System development life-cycle
Design

Petr Svarny, 2020

From just programming to good programming.

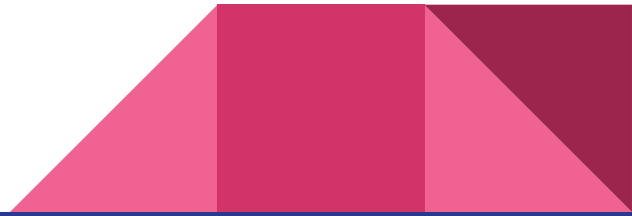
Analysis,
Design,
Tests,
Style.

Analysis to design

Same modeling tools used for analysis can describe the design.

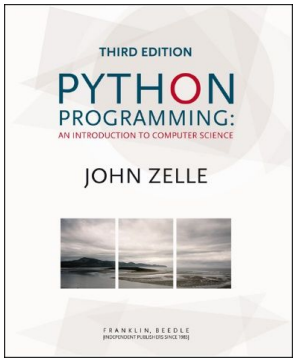
However, what design is good design?

- Computational efficiency
  - Time complexity $\mathcal{O}(n \log n)$
  - Space complexity
- Reliable solution
- Comprehensible code and program logic



Study in Design: Max of Three

- An algorithm to find the largest of three numbers.
- Taken from John Zelle's lectures:



Study in Design: Max of Three

```
def main():
    x_1, x_2, x_3 = eval(input("Please enter three values: "))

    # missing code sets max to the value of the largest

    print("The largest value is", max_val)
```

