Software Engineering materials:

SE Foundations - Overview

First Trimester:

* Intro to:
  + Emacs
  + Vim
  + Git/Github
* C
  + Basics
  + Functions
  + Pointers
  + Data Structures
    - Singly Linked Lists
    - Arrays
* Bit Manipulation
* Variadic Functions
* Memory Management
* File Redirection
* printf
* Shell
* Compilation
* Heap/Stack
* Linux
  + Basics
  + Permissions
  + Redirections
  + Syscalls
* Static Libraries
* Social Media
* Public Speaking/Stand ups

Second Trimester:

* Python
* Basics
* Object-Oriented Programming
* Interpreted Languages
* Test Driven Development
* Error Handling
* I/O
* Web Scraping
* C
  + Data Structures
    - Stacks/Queues
    - Doubly Linked Lists
    - Hash Tables
    - Binary Trees
  + Algorithms
    - Sorting
    - Big O
  + Dynamic Libraries
* Linux
  + Processes and Signals
  + Scripting
  + Command Line
* Databases
  + SQL
  + MySQL
* Systems Engineering
  + Open Systems Interconnection Model
  + TCP/IP
* Airbnb Project
  + HTML
  + CSS
  + JSON
  + Unit Testing

Third Trimester:

* Systems Engineering/Devops
  + SSH
  + SSL/TLS
  + HTTP/S
  + Webstack Debugging
  + Servers
  + Web Servers (Nginx)
  + Application Servers (Gunicorn)
  + Configuration Management (Puppet)
  + Load Balancers
  + Postmortems
  + Networking
  + Monitoring
* C
  + Algorithms
    - Searching
* Airbnb Project
  + Javascript
  + JQuery
  + Deployment
  + APIs
    - Flask
* Hack Days
* Python
  + ORMs (SQLAlchemy)
  + Templating (Jinja2)
* Code Debugging
* Regexes

All Three Trimesters:

* Technical Writing
* Recursion
* Collaboration
* The Framework
* Whiteboarding
* Mock Interviews
* Time Management

Additional Information

After you complete the Foundations program, a whole new series of paths open up for you. Known as “Specializations”, you can pick where you will take your focus for the fourth sprint of the curriculum.

We have two Specializations:

* Back-end
* Front-end

======================================================

Short Specializations - Overview

Front-end

Specialization

* Modern JavaScript
  + ES6
  + TypeScript
* Front-end
  + Advanced HTML
  + Advanced CSS
  + Responsive design
* React I
  + Intro
  + Props
* React II
  + State
  + Immutable
  + Inline-styling
* React Redux
  + Action creator & normalizr
  + Reducer & Selector
  + Connector & Provider
* Portfolio project

Soft-skills & Interview preparation

* Technical Writing
* Collaboration
* The Framework
* Whiteboarding
* Time Management
* Interview algorithm practice

Back-end

Specialization

* Modern JavaScript
  + ES6
  + TypeScript
  + Node JS basic
  + Unittests
* Python
  + Variable annotations
  + Async
  + Unittests and integration tests
* Storage
  + Advanced MySQL
  + NoSQL
  + Redis
* Back-end
  + Pagination
  + Caching
  + i18n
  + Queuing system in JS
* User Data
  + Personal data
  + Authentication - Basic & Session
  + User authentication
* Portfolio project

Soft-skills & Interview preparation

* Technical Writing
* Collaboration
* The Framework
* Whiteboarding
* Time Management
* Interview algorithm practice

========================================================

[Collapse all](https://intranet.alxswe.com/projects/current)

[Interview Preparation](https://intranet.alxswe.com/projects/current#collapse-block-track-48)

[Interview Preparation - Interview Preparation](https://intranet.alxswe.com/projects/current#collapse-block-134)

1. 1213 [0x00. Pascal's Triangle](https://intranet.alxswe.com/projects/1213) *[Jun 26 23...Jun 30 23]* - **50.0%**

