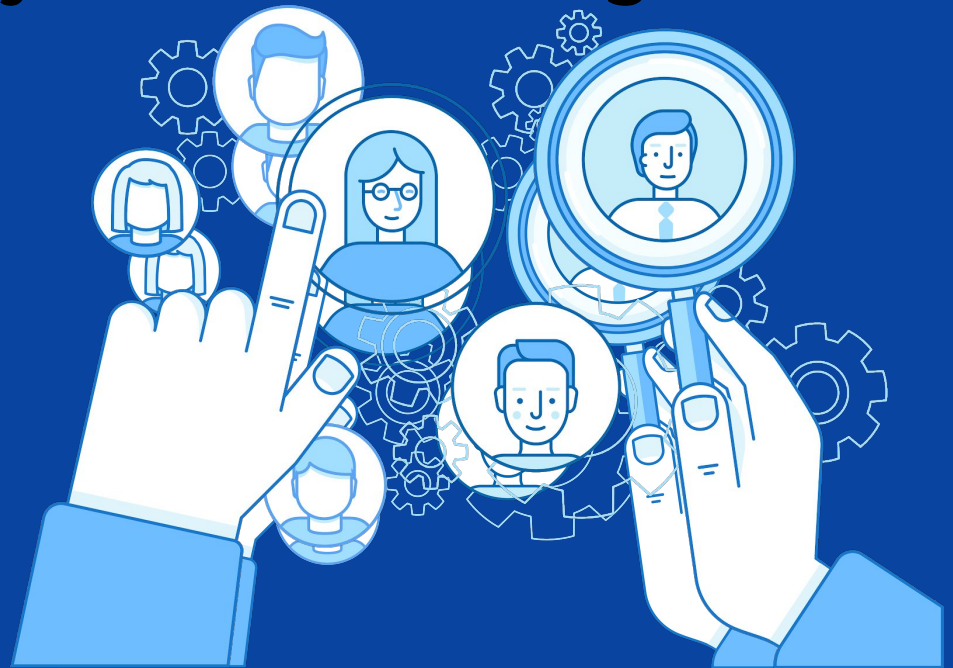
Digital Career Institute

Python Course - Algorithmic Thinking



Digital Career Institute

Goal of the Submodule

The goal of this submodule is to help the learners understand the concepts of algorithms, how to think in terms of algorithms, and what to watch out for when choosing a particular algorithm for a given task.

Topics

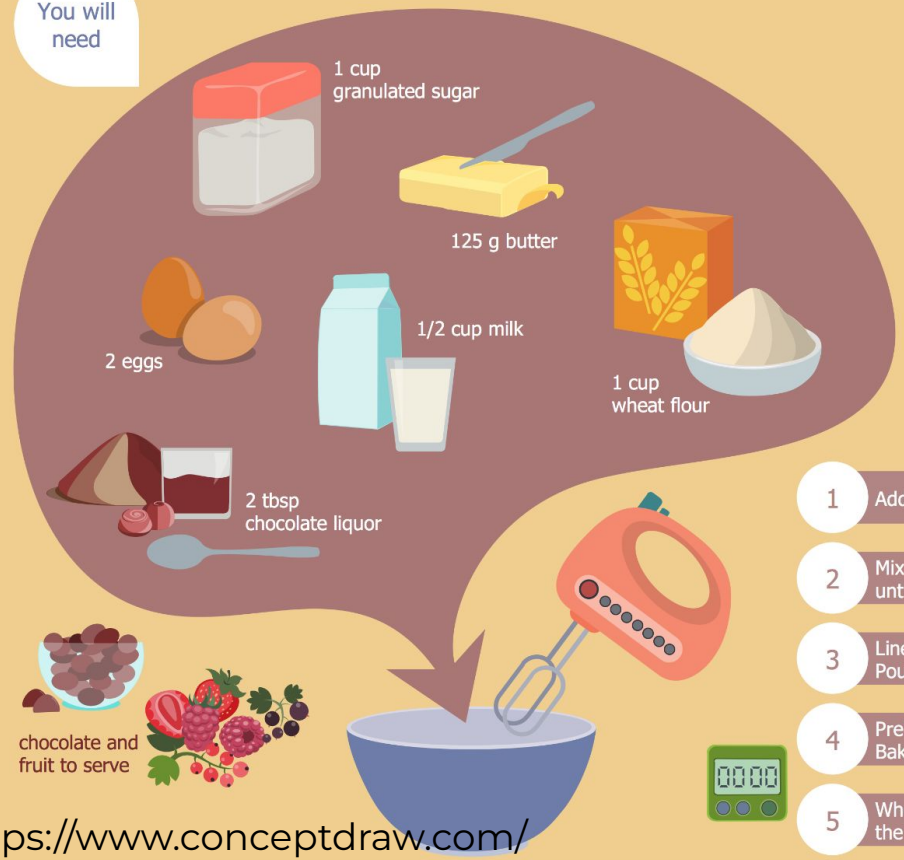
- Algorithms
- Algorithmic thinking in general and in computer programming
- Analyzing speed and memory usage of algorithms
- Quicksort
- Mergesort
- O-notation

