COMP110: Principles of Computing

5: Benchmarking

Research journal

Please go to LearningSpace and choose your paper selection!

Accessing files

Opening files

Opening files

- Files can be opened with the `open` function

Opening files

- Files can be opened with the `open` function
- Takes a **file name** and a **mode string**

FALMOUTH UNIVERSITY

Opening files

- Files can be opened with the `open` function
- Takes a **file name** and a **mode string**
  - See `https://docs.python.org/3/library/functions.html#open` for details

Opening files

- Files can be opened with the `open` function
- Takes a **file name** and a **mode string**
  - See `https://docs.python.org/3/library/functions.html#open` for details
  - E.g. `open("file.txt", "wt")` opens `file.txt` for writing as text

Opening files

- Files can be opened with the `open` function
- Takes a **file name** and a **mode string**
  - See `https://docs.python.org/3/library/functions.html#open` for details
  - E.g. `open("file.txt", "wt")` opens `file.txt` for writing as text
- Returns a **file object**, with methods including `read` and `write`

Writing to a file — example

```
f = open("hello.txt", "wt")
f.write("Hello, world!\n")
f.close()
```

Writing to a file — example

```
f = open("hello.txt", "wt")
f.write("Hello, world!\n")
f.close()
```

- Note that `write` does **not** write a line break automatically, hence the `"\n"`

Cleaning up

FALMOUTH UNIVERSITY

Cleaning up

- It is important to **close** the file once we are done writing to it

Cleaning up

- It is important to **close** the file once we are done writing to it
- We can do this using the `close` method

Cleaning up

- It is important to **close** the file once we are done writing to it
- We can do this using the `close` method
- Or we can use Python's `with` statement to do it automatically

Cleaning up

- It is important to **close** the file once we are done writing to it
- We can do this using the `close` method
- Or we can use Python's `with` statement to do it automatically

```
with open("hello.txt", "wt") as f:
    f.write("Hello, world!\n")
```

Cleaning up

- It is important to **close** the file once we are done writing to it
- We can do this using the `close` method
- Or we can use Python's `with` statement to do it automatically

```
with open("hello.txt", "wt") as f:
    f.write("Hello, world!\n")
```

- `f.close()` is automatically called when we leave the `with` statement

CSV files

CSV files

- Comma Separated Values

CSV files

- **Comma Separated Values**
- A simple text-based file format for storing tables of data

CSV files

- **Comma Separated Values**
- A simple text-based file format for storing tables of data
- Rows = lines of text, cell values separated by commas

CSV files

- **Comma Separated Values**
- A simple text-based file format for storing tables of data
- Rows = lines of text, cell values separated by commas
- Can easily be imported into spreadsheets (e.g. Excel) and data analysis tools (e.g. R)

Computation time

Resources

Resources

- All programs use **resources**

Resources

- All programs use **resources**
  - Time

Resources

- All programs use **resources**
  - Time
  - Memory

Resources

- All programs use **resources**
  - Time
  - Memory
  - Network bandwidth

Resources

- All programs use **resources**
  - Time
  - Memory
  - Network bandwidth
  - Power

FALMOUTH UNIVERSITY

Resources

- All programs use **resources**
  - Time
  - Memory
  - Network bandwidth
  - Power
  - ...

Resources

- All programs use **resources**
  - Time
  - Memory
  - Network bandwidth
  - Power
  - ...
- Often **time** is the resource we care about the most

Resources

- All programs use **resources**
  - Time
  - Memory
  - Network bandwidth
  - Power
  - ...
- Often **time** is the resource we care about the most
  - Particularly in games: want to maintain a good **frame rate** free of **lag** or **stuttering**

Resources

- All programs use **resources**
  - Time
  - Memory
  - Network bandwidth
  - Power
  - ...
- Often **time** is the resource we care about the most
  - Particularly in games: want to maintain a good **frame rate** free of **lag** or **stuttering**
  - To run at 60 frames per second, we only have **16.666 milliseconds** to do everything that needs to be done on every frame

FALMOUTH UNIVERSITY

Basic time measurement in Python

```
import time

start_time = time.perf_counter()

# ... do something here ...

end_time = time.perf_counter()
print("Time:", end_time - start_time, "seconds")
```

Basic time measurement in Python

```
import time

start_time = time.perf_counter()

# ... do something here ...

end_time = time.perf_counter()
print("Time:", end_time - start_time, "seconds")
```