* 0x00. Pascal's Triangle
* **AlgorithmPython**
* Weight: 1
* Project over - took place from Jun 26, 2023 6:00 AM to Jun 30, 2023 6:00 AM
* An auto review will be launched at the deadline
* In a nutshell…
* **Auto QA review:** 5.5/11 mandatory
* **Altogether:**  **50.0%**
* Mandatory: 50.0%
* Optional: no optional tasks
* Resources
* [What is Pascal’s triangle](https://intranet.alxswe.com/rltoken/F458nFkW9StJum2zPI4khg)
* [Pascal’s Triangle - Numberphile](https://intranet.alxswe.com/rltoken/XXMN2RVCCGcF5l5ZnUIv8Q)
* [What are Python Algorithms](https://intranet.alxswe.com/rltoken/q5v0xbgrVxG4Nf-fV-BW2w)
* Additional Resources
* [Mock Technical Interview](https://intranet.alxswe.com/rltoken/vKf7Spm4xxFMom3x4Jx52g)
* Must Know
* To successfully complete this project, you should revise the following Python concepts:
* **Lists and List Comprehensions**:
* Understand how to create, access, modify, and iterate over lists.
* Utilize list comprehensions for more concise and readable code, especially for generating rows of Pascal’s Triangle.
* **Functions**:
* Know how to define and call functions.
* Pass parameters and return values, particularly how to return a list of lists representing Pascal’s Triangle.
* **Loops**:
* Use **for** and **while** loops to iterate through sequences.
* ======================

In Python, both `for` and `while` loops are used for iteration, but they serve different purposes and have different structures. Here’s a breakdown of their differences:

###**for Loop**

- \*\*Purpose\*\*: **Used to iterate over a sequence (like a list, tuple, string, or range).**

- \*\*Structure\*\*: The loop runs for each item in the sequence.

- \*\*Syntax\*\*:

```python

for variable in sequence:

# code to execute

```

- \*\*Example\*\*:

```python

fruits = ['apple', 'banana', 'cherry']

**for** fruit **in** fruits:

print(fruit)

```

### `while` Loop

- \*\*Purpose\*\*: **Used to repeat a block of code as long as a specified condition is true**.

- \*\*Structure\*\*: The loop continues until the condition evaluates to `False`.

- \*\*Syntax\*\*:

```python

while condition:

# code to execute

```

- \*\*Example\*\*:

```python

count = 0

while count < 5:

print(count)

count += 1

```

### Key Differences

1. \*\***Iteration vs. Condition**\*\*:

- `for` loops are used when the number of iterations is known or when iterating over a known sequence.

- `while` loops are used when the number of iterations is not predetermined and depends on a condition.

2. \*\***Control**\*\*:

- In a `for` loop, control is based on the sequence being iterated.

- In a `while` loop, control is based on a condition that must be checked before each iteration.

3. \*\***Use Cases**\*\*:

- Use a `for` loop for iterating through collections or ranges.

- Use a `while` loop for scenarios where you need to repeat actions until a specific condition changes.

### Conclusion

Choose `for` when you know the number of iterations or are iterating over a collection and choose `while` when you need to repeat actions based on a condition.