Strategy 1: Compare Each to All

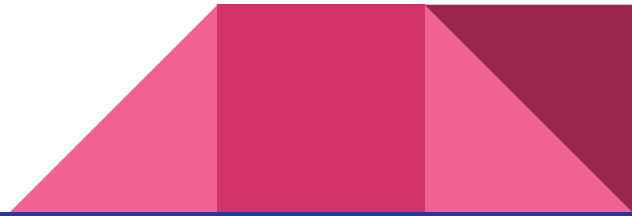
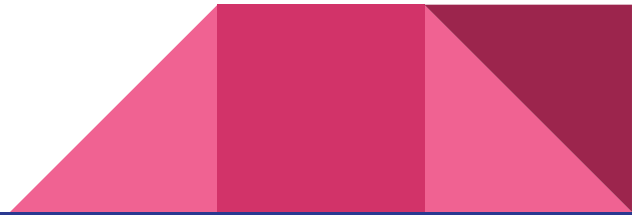
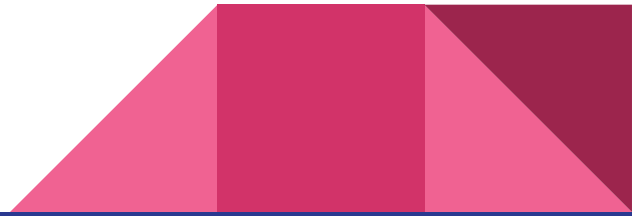
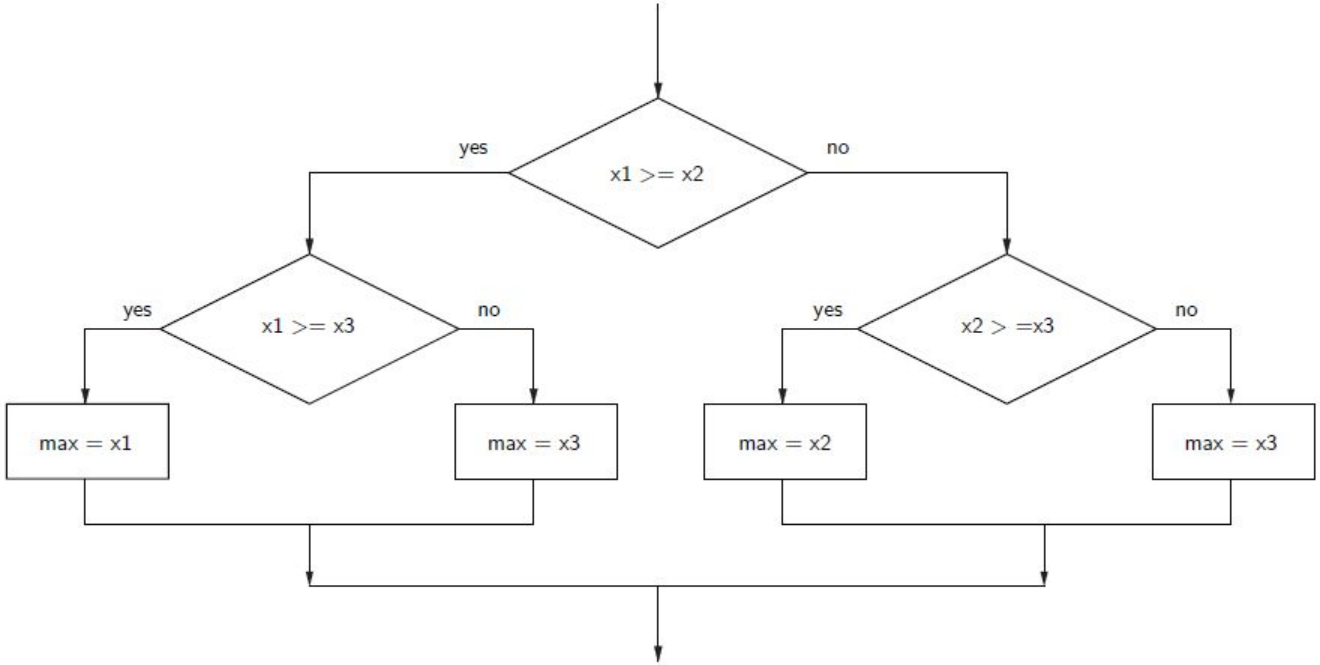
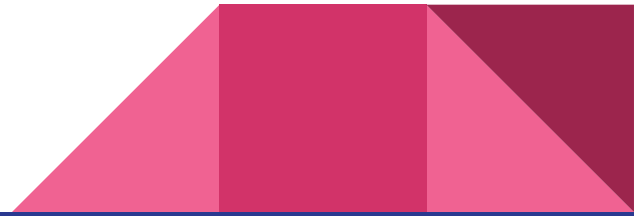
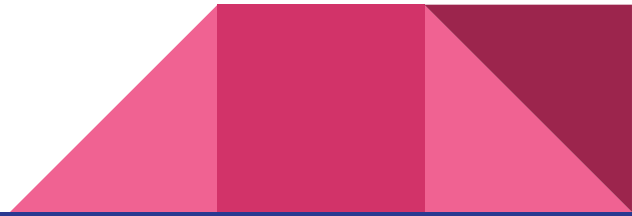
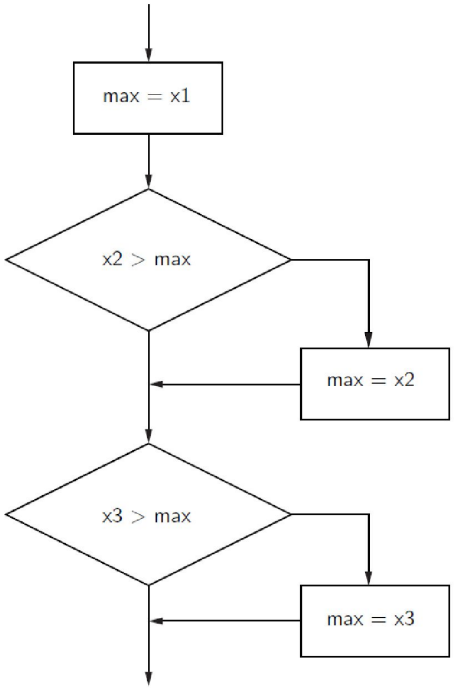
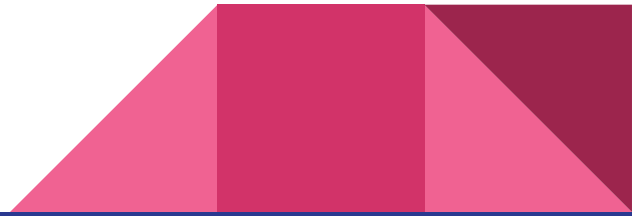
- This looks like a three-way decision, where we need to execute *one* of the following:

```
max_val = x_1
max_val = x_2
max_val = x_3
```

- All we need to do now is preface each one of these with the right condition!

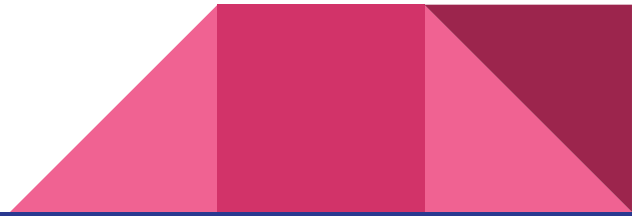
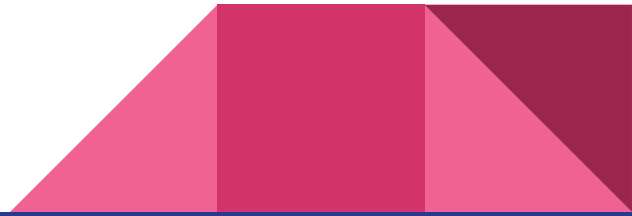
Strategy 1: Compare Each to All