Algorithms and Algorithmic Thinking

Digital Career Institute

CHOCOLATE CAKE

You will need



Method

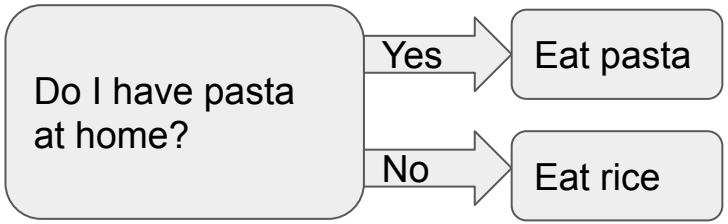
1. Add all the ingredients in a medium sized bowl.
2. Mix with an electric mixer for about 2-3 minutes or until the mixture is smooth.
3. Line a 20 cm round cake tin with baking paper. Pour the cake mixture into the tin.
4. Preheat the oven to 180 degrees C. Bake for approximately 45 minutes.
5. When you remove the cake from oven, let it cool in the pan for about 5 minutes. Leave to cool on a rack.

Source: https://www.conceptdraw.com/

Algorithms - What Is an Algorithm

- A step-by-step guide for the computer to be followed when in a particular situation.
- Can be used for decision making processes.
  - Example: What stories to show to a particular user of a social network.
- Can be used as instructions to go from the current state of things to a desired result.
- Humans use algorithms in their lives in order to make decisions or act in a specific way to achieve a predetermined result:

What should I eat?



Decision making

I want to eat rice. How do I prepare it?

Reach specific result

Algorithmic Thinking

Digital Career Institute

- Splitting a task into single step instructions.
- Make precise instructions for how to arrive at a decision.
- Make a process replicable by following instructions.

