Lecture 12: Algorithms and efficiency

Morten Rieger Hannemose, Vedrana Andersen Dahl

Fall 2023

Course overview

- Data types `int` and `float` and computation
- Functions
- Flow control with conditions and loops
- Data types `str`, `list`, methods, indexing and traversing
- Data types `dict`, `tuple`
- Reading and writing files
- Object-oriented programming
- Numpy, matplotlib

Last two weeks of the course (this week and next week)

- Algorithms and efficiency

  (writing efficient and readable code)
- Summary and discussion of the exam

  (wrapping up, revisiting midterm exam, and extras)

Code quality

- Software quality: reliability, efficiency, security, maintainability.
- In this course, the focus is on:
  - Correctness (the only thing we test)
  - Efficiency
  - Readability
  - Style

Code size

- In this course: 10-20 lines of code.
- My largest project: a few thousand lines of code.
- Video game: A few million lines of code

Code efficiency and style

Examples on

- Counting things
- Searching for things
- (a bit on) Sorting and merging things

Focus on

- Avoiding unnecessary computation
- Carefully choosing variables
- (a bit on) Commenting
- Common pitfalls

Counting and searching

What is the intended result?

- Is there a …
- Where is the …
- How many …
- What are …

What is the occurrence I'm searching for?

- …number 3? …letter 'F'?
- …number larger than 3? …capital letter?
- …number larger than both its predecessor and successor?
- …an item best according to some measure?
- …number with an odd index which is larger than 3?

Remember from the mid-term test exam

- First alarm: When did the alarm occur? (Index of the first occurrence of a number either …)
- Typical successor: What is typically following a letter? (What is …)
- Dice fairness. What appears most frequently and how many times?

Code used for coding examples

Simplifying code

```python
text = 'Something'
too_long = len(text)>10
if too_long:  # instead too_long==True
    print('The text is too long')

def should_pay_half_price(age):
    # instead if-sentence
    return (age < 18) or (age > 65)

age = 75
full_price = 100
# either full price or half price
# instead of if-sentence
price = 0.5 * full_price + 0.5 * full_price * (18 <= age <= 65)


```

Avoid unnecessary computation

```python
text = 'This is a very long text which is slow to compute the length of.'
len_text = len(text)
for p in [10, 50, 90]:
    print(f'{p}% is {p / 100 * len_text}')


```

Code used for coding examples

Searching and counting, lists

```
items = [5, 6, 8, 2, 4, 5, 7, 8, 4, 6, 4, 3, 5, 6, 7, 3, 2,
      4, 5, 6, 7, 8, 9]

# Use built-in list methods
print(3 in items)
print(items.index(3))
print(items.count(3))

# Is there an occurrence?
found_it = False
for item in items:
    if item > 6:
        found_it = True
        break

# Where is the first occurrence?
index = -1 # a dedicated value
for i in range(len(items)):
    item = items[i]
    if item > 6:
        index = i
        break

# How many occurrences?
counter = 0
for item in items:
    if item > 6:
        counter += 1

```

Searching and counting, lists

```
# How many occurrences?
counter = 0
for item in items:
    if item > 6:
        counter += 1
print(counter)

# Looking for the index of somehow best item, with smallest
      abs(item - 5)
# max and min are special cases of this
best_distance = abs(items[0] - 5)
best_distance = 1000
for item in items:
    this_distance = abs(item - 5)
    if this_distance < best_distance:
        best_distance = this_distance

# Larger than both neighbors
for i in range(1, len(items) - 1):
    if items[i] > items[i - 1] and items[i] > items[i + 1]:
        print(items[i])

# Odd index and larger than 6
for i in range(len(items)):
    if i % 2 == 1 and items[i] > 6:
        print(items[i])

```

Code used for coding examples

Searching and counting, numpy and lists

```
import numpy as np

numpy_items = np.array(items)
print(3 in numpy_items)
# print(numpy_items.index(3))  # This will not work
print(numpy_items == 3)
print((numpy_items == 3).any())

# print(numpy_items.count(3))  # This will not work
print((numpy_items == 3).sum())


print(np.where(numpy_items == 3))
print(numpy_items[::2])

peak = (numpy_items[1:-1] > numpy_items[2:]) & (numpy_items[1:-1] > numpy_items[:-2])
print(peak)

print(numpy_items.max())
print(numpy_items.argmax())
```

Sorting and merging

```
items = [5, 6, 8, 2, 4, 5, 7, 8, 4, 6, 4, 3, 5, 6, 7, 3, 2, 4, 5, 6, 7, 8, 9]
print(sorted(items))
print(np.sort(numpy_items))
print(np.unique(numpy_items))

items = [4, 6, 3, 8, 5]
other_items = [5, 8, 11, 13, 9]
for i in other_items:
    if i not in items:
        items.append(i)
print(items)
```