- Let's look at the case where x_1 is the largest.
- ```
  if x_1 >= x_2 >= x_3:
      max_val = x_1
  ```
- Is this syntactically correct?
  - Many languages would not allow this *compound condition*
  - Python does allow it, though. It's equivalent to x_1 ≥ x_2 ≥ x_3.


  ```

Strategy 1: Compare Each to All

- Whenever you write a decision, there are two crucial questions:
  1. When the condition is true, is executing the body of the decision the right action to take?
     - x_1 is at least as large as x_2 and x_3, so assigning max_val to x_1 is OK.
     - Always pay attention to borderline values!
  2. Are we certain that this condition is true in all cases where x_1 is the max?
     - Suppose the values are 5, 2, and 4.
     - Clearly, x_1 is the largest, but does x_1 ≥ x_2 ≥ x_3 hold?
     - We don't really care about the relative ordering of x_2 and x_3, so we can make two separate tests: x_1 >= x_2 *and* x_1 >= x_3.



Strategy 1: Compare Each to All

We can separate these conditions with *and*!

```
if x_1 >= x_2 and x_1 >= x_3:
    max_val = x_1
elif x_2 >= x_1 and x_2 >= x_3:
    max_val = x_2
else:
    max_val = x_3
```

We're comparing each possible value against all the others to determine which one is largest.

Strategy 1: Compare Each to All

- What would happen if we were trying to find the max of five values?
- We would need four Boolean expressions, each consisting of four conditions *and*ed together.
- Yuck!



Strategy 2: Decision Tree

- We can avoid the redundant tests of the previous algorithm using a *decision tree* approach.
- Suppose we start with `x_1 >= x_2`. This knocks either `x_1` or `x_2` out.
- If the condition is true, we need to see which is larger, `x_1` or `x_3`.

Strategy 2: Decision Tree

```
if x_1 >= x_2:
   if x_1 >= x_3:
      max_val = x_1
   else:
      max_val = x_3
else:
   if x_2 >= x_3:
      max_val = x_2
   else
      max_val = x_3
```

Strategy 2: Decision Tree



Strategy 2: Decision Tree

- This approach makes exactly two comparisons, regardless of the ordering of the original three variables.
- However, this approach is more complicated than the first. To find the max of four values you'd need `if-else`s nested three levels deep with eight assignment statements!



Strategy 3: Sequential Processing

- You could probably look at three numbers and just *know* which is the largest. But what if you were given a list of a hundred numbers?
- One strategy is to scan through the list looking for a big number. When one is found, mark it, and continue looking. If you find a larger value, mark it, erase the previous mark, and continue looking.



Strategy 3: Sequential Processing



Strategy 3: Sequential Processing

- This idea can easily be translated into Python.

```
max_val = x_1
if x_2 > max_val:
    max_val = x_2
if x_3 > max_val:
    max_val = x_3
```

Strategy 3: Sequential Programming

- This process is repetitive and lends itself to using a loop.
- We prompt the user for a number, we compare it to our current max, if it is larger, we update the max value, repeat.



Strategy 3: Sequential Programming

```
# program: maxn.py
#   Finds the maximum of a series of numbers

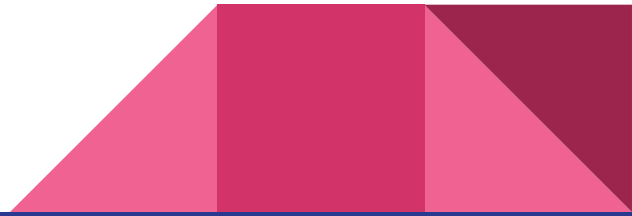
def main():
    n = int(input("How many numbers are there? "))

    # Set max to be the first value
    max_val = float(input("Enter a number >> "))

    # Now compare the n-1 successive values
    for i in range(n-1):
        x = float(input("Enter a number >> "))
        if x > max_val:
            max_val = x
    print("The largest value is", max_val)
```



Strategy 4: Use Python

- Python has a built-in function called `max` that returns the largest of its parameters.

- ```
  def main():
      x_1, x_2, x_3 = eval(input("Please enter three values: "))
      print("The largest value is", max(x_1, x_2, x_3))
  ```


  ```

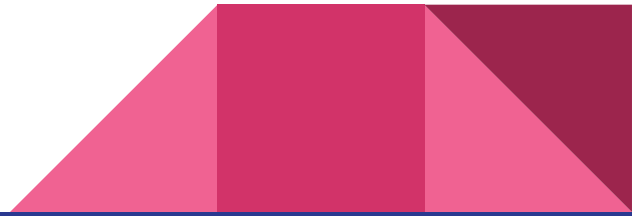
Some Lessons

- Don’t reinvent the wheel.
- Generality is good.
- Be the computer (try to solve it yourself).
- There’s usually more than one way to solve a problem.
  - Decide which suits best your need and don’t get stuck with the first one.
- Keep it simple and stupid (KISS).