Easier to turn into algorithm	**Harder to turn into algorithm**

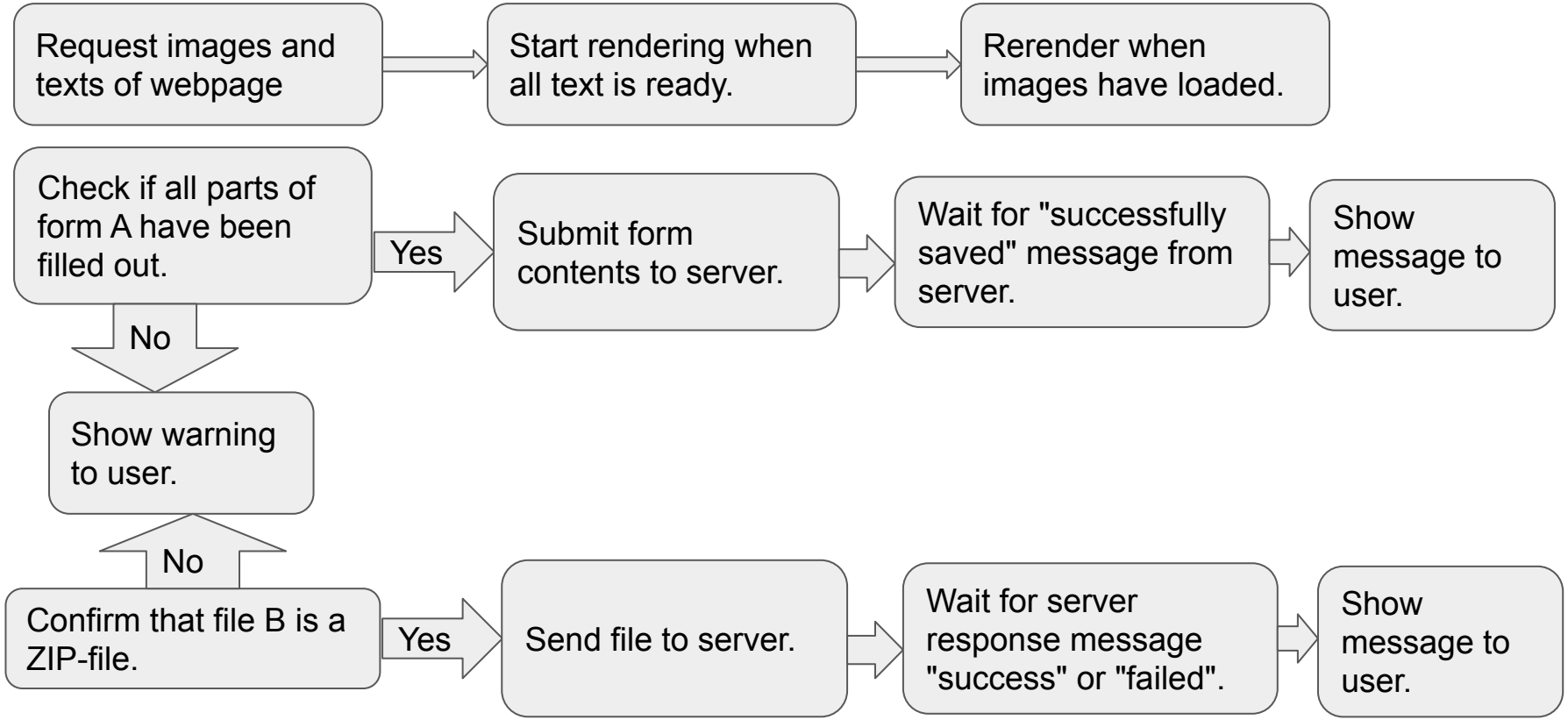
● How does one cook instant noodles?
● How should one drive in the UK?
● How does one enter the US/EU?
  ○ Which visa should one choose?
  ○ What order should one follow?
● Should you eat pasta today?
  ○ Do you have pasta available?
  ○ Have you eaten pasta the past 7 days in a row?
	● How can you become a world class famous chef?
	● Which clothing should you wear today?
	● How does one become a millionaire on the stock market?
	● What is the meaning of life?
	● Which user interface alternative of a given app will users prefer?

Algorithms in Computer Programming

Digital Career Institute

- A snippet of a computer program, telling the program how to behave in a specific situation.
- Possible situations:
  - User has entered the page.
  - User clicked a button.
  - User has started to scroll.
  - User has been inactive for X seconds.
  - User has uploaded file Y.
  - The credit card number the user has provided has been denied by the issuing bank.

Typical Programming Algorithms



Algorithms in Computer Programming

Digital Career Institute

- Very often, the operations required in the code are very common, some other times these operations are rarer.
- Operations of lower level (doing a single well-defined and specific thing) tend to be very common operations. Whereas higher level operations (implementing our business logic) tend to be specific to our application.
- Common low-level operations and algorithms have been extensively analysed and one or more implementations can be found in almost any programming language.
- Examples of common operations:
    - Sorting a list of items.
    - Searching for an item in a sequence.
    - Comparing two items.
    - Hashing an item.
- When an algorithm is **generic** and there are **different implementations** available, they tend to come with **advantages and trade-offs**, often involving a difference in performance and quality of the output.

