Sentinel

Useful Python Modules

DEFENDING OUR DIGITAL WAY OF LIFE

Introduction

- In previous exercises and lessons you encountered many Python libraries
- For example:

  random - working with random numbers and values

  zipfile - working with compressed files (zip)

  pdb - debugging
- Today we'll meet some more

In this lesson, we will explore:

- Handling command line arguments
- Working with dates, times, and durations
- Managing file paths and directories
- Let's start!

Command Line Arguments

- Values passed to a program when it is run from the command line
- Allow users to provide input to the program
- Stored as a list of strings

```
C:\Users> python my_script.py arg1 arg2
```

Why Use Command Line Arguments?

- Advantages over the good old `input` function:
  - Automation - good for running scripts automatically (no interactive actions)
  - Input many values at once - when there are many settings and options to set, the user doesn't have to input them one by one
  - Can be easily reproduced (run again with the same parameters)

Accessing Command Line Arguments in Python

- Python provides built-in ways to access command line arguments
- Two main methods:

  sys.argv: A list containing the command line arguments

  argparse: A more advanced module for parsing command line arguments
- Today we'll focus on sys.argv

sys.argv

- A list containing the command line arguments passed to a Python script
- sys.argv[0] - the name of the script itself
- sys.argv[1], sys.argv[2], etc. - the arguments passed to the script

```
C:\Users> python my_script.py arg1 arg2
```

```
import sys

print(sys.argv)
# Output: ["my_script.py", "arg1", "arg2"]
```

sys.argv - Example

```
import sys

# Print the script name
print("Script name:", sys.argv[0])

# Print the command line arguments
print("Command line arguments:", sys.argv[1:])

# Access individual arguments
if len(sys.argv) < 3:
    print("Usage: add.py <arg1> <arg2>")
else:
    print(int(sys.argv[1]) + int(sys.argv[2]))
```

datetime

- The datetime module provides classes for working with dates, times, and time intervals
- The module includes the following main classes:

  datetime: Represents a specific date and time

  date: Represents a date (year, month, day)

  time: Represents a time (hour, minute, second, microsecond)

  timedelta: Represents a duration or time interval

datetime - example

```
from datetime import datetime

# Creating a datetime object for a specific date and time
dt = datetime(2023, 6, 10, 15, 30, 0)
print(dt)  # Output: 2023-06-10 15:30:00
```

```
print(dt.year)    # Output: 2023
print(dt.month)   # Output: 6
print(dt.day)     # Output: 10
print(dt.hour)    # Output: 15
print(dt.minute)  # Output: 30
print(dt.second)  # Output: 0
```

timedelta - example

```
from datetime import datetime, timedelta

dt1 = datetime(2023, 6, 10, 15, 30, 0)
dt2 = datetime(2023, 6, 15, 10, 0, 0)

# Calculate the difference between two datetime objects
diff = dt2 - dt1
print(diff)  # Output: 4 days, 18:30:00
```

```
# Add a time interval to a datetime object
dt3 = dt1 + timedelta(days=7, hours=2)
print(dt3)  # Output: 2023-06-17 17:30:00
```

Parsing and formatting time strings

- **strptime** - used to convert string to datetime objects (*parsing*)
- **strftime** - used to convert datetime objects to strings (*formatting*)
- Time string format is specified using format codes:

  %Y: Four-digit year (e.g., 2023)

  %m: Two-digit month (01-12)

  %d: Two-digit day of the month (01-31)

  %H: Two-digit hour (00-23)

  %M: Two-digit minute (00-59)

  %S: Two-digit second (00-59)

Example:

- Format: %Y-%m-%d %H:%M:%S
- Time string: 2024-03-09 15:30:45

Time string parsing - example

```
from datetime import datetime

# Parsing a time string
time_string = "2023-06-10 15:30:00"
time_format = "%Y-%m-%d %H:%M:%S"

dt = datetime.strptime(time_string, time_format)
print(dt)  # Output: 2023-06-10 15:30:00
```

Time string formatting - example

```
from datetime import datetime

# Formatting a datetime object
dt = datetime(2023, 6, 10, 15, 30, 0)

format1 = dt.strftime("%Y-%m-%d %H:%M:%S")
print(format1)  # Output: 2023-06-10 15:30:00

format2 = dt.strftime("%B %d, %Y")
print(format2)  # Output: June 10, 2023
```

