Algorithms and Data Structures (ECS529)

Nikos Tzevelekos

Lecture 0

Introduction to Python

ADS Keywords: algorithm

An algorithm is a description of a formal process, i.e. sequence of steps or rules, to solve a specific problem:

- we use algorithms in every-day life (e.g. routing, timetabling, cooking)
- number calculations use algorithms (e.g. *column addition*)
- web protocols use algorithms (e.g. authentication, routing)
- databases use algorithms (e.g. searching, inserting)
- image processing, artificial intelligence, cryptography, cryptocurrencies, cloud computing and more generally **every** App/program/website that does anything useful uses some (optimised!) algorithm for it

Example algorithms: Searching

Every algorithms course starts with searching. *– Anonymous*

SEARCH is the following problem:

- given an array of integers A and an integer k
- if k is in A then return its position, otherwise return -1

Notes:

- return its position means: return some i such that $A[i] = k$
- it is OK if k occurs many times in A: just return one of its positions
- the array A can be arbitrarily large
- in general, SEARCH is about arrays of elements of any type (not necessarily integers)

First solution: linear search

Algorithm:

> We go through all the elements of the array A and compare each of them with k. At the end, if we found an i such that $A[i] = k$ then we return that i, otherwise we return -1.

This is a correct algorithm description, but written in plain language:

- it correctly describes a solution to SEARCH
- it has some imprecision: e.g. what does "*go through all the elements*" mean? Or "*if we found an i such that*"?
- it is not very helpful for writing a program solving SEARCH

A more technical-algorithmic language is needed in order to describe algorithms precisely.

We will use a programming language for that purpose: 

Linear search code

Here is our solution to SEARCH, in Python (top) and in Java (bottom)

```
def search(A, k):
    found = -1
    for i in range(len(A)):
        if A[i] == k:
            found = i
    return found
```

```
public static int search(int[] A, int k)
{
    int found = -1;
    for (int i=0; i<A.length; i++) {
        if (A[i] == k) found = i;
    }
    return found;
}
```

Do I really have to learn Python now?

Yes, we will use Python as our algorithmic language:

- all algorithms in the lectures will be written in Python
- most lab exercises will require you to write Python code

We will not ban Java though:

- most algorithms will also be given in Java
- in the exams, you can write algorithms in either Python or Java
  - you are not going to lose marks for choosing one over the other
  - but note this does not apply to pythonic lab exercises
- you can solve the mini-project using either Python or Java

Python from a Java perspective

Python is like Java *without the extra stuff*:

- we write definitions of functions/procedures like this:

```
def myFunction(x, s):
```

  without having to include

```
public static void myFunction(int x, String s)
```

  or `class myClass { … }`

- we don't need to write types in our code or declare variables,

  but we *do* need to be careful so that types match (e.g. we should not write: `5 + "six"`)

- we don't need `{ … }` or `;`. We can use careful **indentation** instead!
- we can write code scripts and run them without compiling them

Python analysis

this says we define a function
(think static methods in Java)

function header:
- gives the function's name (`search`)
- gives the arguments (`A` and `n`)
- ends in colon (:)

define a variable `found`
and initialise it to `-1`

`len(A)` returns the length of array `A`

```
def search(A, k):
    found = -1
    for i in range(len(A)):
        if A[i] == k:
            found = i
    return found
```

body of `for` loop

this is how we write for-loops:

```
for i in range(lo,hi):
    # body of loop
```

This will loop for `i` = `lo`, ..., `hi-1`.
- `range(n)` means `range(0,n)`
- `for i in range(len(A))` will loop with `i` = `0`, ..., `len(A)-1`.

this is how we write if-then-else statements:

```
if condition:
    # code for if case
else:
    # code for else case
```

Python indentation

Note the body of the `for` loop needs to be **indented to the right of the `for`!**

The loop ends when we stop being indented to right of the `for` .

Similarly, the code for the `if` case needs to be **indented to the right of the `if`!**

The `if` case ends when we stop being indented to right of the `if` .

```
def search(A, k):
    found = -1
    for i in range(len(A)):
        if A[i] == k:
            found = i
    return found
```

body of `for` loop

Comments in Python start with #

this is how we write for loops:

```
for i in range(lo,hi):
    # body of loop
```

This will loop for `i` = `lo`, ..., `hi`-1.

- `range(n)` means `range(0,n)`
- `for i in range(len(A))` will loop with `i` = 0, ..., `len(A)`-1.

this is how we write if-then-else statements:

```
if condition:
    # code for if case
else:
    # code for else case
```

Python in more detail: variables

Variables do not need to be declared – we just use them:

```
myVar = 0
# some code here that can use and change the value of myVar
```

But we do need to initialise variables before using them (otherwise Python does not know what value to start with!)

```
# code that does not mention myVar
myVar = myVar + 1    # this throws an undefined-name exception
```

correct version:

```
myVar = 0            # initialise myVar to 0 for this example
myVar = myVar + 1
```

Notation: to increase/decrease integers, we can use compact notation, e.g:

```
myVar = 0            # this code works the same as the one above
myVar += 1
```

Python in more detail: if-else

If statements are written as follows (note that ":" is important):

```
if <condition>:
    # code for if-case, must be indented to the right of the if
```

Conditions are written similarly to Java, but with different AND/OR/NOT syntax.