================================

* Nested loops may be necessary for generating each row and calculating the values of Pascal’s Triangle.
* **Conditional Statements**:
* Apply if, elif, and else conditions to implement logic based on the position within Pascal’s Triangle (e.g., the edges of the triangle always being 1).
* **Recursion (Optional)**:
* While not strictly necessary, understanding recursion can provide an alternative approach to generating Pascal’s Triangle.
* Recognize base cases and recursive cases for a function that generates the triangle’s rows.
* **Arithmetic Operations**:
* Perform addition, a fundamental operation for calculating each element of Pascal’s Triangle as the sum of the two elements directly above it.
* **Indexing and Slicing**:
* Access elements and slices of lists, crucial for identifying and summing the correct elements when constructing each row of the triangle.
* **Memory Management**:
* Be mindful of how lists are stored and copied, especially when creating new rows based on the values of the previous row.
* **Error and Exception Handling (Optional)**:
* Use try-except blocks as needed to handle potential errors, such as invalid input types or values.
* **Efficiency and Optimization**:
* Consider the time and space complexity of different approaches to generating Pascal’s Triangle.
* Evaluate and apply optimizations to improve the performance of the solution.
* By revisiting these concepts, you will be well-prepared to tackle the challenges of implementing Pascal’s Triangle in Python, applying both your mathematical understanding and programming skills to develop an efficient and effective solution.
* Tasks
* 0. Pascal's Triangle
* **mandatory**
* Score: 50.0% (*Checks completed: 100.0%*)
* Create a function def pascal\_triangle(n): that returns a list of lists of integers representing the Pascal’s triangle of n:
* Returns an empty list if n <= 0
* You can assume n will be always an integer
* guillaume@ubuntu:~/0x00$ cat 0-main.py
* #!/usr/bin/python3
* """
* 0-main
* """
* pascal\_triangle = \_\_import\_\_('0-pascal\_triangle').pascal\_triangle
* def print\_triangle(triangle):
* """
* Print the triangle
* """
* for row in triangle:
* print("[{}]".format(",".join([str(x) for x in row])))
* if \_\_name\_\_ == "\_\_main\_\_":
* print\_triangle(pascal\_triangle(5))
* guillaume@ubuntu:~/0x00$
* guillaume@ubuntu:~/0x00$ ./0-main.py
* [1]
* [1,1]
* [1,2,1]
* [1,3,3,1]
* [1,4,6,4,1]
* guillaume@ubuntu:~/0x00$
* **Repo:**
* GitHub repository: alx-interview
* Directory: 0x00-pascal\_triangle
* File: 0-pascal\_triangle.py

1. 1214 [0x01. Lockboxes](https://intranet.alxswe.com/projects/1214) *[Jul 3 23...Jul 7 23]* - **100.0%**
2. 1215 [0x02. Minimum Operations](https://intranet.alxswe.com/projects/1215) *[Jul 10 23...Jul 14 23]* - **59.58%**
3. 1216 [0x03. Log Parsing](https://intranet.alxswe.com/projects/1216) *[Jul 17 23...Jul 21 23]* - **50.0%**
4. 1217 [0x04. UTF-8 Validation](https://intranet.alxswe.com/projects/1217) *[Jul 24 23...Jul 28 23]* - **50.0%**
5. 1218 [0x05. N Queens](https://intranet.alxswe.com/projects/1218) *[Jul 31 23...Aug 4 23]* - **67.33%**
6. 1219 [0x06. Star Wars API](https://intranet.alxswe.com/projects/1219) *[Aug 7 23...Aug 11 23]* - **100.0%**
7. 1220 [0x07. Rotate 2D Matrix](https://intranet.alxswe.com/projects/1220) *[Aug 14 23...Aug 18 23]* - **100.0%**
8. 1221 [0x08. Making Change](https://intranet.alxswe.com/projects/1221) *[Aug 21 23...Aug 25 23]* - **100.0%**
9. 1222 [0x09. Island Perimeter](https://intranet.alxswe.com/projects/1222) *[Aug 28 23...Sep 1 23]* - **100.0%**
10. 1223 [0x0A. Prime Game](https://intranet.alxswe.com/projects/1223) *[Sep 4 23...Sep 8 23]* - **81.25%**

[Front-end](https://intranet.alxswe.com/projects/current#collapse-block-track-49)

[Front-end - Modern JavaScript](https://intranet.alxswe.com/projects/current#collapse-block-135)

* 1186 [0x00. ES6 Basics](https://intranet.alxswe.com/projects/1186)
* 1187 [0x01. ES6 Promises](https://intranet.alxswe.com/projects/1187)
* 1188 [0x02. ES6 classes](https://intranet.alxswe.com/projects/1188)
* 1189 [0x03. ES6 data manipulation](https://intranet.alxswe.com/projects/1189)
* 1190 [0x04. Typescript](https://intranet.alxswe.com/projects/1190)

[Front-end - Front-end](https://intranet.alxswe.com/projects/current#collapse-block-137)