At the Core of the Lesson

Digital Career Institute

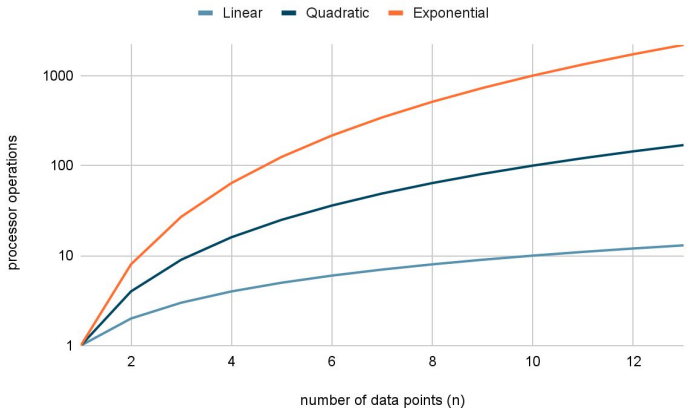
Algorithms and algorithmic thinking

- Humans use algorithms as recipes of how to behave in certain situations or to obtain a specific result.
- Algorithmic thinking involves describing actions to be taken in a step-by-step guide, thereby making it reproducible for others.
- Some things we do are more easily translated into algorithms than others.
- Computer programs consist of many algorithms.
- Most of the low-level algorithms, dealing with basic operations, have been extensively studied and there are often one or more implementations available.

Algorithm Analysis

Algorithm Analysis

- Look at an algorithm and predict how much longer the algorithm will take or how much more memory it will require if it is run on more or more complex arrays/objects.
- More advanced: takes I/O and net activity into account.
- Try to avoid using algorithms that will cause resource increase in production settings.
- Test data and test settings on the laptops of developers are often smaller than real world production data - therefore problems of algorithms that do not do well with larger amounts of data will often not be directly visible before the application has been deployed.
- Execution time and memory usage will almost always grow when processing more data. What matters is the rate of growth:
  - linear
  - quadratic
  - exponential
  - etc.



Algorithm Analysis

Digital Career Institute

Which algorithm to choose (run-time)?

```
def prefix_averages(x: list[int]) -> list[int]:
      n = len(x)
      a = []
      for i in range(0, n):
            s = 0
            for j in range(0, i+1):
                  s = s + x[j]
            a.append(s / (i + 1))
      return a
```

```
def prefix_averages(x: list[int]) -> list[int]:
      n = len(x)
      a = []
      s = 0
      for i in range(0, n):
            s = s + x[i]
            a.append(s / (i + 1))
      return a
```

Algorithm Analysis

Digital Career Institute

Which algorithm to choose (memory)?

```
def compare_lists(x: list[int], y: list[int]) ->
list[object]:
      n = len(x)
      m = len(y)
      a = []
      for i in range(0, n):
            b = []
            for j in range(0, m):
                  b.append([x[i], y[j]])
            a.append(b)
      c = []
      for a_item in a:
            d = []
            for b_item in a_item:
                  d.append(b_item[0] == b_item[1])
            c.append(d)
      return c
```

```
def compare_lists(x: list[int], y: list[int]) ->
list[object]:
      a = []
      for i in x:
            b = []
            for j in y:
                  b.append(i == j)
            a.append(b)
      return a
```

Algorithm Analysis

Digital Career Institute

Aspects to consider:

- **Memory:** How much information needs to be stored in memory for data of size X.
- **Run-time:**
  - Count primitive operations.
  - Look for loops on input data:
    - single loops: linear growth.
    - loops inside of loops: quadratic growth.
    - multiple level of loops: exponential growth.
- How algorithm will behave:
  - worst-case - important: can cause app to halt/slow down significantly.
  - best case.
  - average.
  - amortized (averaged over time).

Algorithm Analysis

Digital Career Institute

Worst case vs. best case

```
def x_contains_y(x: list[int], y: int) -> bool:
      n = len(x)
      for i in range(0, n):
            if x[i] == y:
                  return True
      return False
```

- Best case: y == x[0].
- Worst case: y not in x.
- **Worst case creates a problem** so we mostly focus on worst case scenarios.

At the Core of the Lesson

- Algorithm analysis is used to quickly spot problems in programs when the amount of data increases.
- Program run-time and memory usage are key points to look at.
- We look at best case, worst case and averages, but worst-case scenarios are extra important as they can create the biggest kinds of problems.