- `time.perf_counter()` gives the "current time" in seconds

Basic time measurement in Python

```python
import time

start_time = time.perf_counter()

# ... do something here ...

end_time = time.perf_counter()
print("Time:", end_time - start_time, "seconds")
```

- `time.perf_counter()` gives the "current time" in seconds
- On Windows, this is the time since you first called `time.perf_counter()`

Basic time measurement in Python

```
import time

start_time = time.perf_counter()

# ... do something here ...

end_time = time.perf_counter()
print("Time:", end_time - start_time, "seconds")
```

- `time.perf_counter()` gives the "current time" in seconds
- On Windows, this is the time since you first called `time.perf_counter()`
- Means little by itself, but **comparing** two values tells us how much time has **elapsed**

Repeating for better accuracy

```python
import time

start_time = time.perf_counter()

repetition_count = 1000

for repetition in range(repetition_count):
    # ... do something here ...

end_time = time.perf_counter()
total_time = end_time - start_time
print("Time:", total_time, "seconds")
```

FALMOUTH
UNIVERSITY

Repeating for better accuracy

```
import time

start_time = time.perf_counter()

repetition_count = 1000

for repetition in range(repetition_count):
    # ... do something here ...

end_time = time.perf_counter()
total_time = end_time - start_time
print("Time:", total_time, "seconds")
```

- There is some **overhead** from the `for` loop, but in practice it is negligible

```python
# Time creation of a list of random elements
# Write the results to a CSV file

import time
import random

rep_count = 10

with open("results.csv", "wt") as f:
    for n in range(10, 10000, 10):
        print(n)

        start_time = time.perf_counter()
        for repetition in range(rep_count):
            # Create a list of size n
            my_list = []
            for i in range(n):
                my_list.append(random.randrange(1000))

        end_time = time.perf_counter()
        total_time = end_time - start_time
        f.write("{0},{1}\n".format(n, total_time))
```

```
# Time appending to a list
# Write the results to a CSV file

import time
import random

rep_count = 1000

with open("results.csv", "wt") as f:
    for n in range(100, 100000, 100):
        print(n)

        # Create a list of n elements (not timed)
        my_list = []
        for i in range(n):
            my_list.append(random.randrange(1000))

        # Time appending elements to the list
        start_time = time.perf_counter()
        for repetition in range(rep_count):
            my_list.append(random.randrange(1000))

        end_time = time.perf_counter()
        total_time = end_time - start_time
        f.write("{0},{1}\n".format(n, total_time))
```

Workshop exercise

Workshop exercise

- Investigate various operations on Python lists, and how their running time varies with the size of the list

Workshop exercise

- Investigate various operations on Python lists, and how their running time varies with the size of the list
- For each of the operations listed on the next slide:

Workshop exercise

- Investigate various operations on Python lists, and how their running time varies with the size of the list
- For each of the operations listed on the next slide:
  - **Find out** how to do the operation

Workshop exercise

- Investigate various operations on Python lists, and how their running time varies with the size of the list
- For each of the operations listed on the next slide:
  - **Find out** how to do the operation
  - **Write** code similar to the previous slides, to generate a list of size *n* (for various values of *n*) and then time the operation on that list

Workshop exercise

- Investigate various operations on Python lists, and how their running time varies with the size of the list
- For each of the operations listed on the next slide:
  - **Find out** how to do the operation
  - **Write** code similar to the previous slides, to generate a list of size *n* (for various values of *n*) and then time the operation on that list
  - **Plot** graphs of the operations using Excel

Workshop exercise

- Investigate various operations on Python lists, and how their running time varies with the size of the list
- For each of the operations listed on the next slide:
  - **Find out** how to do the operation
  - **Write** code similar to the previous slides, to generate a list of size *n* (for various values of *n*) and then time the operation on that list
  - **Plot** graphs of the operations using Excel
  - (Advanced mode: instead of using Excel, plot graphs directly from Python using the Matplotlib library)

Operations to time

- Append an element
- Insert an element at the beginning
- Insert an element at a random position
- Delete the first element
- Delete the last element
- Delete a random element
- Get the first element
- Get the last element
- Get a random element
- Find if the list contains a specific element
- Get the smallest element
- Get the largest element
- Get the sum of all elements
- Get the length of the list
- Copy the list
- Reverse the list
- Sort the list
- Randomly shuffle the list
- Convert the list to string