* 1191 [0x00. Advanced HTML](https://intranet.alxswe.com/projects/1191)
* 1192 [0x01. Developer tools](https://intranet.alxswe.com/projects/1192)
* 1193 [0x02. Advanced CSS](https://intranet.alxswe.com/projects/1193)
* 1194 [0x03. Responsive design](https://intranet.alxswe.com/projects/1194)

[Front-end - React](https://intranet.alxswe.com/projects/current#collapse-block-138)

* 1195 [0x00. Webpack](https://intranet.alxswe.com/projects/1195)
* 1196 [0x01. React intro](https://intranet.alxswe.com/projects/1196)
* 1197 [0x02. React props](https://intranet.alxswe.com/projects/1197)
* 1198 [0x03. React component](https://intranet.alxswe.com/projects/1198)
* 1199 [0x04. React inline styling](https://intranet.alxswe.com/projects/1199)
* 1200 [0x05. React state](https://intranet.alxswe.com/projects/1200)
* 1201 [0x06. React Immutable](https://intranet.alxswe.com/projects/1201)
* 1202 [0x07. React Redux action creator+normalizr](https://intranet.alxswe.com/projects/1202)
* 1203 [0x08. React Redux reducer+selector](https://intranet.alxswe.com/projects/1203)
* 1204 [0x09. React Redux Connectors and Providers](https://intranet.alxswe.com/projects/1204)

[Front-end - More](https://intranet.alxswe.com/projects/current#collapse-block-140)

* 1207 [Markdown to HTML](https://intranet.alxswe.com/projects/1207)
* 1208 [Flexbox](https://intranet.alxswe.com/projects/1208)
* 1209 [Forms](https://intranet.alxswe.com/projects/1209)
* 1210 [Accessibility](https://intranet.alxswe.com/projects/1210)
* 1211 [Fun with CSS](https://intranet.alxswe.com/projects/1211)
* 1212 [Sass & Scss](https://intranet.alxswe.com/projects/1212)

[Back-end](https://intranet.alxswe.com/projects/current#collapse-block-track-50)

[Back-end - Modern JavaScript](https://intranet.alxswe.com/projects/current#collapse-block-136)

* 1224 [0x00. ES6 Basics](https://intranet.alxswe.com/projects/1224) *[Jun 26 23...Jun 27 23]* - **75.0%**
* 1225 [0x01. ES6 Promises](https://intranet.alxswe.com/projects/1225) *[Jun 27 23...Jun 29 23]* - **50.24%**
* 1226 [0x02. ES6 classes](https://intranet.alxswe.com/projects/1226) *[Jun 29 23...Jun 30 23]* - **63.63%**
* 1227 [0x03. ES6 data manipulation](https://intranet.alxswe.com/projects/1227) *[Jul 3 23...Jul 5 23]* - **200.0%**
* 1228 [0x04. Typescript](https://intranet.alxswe.com/projects/1228) *[Jul 5 23...Jul 6 23]* - **49.93%**
* 1243 [0x05. NodeJS Basics](https://intranet.alxswe.com/projects/1243) *[Aug 21 23...Aug 23 23]* - **74.69%**
* 1244 [0x06. Unittests in JS](https://intranet.alxswe.com/projects/1244) *[Aug 23 23...Aug 25 23]* - **100.0%**

[Back-end - Python](https://intranet.alxswe.com/projects/current#collapse-block-141)

* 1229 [0x00. Python - Variable Annotations](https://intranet.alxswe.com/projects/1229) *[Jul 6 23...Jul 7 23]* - **160.0%**
* 1230 [0x01. Python - Async](https://intranet.alxswe.com/projects/1230) *[Jul 10 23...Jul 11 23]* - **65.0%**
* 1231 [0x02. Python - Async Comprehension](https://intranet.alxswe.com/projects/1231) *[Jul 11 23...Jul 12 23]* - **65.0%**
* 1237 [0x03. Unittests and Integration Tests](https://intranet.alxswe.com/projects/1237) *[Jul 27 23...Aug 1 23]* - **100.97%**

