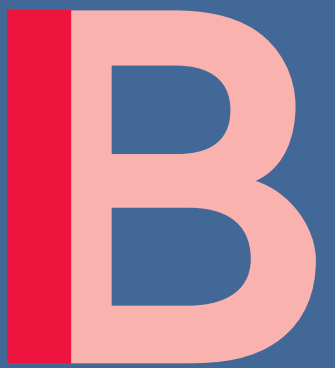
Data Structures and Algorithms

Live Class 3

Heikki Peura

h.peura@imperial.ac.uk



Announcements

There was an error on the Hub in Session 2

- Exercise q2 missing from instructions
- `python ok -q q2 -u`
- You won't lose points for it
- Thank you those who alerted us!

Today

1. **Recap**
2. For loops
3. Sequential data
   - Strings
   - Lists
4. Problem solving and debugging

So far

- Variables allow us to store results
- Conditional statements allow us to make decisions
- While loops let us repeat things
- Algorithms are step-by-step instructions
- Functions let us give names to procedures

Go to menti.com

What is the output sequence?

```
1  i = 4
2  while i > 2:
3      print(i*i)
4  i = i - 1
```

A. 16, 9, 4

B. 16, 9

C. 9, 4

D. An infinite loop

E. I don’t know

What is the output?

```
1 def fun_function(a, b):
2     if b > a:
3         return b
4     print(a)
5 x = fun_function(3, 4)
6 print(x)
```

A. 4

B. 3

C. 3, then 4

D. 4, then 3

E. I don’t know

What is the output?

```
1 def fun_function(a, b):
2     if b > a:
3         print(2*b)
4     print(a)
5 x = fun_function(-3, -1)
6 print(abs(x))
```

A. -2

B. -2, -3, 3

C. -2, -3, Error

D. -2, Error

E. I don't know

Today

1. Recap
2. **For loops**
3. Sequential data
   - Strings
   - Lists
4. Problem solving and debugging

For loops

We know how to use while loops:

```
i = 0
while i < 4:
    print(i)
    i = i + 1
```

We can also use a for loop:

```
# range(n) generates integers 0, ..., n-1
for i in range(4):
    print(i)
```

General form:

```
for item in sequence:
    # do something with item
```

Range:

```
# range(start:end:step)
# step can be negative, for example range(10, 7, -1)
for i in range(5, 11, 2):
    print(i)
```

What is the output sequence?

```
1 for i in range(1, 4):
2     print(i)
3     print(i*i)
```

A. 2, 4, 3, 9, 4, 16

B. 1, 1, 2, 4, 3, 9, 4, 16

C. 1, 1, 2, 4, 3, 9

D. 2, 4, 3, 9

E. I don’t know

How many times is 'Hello' printed?

```
1 for i in range(1, 4):
2     print(i)
3 print('Hello')
```

A. 0

B. 1

C. 3

D. 4

E. I don't know

What is the output?

```
1  s = 0
2  for i in range(3):
3      for j in range(2):
4          s += 1
5  print(s)
```

A. 0

B. 2

C. 3

D. 6

E. I don’t know

Today

1. Recap
2. For loops
3. **Sequential data**
   - Lists
   - Strings
4. Problem solving and debugging

Lists

So far we have worked with simple data like numbers.

Often we have multiple sequential data, for example stock prices.

```
1  prices = [5.2, 4.9, 5.1, 5.6, 7.0]
```

List syntax: square brackets, comma-separated values (which can be anything).

List example

```
>>> names = [] # create empty list
>>> names.append('Ruth') # add an item to the end of list
>>> names.append('Mo')
>>> print(names)
['Ruth', 'Mo']
>>> print(names[0]) # indexing: 0, 1, 2, ..., we can slice as with lists
Ruth
>>> names[1] = 'Olivia' # change Mo -> Olivia
>>> for name in names:
>>>     print('Hello ' + name)
Hello Ruth
Hello Olivia
```

Go to menti.com

Are these valid lists?

```
a = [5, 'hippo', 3]
b = [1]
```

A. Both are

B. Neither is

C. `a` is but `b` is not

D. `b` is but `a` is not

E. I don't know

What is the output?

```
1 c = ['Berlin', 'Jakarta', 'Antananarivo']
2 print(c[0])
3 c.append('Kinshasa')
4 print(c[1:3])
```

A. `Berlin`, then `['Jakarta', 'Antananarivo', 'Kinshasa']`

B. `Berlin`, then `['Jakarta', 'Antananarivo']`

C. `Berlin`, then `['Jakarta']`

D. An error

E. I don't know

What is the output?

```
1 c = [['Berlin', 'Germany'], ['Jakarta', 'Indonesia']]
2 print(c[1])
```

A. `Germany`

B. `['Berlin', 'Germany']`

C. `['Jakarta', 'Indonesia']`

D. An error

E. I don't know

What is the output?

```
1  c = [['Berlin', 'Germany'], ['Jakarta', 'Indonesia'], ['Antananarivo', 'Madagascar']]
2  for element in c:
3      print(element[0])
```

A. `Berlin, Jakarta, Antanarivo`

B. `B, G, J, I, A, M`

C. `Germany, Indonesia, Madagascar`

D. An error

E. I don't know

What is the output?

```
numbers = [1, 2, 3]
more_numbers = numbers
more_numbers[0] = 5
all_numbers = numbers + more_numbers
print(all_numbers)
```

