TIP102 | Intermediate Technical Interview Prep

Intermediate Technical Interview Prep Spring 2025 (@ Section 3 | Tuesdays and Thursdays 6PM - 8PM PT)
Personal Member ID#: **117667**

Session 1: Strings & Arrays

Session Overview

Students will be introduced to the foundational concepts of strings and arrays in Python, which are essential for solving common coding problems. They will also learn about the UPI method, a structured approach to planning and solving technical interview questions. The lesson will cover basic operations like accessing, iterating over, and modifying lists, as well as performing advanced string manipulation.

> You can find all resources from today including session slide decks, session recordings, and more on the resources tab

🎢 Part 1 : Instructor Led Session

We'll spend the first portion of the synchronous class time in large groups, where the instructor will lead class instruction for 30-45 minutes.

🧑‍💻 Part 2: Breakout Session

In breakout sessions, we will explore and collaboratively solve problem sets in small groups. Here, the **collaboration, conversation, and approach** are just as important as "solving the problem" - please engage warmly, clearly, and plentifully in the process!

In breakout rooms you will:

- Screen-share the problem/s, and verbally review them together
- Screen-share an interactive coding environment, and talk through the steps of a solution approach
    - ProTip: - An Integrated Development Environment (IDE) is a fancy name for a tool you could use for shared writing of code - like Replit.com, Collabed.it, CodePen.io, or other - your staff team will specify which tool to use for this class!
- Screen-share an implementation of your proposed solution
- Independently follow-along, or create an implementation, in your own IDE.

Your program leader/s will indicate which code sharing tool/s to use as a group, and will help break down or provide specific scaffolding with the main concepts above.

▶ **Note on Expectations**

🔎 Problem Solving Approach

To build a long-term organized approach to problem solving, we'll start with three main steps. We'll refer to them as **UPI: Understand, Plan, and Implement**.

We'll apply these three steps to most of the problems we'll see in the first half of the course.

We will learn to:

- **Understand** the problem,
- **Plan** a solution step-by-step, and
- **Implement** the solution

▶ Comment on UPI

▶ UPI Example

Breakout Problems Session 1

▼ Standard Problem Set Version 1

Problem 1: Hundred Acre Wood

Write a function `welcome()` that prints the string `"Welcome to The Hundred Acre Wood!"`.

```
def welcome():
        pass
```

Example Usage:

```
welcome()
```

Example Output:

```
Welcome to The Hundred Acre Wood!
```

▶ 💡 **Hint: Python Functions**

▶ 💡 **Hint: Python Strings**

▶ 💡 **Hint: `print()` function**

Problem 2: Greeting

Write a function `greeting()` that accepts a single parameter, a string `name`, and prints the string `"Welcome to The Hundred Acre Wood <name>! My name is Christopher Robin."`

```
def greeting(name):
        pass
```

Example Usage:

```
greetings("Michael")
greetings("Winnie the Pooh")
```

Example Output:

```
Welcome to The Hundred Acre Wood Michael! My name is Christopher Robin.
Welcome to The Hundred Acre Wood Winnie the Pooh! My name is Christopher Robin.
```

▶ 💡 **Hint: Variables**

▶ 💡 **Hint: Parameters**

▶ 💡 **Hint: Formatted Strings**

Problem 3: Catchphrase

Write a function `print_catchphrase()` that accepts a string `character` as a parameter and prints the catchphrase of the given character as outlined in the table below.

Character	**Catchphrase**
`"Pooh"`	`"Oh bother!"`
`"Tigger"`	`"TTFN: Ta-ta for now!"`
`"Eeyore"`	`"Thanks for noticing me."`
`"Christopher Robin"`	`"Silly old bear."`

If the given `character` does not match one of the characters included above, print `"Sorry! I don't know <character>'s catchphrase!"`

```
def print_catchphrase(character):
	pass
```

Example Usage

```
character = "Pooh"
print_catchphrase(character)

character = "Piglet"
print_catchphrase(character)
```

Example Output:

```
"Oh bother!"
"Sorry! I don't know Piglet's catchphrase!"
```