Design process (bottom-up)

- Express the algorithm as a series of smaller problems.
- Develop an interface for each of the small problems.
- Detail the algorithm by expressing it in terms of its interfaces with the smaller problems.
- Repeat the process for each smaller problem.

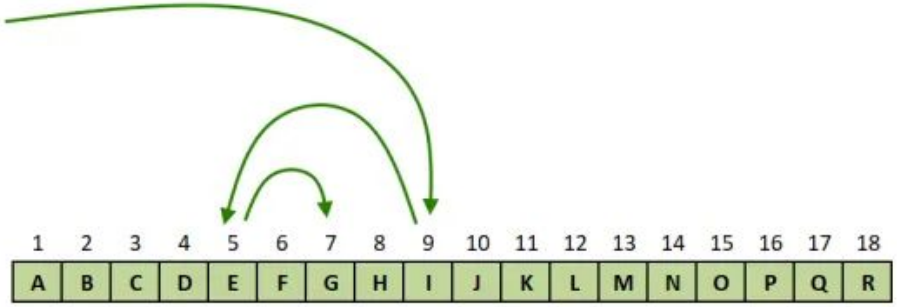


Favourite (toy) problems

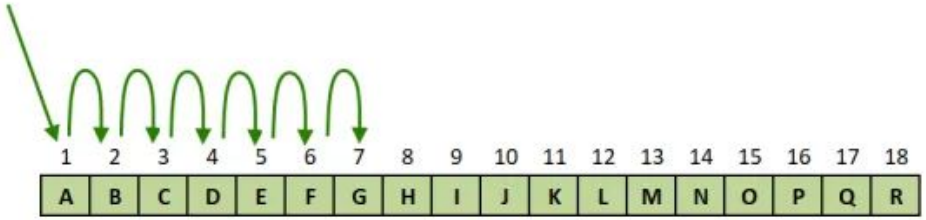
- Search
- Sorting

(in practice often already implemented by someone)

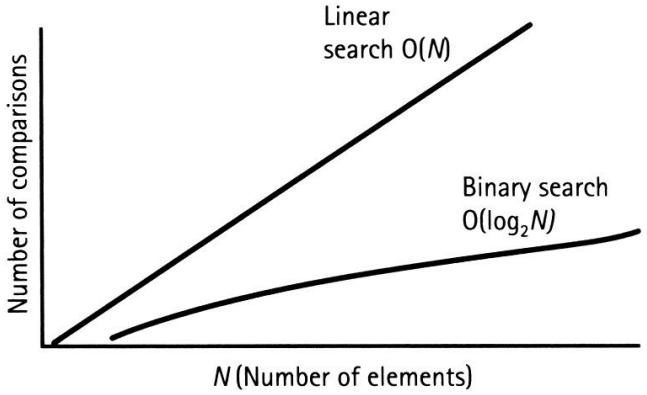
Search: linear vs binary search



Binary Search - Find 'G' in sorted list A-R



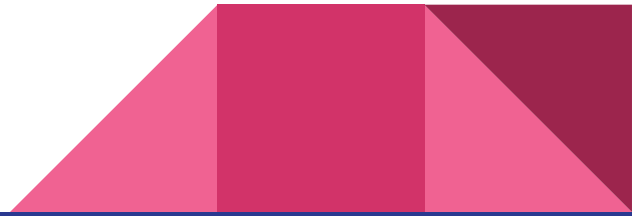
Linear Search - Find 'G' in sorted list A-R



https://algorithms.tutorialhorizon.com/linear-search-vs-binary-search/
https://towardsdatascience.com/a-guide-to-linear-search-and-binary-search-on-arrays-data-structures-algorithms-2c23a74af28a

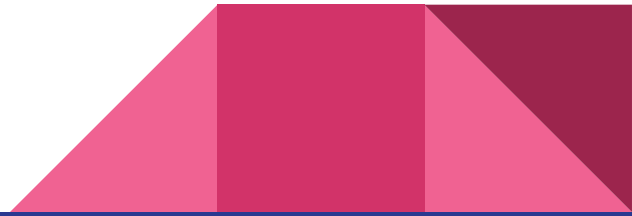
Exercise

- Implement a linear and binary search algorithm in Python
- It should have as an input a sorted list of numbers (e.g. [1, 5, 9]) and it tries to identify the numbers location or return that the number is not in the list
- IMPLEMENT the search, do not just use `in` or similar Python shortcuts ;)



Sort

- Search is most effective on sorted data (can use assumptions) thus sorting is an important tool
- Similar as search, can have many implementations



Sorting

- Sorting algorithm demonstrations
- Should be chosen based on the data and data structures (e.g., binary trees)