A. [1, 2, 3, 5, 2, 3]

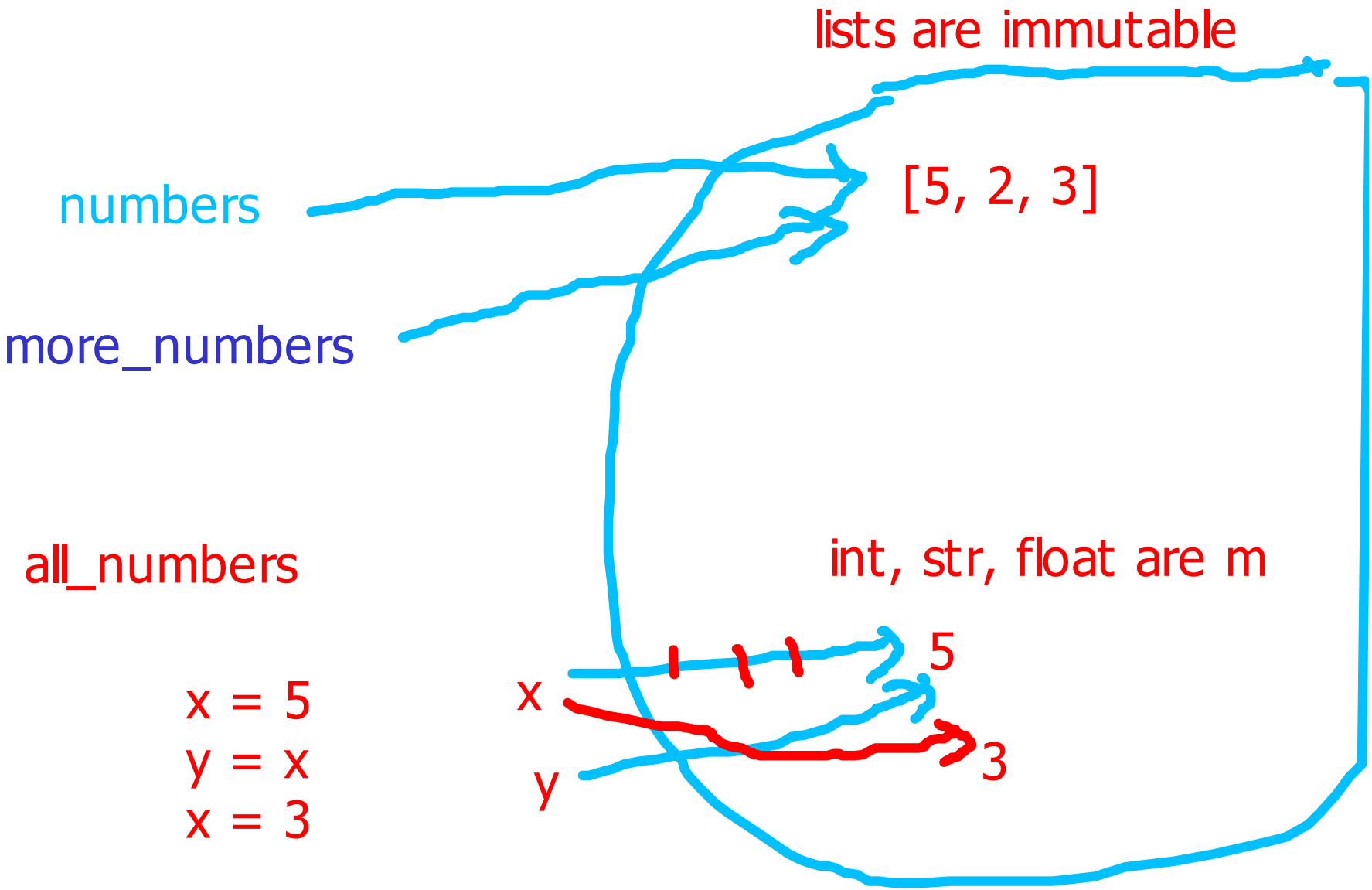
B. [2, 4, 6]

C. [5, 2, 3, 5, 2, 3]

D. [1, 2, 3, 1, 2, 3]

E. I don’t know

Lists in memory



Working with text: strings

We have already seen some strings in the first session

```
s = 'hippopotamus'
```

We can get parts of a string by **slicing**

```
>>> print(s[0]) # indexing is 0, 1, 2, ...
h
>>> print(s[1:4])
ipp
>>> print(s[4:9:2])
ooa
>>> print(s[:len(s)-1])
hippopotamu
```

`s[start:end:step]` gets characters from `start` index to `end - 1` index with specified `step` size (which could be negative).

Looping and methods

```
>>> s1 = 'hiphop'
>>> s2 = 'opotamus'
>>> s = s1 + s2
>>> print(s)
hiphopopotamus
>>> for char in s:
>>>     print(char + '!')
```

Methods are functions associated with a type of object, here strings. They are used with the “dot notation”.

```
>>> s = 'I like blackberries.'
>>> new_s = s.replace('black', 'blue')
>>> print(new_s)
I like blueberries.
```

There are many string methods – some useful are ones in the online session materials

Today

1. Recap
2. For loops
3. Sequential data
   - Strings
   - Lists
4. **Problem solving and debugging**

Problem solving

1. Make sure you understand the problem. Try examples.
2. Design a solution. Use paper. Find patterns. Check module materials. Try things out. Solve a part of the problem or a special case.
3. Implement the solution. Start from small parts. If your code does not work, trace what the program does line by line to find where the problem is. Use the print function.
4. Test the solution with examples.

Problem: Given a list containing numbers, write a function that returns a list of cumulative sums up to each element.

Debugging

(Demo)

Review

With lists and for loops, we are ready to start working on a broad range of problems.

Up next:

- Session 3 Review - optional exercises on Trump tweets
- Session 4 prep - computational complexity