[Back-end - Storage](https://intranet.alxswe.com/projects/current#collapse-block-142)

* 1232 [0x00. MySQL advanced](https://intranet.alxswe.com/projects/1232) *[Jul 12 23...Jul 14 23]* - **61.32%**
* 1233 [0x01. NoSQL](https://intranet.alxswe.com/projects/1233) *[Jul 17 23...Jul 19 23]* - **60.51%**
* 1234 [0x02. Redis basic](https://intranet.alxswe.com/projects/1234) *[Jul 19 23...Jul 20 23]* - **50.0%**

[Back-end - Back-end](https://intranet.alxswe.com/projects/current#collapse-block-143)

* 1235 [0x00. Pagination](https://intranet.alxswe.com/projects/1235) *[Jul 20 23...Jul 25 23]* - **50.0%**
* 1236 [0x01. Caching](https://intranet.alxswe.com/projects/1236) *[Jul 25 23...Jul 27 23]* - **50.0%**
* 1238 [0x02. i18n](https://intranet.alxswe.com/projects/1238) *[Aug 1 23...Aug 2 23]* - **88.89%**
* 1245 [0x03. Queuing System in JS](https://intranet.alxswe.com/projects/1245) *[Aug 28 23...Aug 31 23]* - **200.0%**
* 1246 [0x04. Files manager](https://intranet.alxswe.com/projects/1246) *[Aug 31 23...Sep 7 23]* - **195.72%**

[Back-end - User Data](https://intranet.alxswe.com/projects/current#collapse-block-144)

* 1239 [0x00. Personal data](https://intranet.alxswe.com/projects/1239) *[Aug 2 23...Aug 4 23]* - **52.86%**
* 1240 [0x01. Basic authentication](https://intranet.alxswe.com/projects/1240) *[Aug 7 23...Aug 9 23]* - **73.81%**
* 1241 [0x02. Session authentication](https://intranet.alxswe.com/projects/1241) *[Aug 9 23...Aug 11 23]* - **103.34%**
* 1242 [0x03. User authentication service](https://intranet.alxswe.com/projects/1242) *[Aug 14 23...Aug 18 23]* - **195.5%**

[Portfolio project](https://intranet.alxswe.com/projects/current#collapse-block-track-68)

[Portfolio project - Portfolio project](https://intranet.alxswe.com/projects/current#collapse-block-145)

* 1247 [Webstack - Portfolio Project - Pitch](https://intranet.alxswe.com/projects/1247) *[Aug 28 23...Sep 8 23]* - **100.0%**
* 1248 [Webstack - Portfolio Project](https://intranet.alxswe.com/projects/1248) *[Sep 8 23...Oct 9 23]* - **200.0%**

**===================================================================**

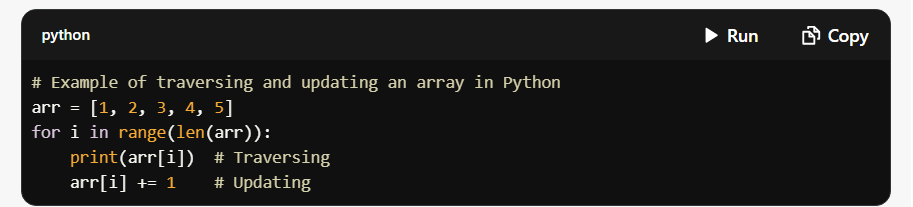
**Python studding notes from hacker rank:**

**Basic Data Structures** - Use data structures such as **arrays** and **strings**. **Traverse through arrays**, **strings, trees, and linked lists**. **Access and update individual elements in arrays, and characters in strings.**  
**Basic Algorith**ms (such as **sorting and searching**) - **Create simple sorting algorithms** such as **bubble sort, merge sort, and counting sort. Create simple brute force and sub-optimal solutions**.

---------------------

**Basic Data Structures**

1. **Arrays**
   * A collection of items stored at contiguous memory locations.
   * **Traversal**: Accessing each element in the array.
   * **Accessing Elements**: Using the index, e.g., array[i].
   * **Updating Elements**: Assigning a new value, e.g., array[i] = newValue.



