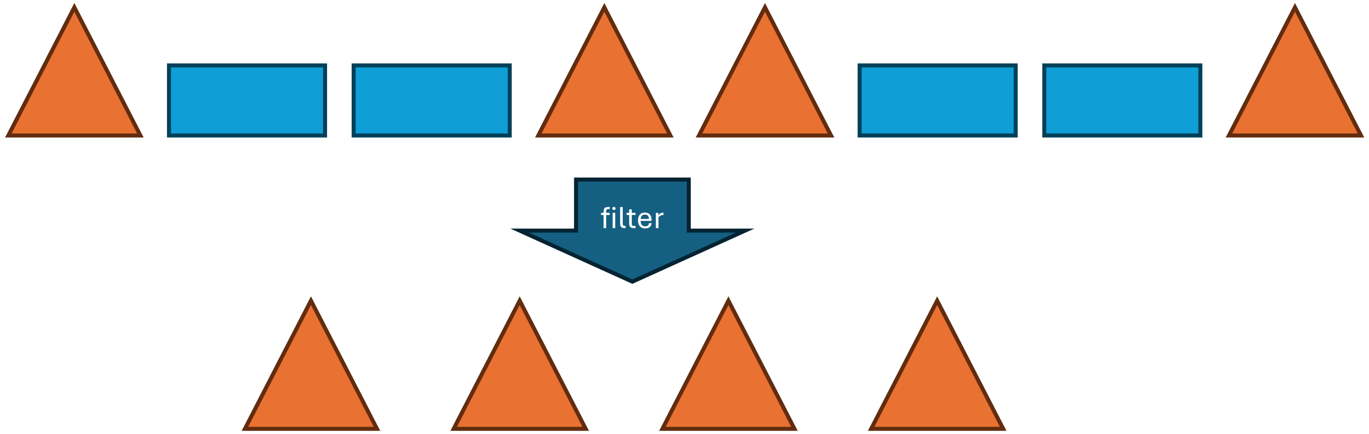


```
students = ['John\t100','Jane\t101','Josh\t98','James\t120']
print(students)
def get_score(s):
  name,score = s.split('\t')
  return int(score)
scores = map(get_score, students)
print(list(scores))
```

```
['John\t100', 'Jane\t101', 'Josh\t98', 'James\t120']
[100, 101, 98, 120]
```

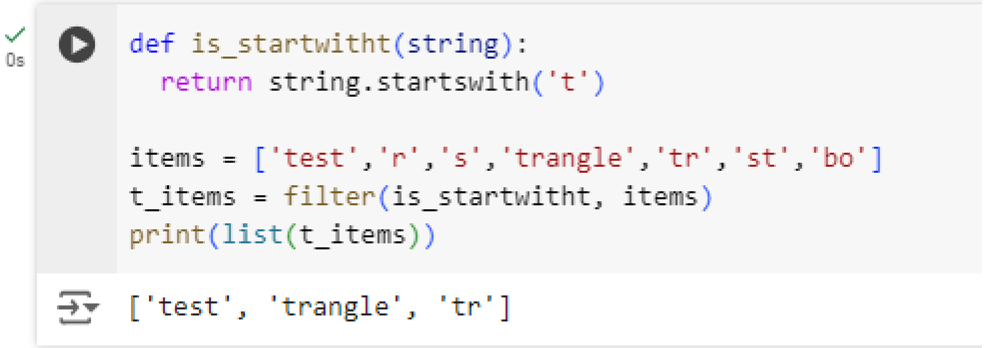
Built-in higher order functions - filter

- filter (func, sequence)
  - func: the filtering function
  - seq: a sequence of the values
  - Output: a sequence of values for which the function func returns true
- Function applies to each item to decide whether the item shall be kept or not
  - If function returns True, the item is kept
  - Otherwise, it is removed



filter

- Given a list of strings, filter those are start with 't'
  - Define a function names is_startwitht(string) as the filter function
  - Do the filter

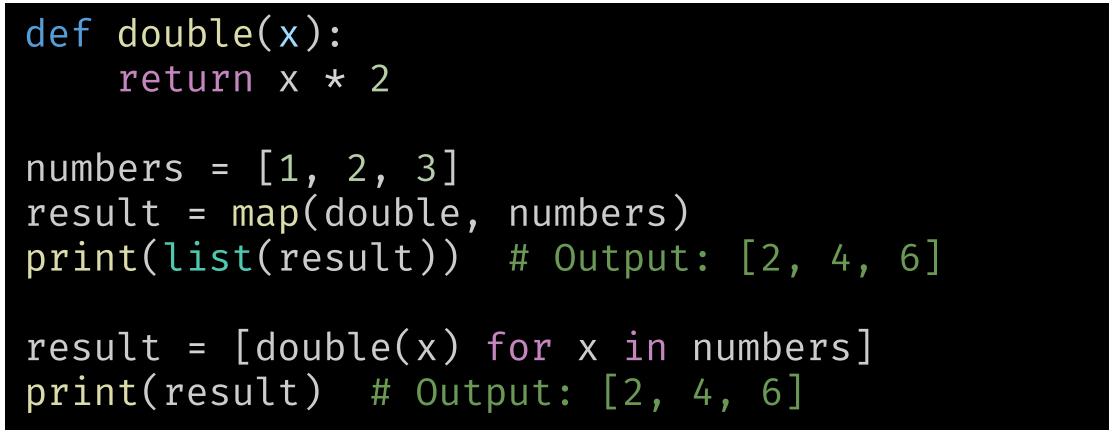


```
def is_startwitht(string):
  return string.startswith('t')

items = ['test','r','s','trangle','tr','st','bo']
t_items = filter(is_startwitht, items)
print(list(t_items))
```