Useful datetime functions

- **now()** - returns the current local date and time as a datetime object
- **today() -** returns the current local date as a date object
- **replace()** - creates a new datetime object with some of its components replaced

```
from datetime import datetime

original_datetime = datetime(2023, 6, 10, 12, 30, 0)

modified_datetime = original_datetime.replace(year=2024, hour=15)
print(modified_datetime)  # Output: 2024-06-10 15:30:00
```

Datetime use case example - Timer

```
from datetime import datetime

# Record the start time
start_time = datetime.now()

input('Press enter to stop the timer')

# Record the end time
end_time = datetime.now()

# Calculate the duration
duration = end_time - start_time

print(f"Task started at: {start_time.strftime('%Y-%m-%d %H:%M:%S')}")
print(f"Task ended at: {end_time.strftime('%Y-%m-%d %H:%M:%S')}")
print(f"Total duration: {duration.seconds} seconds")
```

os.path

- Module for working with files and directories paths
- Example use cases:
  - Checking if a file exists before opening it
  - Checking if a path is a directory or a file
  - Creating a new path, based on a directory name and a file name
- Let's see how to use it

Manipulating paths

- A path of a file (or directory) is made of:

  C:\folder1\folder2\folder3\...\file_or_dir_name

  location — the name of the file or the directory

- You can get the difference parts of the path using:

  os.path.basename(path) - returns the name of the file or the directory

  os.path.dirname(path) - returns the location

  os.path.join(parts_list) - takes a list of parts, and returns the joined path

  os.path.split(path) - takes a path, and returns a tuple of basename and dirname

Manipulating paths - example

```
# Joins parts of a path
full_path = os.path.join('C:\\Temp', 'file.txt')
print('Full path:', full_path) # Output: C:\Temp\file.txt

# Splits the path into the directory and the file
dirname, filename = os.path.split('C:\\Temp\\file.txt')
print('Directory path:', dirname) # Output: C:\Temp
print(File name:', filename) # Output: file.txt
```

Checking file paths

- Some useful functions for checking file paths are:
  - **os.path.exists(*path*)** - check if a file (or directory) exists
  - **os.path.isfile(*path*)** - check if a path exists and is a file
  - **os.path.isdir(*path*)** - check if a path exists and is a directory

```
# Checks if a path is a directory
isdir = os.path.isdir('C:\\Temp\\file.txt')
print('Is dir:', isdir) # Output: True if the path is a dir
```

Working with files

- Some useful functions for working with files are:

  os.listdir(*path*) - show all of the files inside of a directory

  os.rename(*current_path, new_path*) - renames a file

  os.remove(*path*) - delete a file

```
# Remove file
os.remove('C:\\Temp\\file.txt')
```

Use case example - Split to directories based on file type

- Suppose you have a directory of downloaded files (e.g., reports, images, data sheets) that you need to sort into subdirectories based on their file types.
- Let's see how we do it.

How to find useful modules (by yourself)

- Define your task
  - Decompress a file? Send an email?
- Browse builtin modules
    - Python Standard Library reference
- Search external modules
    - Python Package Index (PyPI)
- Still don't know which module to use?
  - Google
  - Other sites - reddit / stack overflow (we'll talk about them later)