**Strings**

* A sequence of characters.
* **Traversal**: Iterating through each character.
* **Accessing Characters**: Using the index, e.g., string[i].
* **Updating Characters**: Strings are immutable in Python, but you can create a new string with the changes.

A black screen with white text

Description automatically generated

**Linked Lists**

* A collection of nodes where each node contains data and a reference to the next node.
* **Traversal**: Following the links from one node to the next.
* **Accessing/Updating**: Requires traversal to find the node.

A screen shot of a computer program

Description automatically generated

**Trees**

* A hierarchical structure with nodes, where each node has a value and references to child nodes.
* **Traversal**: Common methods include in-order, pre-order, and post-order.

A screenshot of a computer program

Description automatically generated

**Basic Algorithms**

1. **Sorting Algorithms**
   * **Bubble Sort**: Repeatedly steps through the list, compares adjacent elements, and swaps them if they are in the wrong order.

A black screen with white text

Description automatically generated

**Merge Sort**: Divides the array into halves, sorts each half, and merges them back together.

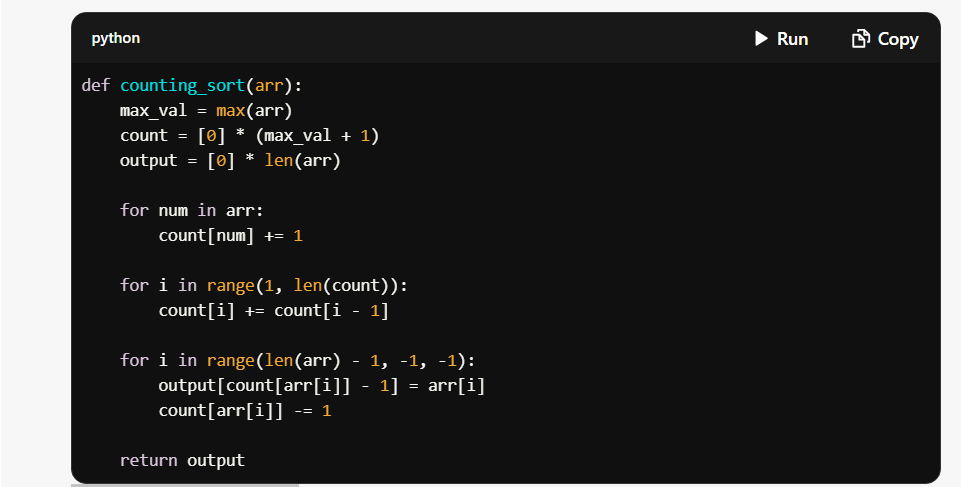
A computer screen shot of a program code

Description automatically generated

A black rectangular object with white text

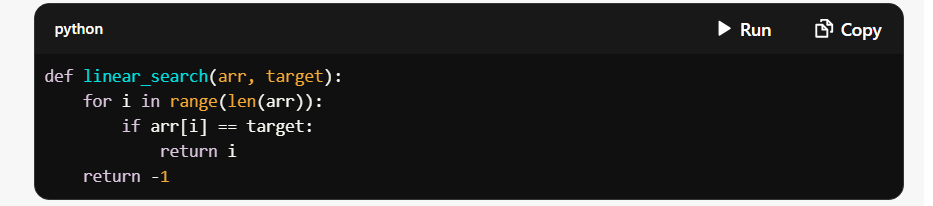
Description automatically generated

**Counting Sort**: Counts occurrences of each value and calculates the position of each value in the sorted array.



Searching Algorithms

Linear Search: Checks each element until the target is found.



**Binary Search**: Efficiently finds an element in a sorted array by repeatedly dividing the search interval in half.

A screenshot of a computer program

Description automatically generated

Summary

Understanding these basic data structures and algorithms is essential for efficient programming and problem-solving. You can implement these concepts in various programming languages, adapting the syntax as needed. If you have specific questions or need further explanations, feel free to ask!

-------------------------------------------------------------------------------------------