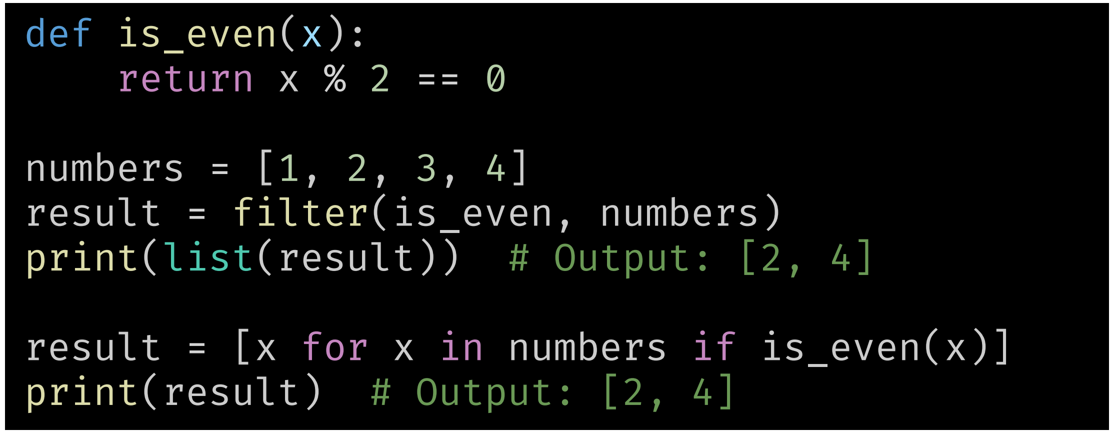
```
['test', 'trangle', 'tr']
```

- Implement the same function using for loop

Map, Filter, and List Comprehension



```
def double(x):
    return x * 2

numbers = [1, 2, 3]
result = map(double, numbers)
print(list(result))  # Output: [2, 4, 6]

result = [double(x) for x in numbers]
print(result)  # Output: [2, 4, 6]
```



```
def is_even(x):
    return x % 2 == 0

numbers = [1, 2, 3, 4]
result = filter(is_even, numbers)
print(list(result))  # Output: [2, 4]

result = [x for x in numbers if is_even(x)]
print(result)  # Output: [2, 4]
```

Map, Filter, and List Comprehension

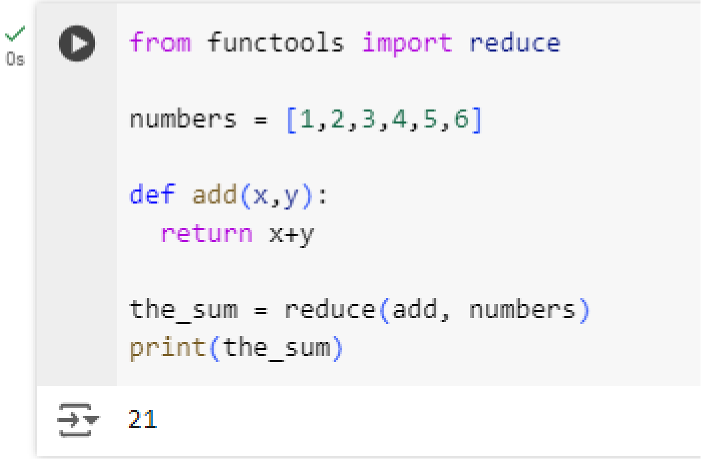
- Lazy evaluation
  - map() and filter() return iterators that delay computation until results are requested (e.g., with list(), for loop, or next()).
  - Strategy: Expressions are evaluated only when needed.
  - Benefits
    - Saves memory by not storing all elements at once.
    - Reduces unnecessary computations when only a portion of the data is needed.
    - Efficient when working with large datasets or infinite sequences.
- List comprehension immediately calculates and returns a complete list.

reduce

- reduce (func, seq)
  - Takes a sequence of data and return a single value
  - Func: Takes two values and returns one value
    1. Consume the first two values from the sequence
    2. Return a value
    3. Consume the next value and the returned value
    4. Repeat 3, until all values in the sequence have been consumed
    5. Return the final value
- In python3, it is not a built-in function any longer. It is moved to 'functools' module
- from functools import reduce

reduce

- Calculate the summation of all the numbers in one list using reduce
- Define a function that adds two numbers
- reduce (func, seq)



```
from functools import reduce

numbers = [1,2,3,4,5,6]

def add(x,y):
  return x+y

the_sum = reduce(add, numbers)
print(the_sum)
```

```
21
```

Notes

- All these functions does not change the original sequence
- A new sequence is created to hold the output

Review

- Sorted
- Map
- Filter
  - Lazy evaluation
- Reduce