Example

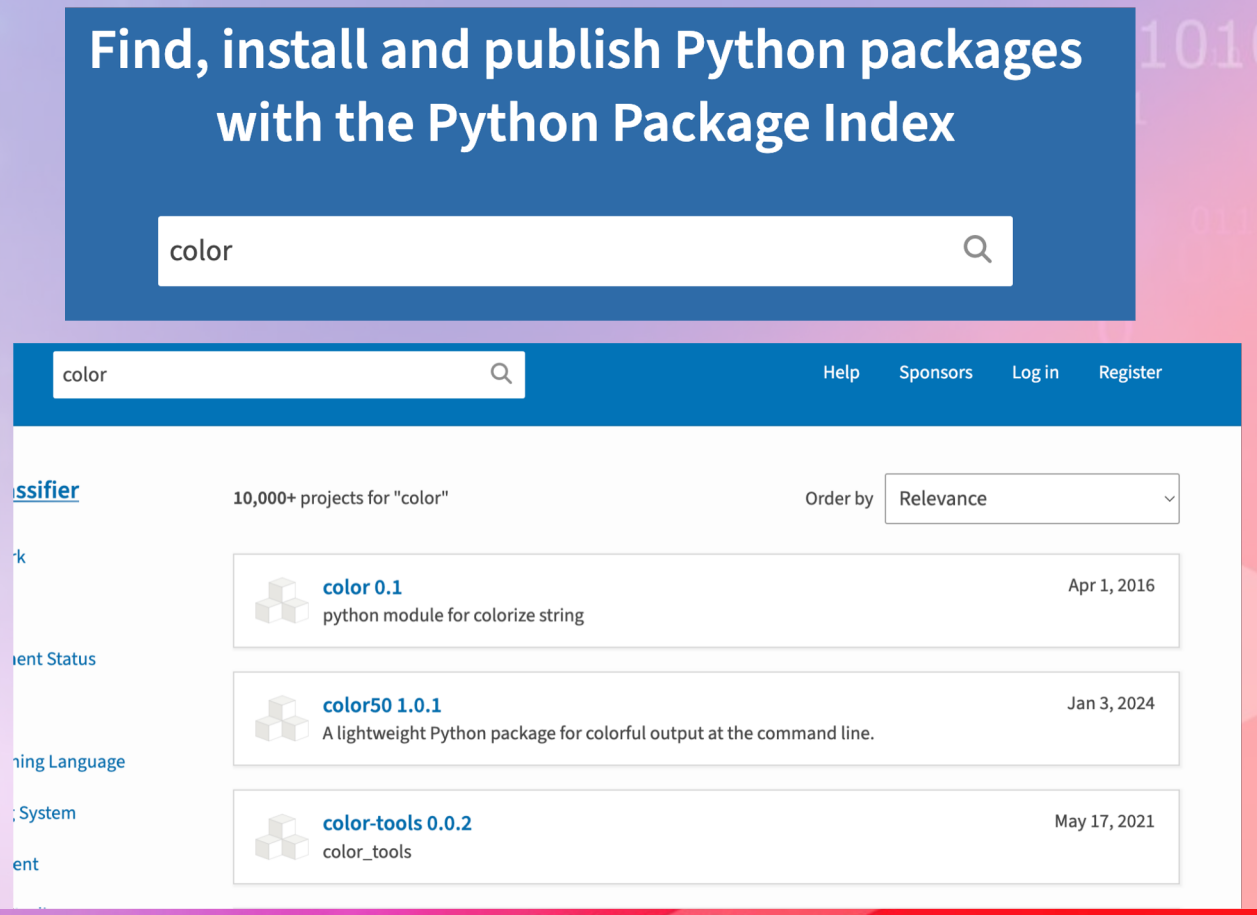
- I want to print text in color
- How do I start?

  Let's go to PyPi

- Look at the most recently updated module

- Install with:

  `pip install module_name`



How to get started with a new module

- Import the module and play with it

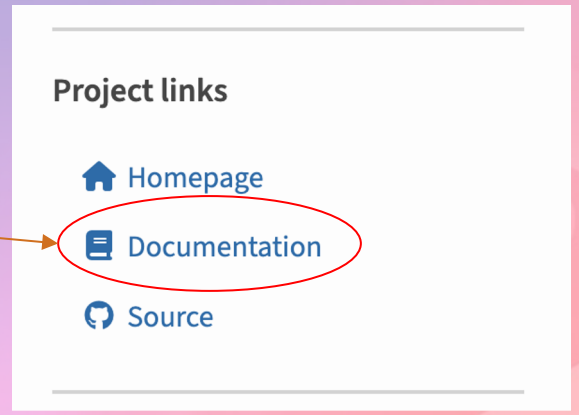
  dir(*module_name*) - list all of the functions and objects in the module

  help(*function_name*) - print the function documentation

- Find the official documentation

  - Python Standard Library reference

  - For external modules - Google / PyPi



Example

- Let's take a look at the documentation for the Color50 module
  https://color50.readthedocs.io/
- Read "Usage"
- Copy the examples and play with them

Finding more help

- Community forums

  Stack overflow - Q&A for programmers

  Reddit - forums for every topic in the world

- Search stack overflow / reddit

  Use keywords

  Look for newer answers

- Still don't find an answer? Post a new question.

Even more help

- Google
- Blog posts, tutorials, video tutorials
- Look for examples

What did we learn?

- Command line arguments
  - What are command line arguments
  - Why use them
  - sys.argv
- Working with times and dates in Python
  - datetime, date, timedelta
  - Time formats - strptime, strftime
- Working with files and directory paths
  - Manipulating paths - basename, dirname, join and split
  - Working with files - exists, isfile, isdir, listdir
- Finding new modules
- Reading documentation
- Searching for help