If-else statements:

```
if <condition>:
    # code for if-case, must be indented to the right of the if
else:
    # code for else-case, must be indented to the right of the if
```

for example:

```
if balance == 0:
    # on limit
else:
    # OK
```

```
if not balance >= 0:
    # in debt
else:
    # OK
```

```
if balance + overdraft < 0:
    # in trouble
else:
    # OK
```

Python in more detail: if-(elif)*-else

If we have more than 2 cases to consider, we can stack them together with **elif**'s:

```
if <condition 1>:
    # code 1 (indented to the right of the if)
elif <condition 2>:
    # code 2 (indented to the right of the if)
...
elif <condition n>:
    # code n (indented ...)
else:
    # code n+1 (indented ...)
```

This is shorthand for:

```
if <condition 1>:
    # code 1
else:
    if <condition 2>:
        # code 2
    else:
        ...
        if <condition n>:
            # code n
        else:
            # code n+1
```

For example:

```
if balance > 0:
    print("you are doing good")
elif balance == 0:
    print("you are on the limit")
elif balance + overdraft >= 0:
    print("you are in debt")
else:
    print("you are in trouble")
```

Python in more detail: for loops

For loops are written as follows:

```
for i in <range>:
    # code of the loop, must be indented to the right of the for
```

Ranges are basically the sequences of numbers that we want to loop over.

They are defined in different ways, for example we will be using:

```
for i in range(1,5):   # range is: 1, 2, 3, 4

for i in range(5):   # range is: 0, 1, 2, 3, 4

for i in range(1,5,2):   # range is: 1, 3

for i in range(10,4,-2):   # range is: 10, 8, 6
```

Rule: the start of the range is **in**, the end is **not in**

Python in more detail: while loops

While loops are written as follows:

```
while <condition>:
    # code of the while loop
```

We can also break early from a loop using `break`:

```
i = 10
while True:
    if i == 0:
        break
    print(i)
    i -= 1
```

This will print 10, 9, 8, ..., 1.

We can also use `break` to break out of for loops.

Python in more detail: strings and printing

Strings can be defined directly, e.g. by:

```
myString = "the best string ever"
```

We can also convert values of other types to strings by using the `str` function:

```
result = f(42)   # f is a function from integers to integers
myString = "the result is: "+str(result)
```

We can print strings using the `print` function:

```
result = f(42)
print("the result is: "+str(result))
```

We can also use `print` to directly print e.g. integers:

```
print(f(42))   # this actually calls print(str(f(42))) internally
```

Python in more detail: strings and inputting

We can input strings using the `input` function:

```
name = input("Type in your name: ")
print("the name I got is: "+name)
```

We can the convert strings e.g. to integers using the `int` function:

```
balance = int(input("Enter you account balance: "))
if money < 0:
   message = "you are in trouble "
else:
   message = "you are doing OK "
print(message+name)
```

Python in more detail: arrays

Arrays in Python have similar behaviour to Java arrays:

```
myArray[0] = 42   # set the first element of myArray to 42
```

i.e. we use indexing and square brackets to access the elements of an array.

But we need to initialise arrays before we can use them. E.g. the following is bad:

```
# code that does not mention myArray
myArray[0] = 42    # this throws an undefined-name exception
```

We can initialise arrays using range notation:

```
myArray = [0 for i in range(10)]  # initialises myArray to [0,..,0]

myArray = [i**2 for i in range(10)]  # initialises to [0,1,4,..,81]
```

We can get the length of an array by: `length = len(myArray)`

Warning: Python arrays are actually lists (cf. Lecture 6), but we will only use their array capabilities, i.e. what is included in this slide!

Python in more detail: functions

Python functions are like methods in Java: they receive inputs and return outputs. E.g:

```
def plusOne(n):    # function from integers to integers
   return n+1      # i.e. with integer input and integer output

def printSquare(n):  # integer input, void output
   print(n**2)

def printIfSquare(n):  # integer input, void output
   for i in range(n+1):
      if i**2 == n:
         print("number "+str(n)+" is the square of "+str(i))
         return
      if i**2 > n:
         print("number "+str(n)+" is not a square")
         return
```

Python in more detail: objects

Python is an object-oriented language. A class is defined as follows:

```
class BestPythonClassEver:
    def myFunction1(self, ...):
        ...
    def myFunction2(self, ...):
        ...
    ...
    def myLastFunction(self, ...):
```

Note each class function always has `self` as first argument.

This refers to the object the function is called on (like `this` in Java).

One of the class functions is the constructor, which is `__init__`

Another one is the function `__str__` for converting objects of the class to strings (this is the function called when we write `str(obj)`)

Python in more detail: objects

Here is an example class for 2-dimensional coordinates:

```
import math   # need math module function for square root

class Coordinate:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def distance(self, coord):
        return (math.sqrt((self.x-coord.x)**2+(self.y-coord.y)**2))

    def __str__(self):
        return (str([self.x,self.y])) # put x and y in an array and
                                      # return its string representation
```

Python in more detail: the interpreter

Python is an interpreted language: the code we write are commands that are executed by an **interpreter** one by one:

The interpreter is a program that reads Python code and executes commands – think of this like writing Unix/Windows commands in a terminal.

This means in particular that Python code is not pre-compiled:

- this allows us to write and check small pieces of code very fast
- but we only find out about errors in our code when we actually run it
- Python programs will typically be slower than programs of compiled languages like C++ or even Java

The Jupyter notebook environment

There are different environments to run Python in, we will be using Jupyter notebook:

```
http://jupyter.org/
```

In the QM+ page you can find a notebook file for today's examples.

We will have such a file each week.

Summary

Python will be our algorithms language:

- it is easy for describing algorithms (not much ‘extra’)
- we can write and check code fast on the interpreter
- we need to be careful with indentation
- we need to be careful with type mismatches (e.g. `5+"foo"`) as there is no compiler to warn us