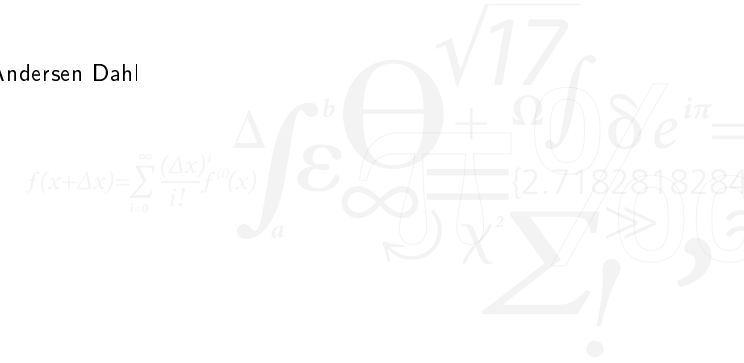
Lecture 5: Lists

Morten Rieger Hannemose, Vedrana Andersen Dahl

Fall 2023



Today's lecture

1. Lists (ca. 30 min)
2. Coding example (ca. 30 min)

Lists: creation

```
my_list = [1, 2, 3, 4, 5]
another_list = [1, 'Sasha', 3, 'Emily', 5]
yal = ['one', 'two', [1, 2, 3], 'four', 5]
another = []
```

```
my_list = list(range(10))
another_list = list('my string')
back_to_string = str(my_list)
```

- A list is a sequence of values.
- Values in a list are called elements or items.
- Often, all elements of a list have the same type, but elements can be of any type – also lists.
- A list can be created by enclosing the elements in square brackets.
- Remember: word `list` is a built-in type, don't use it as a variable name.

Lists: indexing

```
# indexing
print(my_list[2])

# indexing from the end
print(my_list[-2])

# slicing
print(my_list[1:3])
print(my_list[1:-1])

# slicing with step
print(my_list[0:10:2])
print(my_list[::2])

# slicing with negative step (reversing)
print(my_list[::-1])
print(my_list[::-2])

# difference between indexing and slicing
print(my_list[0])
print(my_list[0:1])
```

Similar to strings

- Indexing
- Slicing
- Traversing

```
# traversing a list
for item in my_list:
    print(item)

# traversing a list with index
for index in range(len(my_list)):
    print('List element', index, 'is',
     my_list[index])
```

Lists, mutable

```python
# lists are mutable
first_list = [1, 2, 3, 4, 5]
first_list[2] = 8
print(first_list)
```

```python
# change affects all references
second = ['Elements', 'of', 'list']
third = second

second[0] = 'Changed' # item assignment

print(third)
print(second)
```

Unlike strings

- Lists are mutable
- Modifying list affects all aliased objects

Lists, methods, operators and functions

```
# list method append
one_list = [1, 2, 3, 4, 5]
one_list.append(6)

print(one_list)

# list concatenation
second_list = [7, 8, 9, 10]
second_list += [11]

print(second_list)
```

List methods

- Dot notation
- `append()`, `copy()`, `sort()`, `remove()`...

List operators

- Concatenate, repeat
- Also as augmented assignment +=

List functions

- List functions: `min()`, `max()`, `sum()`, `len()`,

Lists and strings

```
1 text = 'This is some text containing some
      words.'
2 words = text.split()
3 new_text = '-'.join(words)
```

String methods with lists

- `split()` returns a list. Check optional arguments.
- `join()` takes a list as input.

Example 1

Write a function that takes a list as input. The function should return a list that contains all elements of the input list, and one additional last element which is a string `'last'`.

Solve the problem in two different ways:

- By creating and returning a new list.
- By modifying and returning the input list.

Investigate what happens with the original list in the main scope.

Coding example

`stable_measurement.py`, (slightly modified) exam from June 2020.

Stable measurements

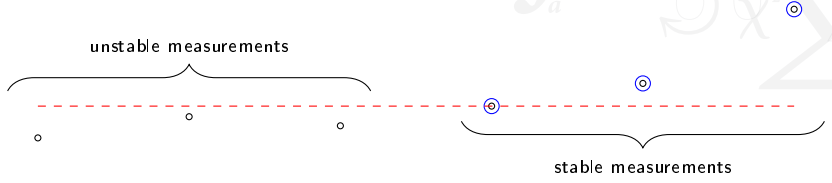
The measuring equipment gives unstable values in the initial part of the experiment. The stable interval starts with the first value which is strictly larger than all previous values, and strictly smaller than all later values.

Problem formulation

Create a function `stable_measurements` which takes a list of measurements as input, and returns a list with only stable measurements. If the stable interval is not found, an empty list is to be returned.

Consider the sequence

$$m = [4.3,\quad 5.7,\quad 5.1,\quad 6.4,\quad 7.9,\quad 12.8]$$



Code used for coding examples

Example 1

```
def add_last_reassign(my_list):
    my_list = my_list + ['last']
    return my_list

def add_last_modify(my_list):
    my_list.append('last')
    return my_list

test_list = ['first', 'second', 'third']

out1 = add_last_reassign(test_list)
print(out1)
print(test_list)

out2 = add_last_modify(test_list)
print(out2)
print(test_list)
```

Stable measurements

```
def valid_start(m, index):

    return (max(m[:index]) < m[index]) and
     (m[index] < min(m[index+1:]))


def stable_measurements(m):

    for i in range(1, len(m) - 1):

        if valid_start(m, i):
            return m[i:]

    return []
```