▶ 💡 **Hint: Conditionals**

Problem 4: Return Item

Implement a function `get_item()` that accepts a 0-indexed list `items` and a non-negative integer `x` and **returns** the element at index `x` in `items`. If `x` is not a valid index of `items`, return `None`.

```
def get_item(items, x):
	pass
```

Example Usage

```
items = ["piglet", "pooh", "roo", "rabbit"]
x = 2
get_item(items, x)

items = ["piglet", "pooh", "roo", "rabbit"]
x = 5
get_item(items, x)
```

Example Output:

```
"roo"
None
```

▶ 💡 **Hint: List Indexing**

▶ 💡 **Hint: Return Values**

Problem 5: Total Honey

Winnie the Pooh wants to know how much honey he has. Write a function `sum_honey()` that accepts a list of integers `hunny_jars` and returns the sum of all elements in the list. Do not use the built-in function `sum()`.

```
def sum_honey(hunny_jars):
        pass
```

Example Usage

```
hunny_jars = [2, 3, 4, 5]
sum_honey(hunny_jars)

hunny_jars = []
sum_honey(hunny_jars)
```

Example Output:

```
14
0
```

▶ 💡 **Hint: For Loops**

▶ 💡 **Hint: Accumulator Variable**

Problem 6: Double Trouble

Help Winnie the Pooh double his honey! Write a function `doubled()` that accepts a list of integers `hunny_jars` as a parameter and multiplies each element in the list by two. Return the doubled list.

```
def doubled(hunny_jars):
        pass
```

Example Usage

```
hunny_jars = [1, 2, 3]
doubled(hunny_jars)
```

Example Output:

```
[2, 4, 6]
```

Problem 7: Poohsticks

Winnie the Pooh and his friends are playing a game called Poohsticks where they drop sticks in a stream and race them. They time how long it takes each player's stick to float under Poohsticks Bridge to score each round.

Write a function `count_less_than()` to help Pooh and his friends determine how many players should move on to the next round of Poohsticks. `count_less_than()` should accept a list of integers `race_times` and an integer `threshold` and return the number of race times less than `threshold`.

```
def count_less_than(race_times, threshold):
        pass
```

Example Usage

```
race_times = [1, 2, 3, 4, 5, 6]
threshold = 4
count_less_than(race_times, threshold)

race_times = []
threshold = 4
count_less_than(race_times, threshold)
```

Example Output:

```
3
0
```

Problem 8: Pooh's To Do's

Write a function print_todo_list() that accepts a list of strings named tasks. The function should then number and print each task on a new line using the format:

`Pooh's To Dos:`
`1. Task 1`
`2. Task 2`
`...`

```python
def print_todo_list(task):
        pass
```

Example Usage

```python
task = ["Count all the bees in the hive", "Chase all the clouds from the sky", "Thin
print_todo_list(task)

task = []
print_todo_list(task)
```

Example Output:

```
Pooh's To Dos:
1. Count all the bees in the hive
2. Chase all the clouds from the sky
3. Think
4. Stoutness Exercises

Pooh's To Dos:
```

▶ 💡 **Hint: `range()` function**

Problem 9: Pairs

Rabbit is very particular about his belongings and wants to own an even number of each thing he owns. Write a function `can_pair()` that accepts a list of integers `item_quantities`. Return `True` if each number in `item_quantities` is even. Return `False` otherwise.

```python
def can_pair(item_quantities):
        pass
```

Example Usage

```
item_quantities = [2, 4, 6, 8]
can_pair(item_quantities)

item_quantities = [1, 2, 3, 4]
can_pair(item_quantities)

item_quantities = []
can_pair(item_quantities)
```

Example Output:

```
True
False
True
```

▶ 💡 **Remainders with Modulus Division**

Problem 10: Split Haycorns

Piglet's has collected a big pile of his favorite food, haycorns, and wants to split them evenly amongst his friends. Write a function `split_haycorns()` to help Piglet determine the number of ways he can split his haycorns into even groups. `split_haycorns()` accepts a positive integer `quantity` as a parameter and returns a list of all divisors of `quantity`.

```
def split_haycorns(quantity):
	pass
```

Example Usage

```
quantity = 6
split_haycorns(quantity)

quantity = 1
split_haycorns(quantity)
```

Example Output:

```
[1, 2, 3, 6]
[1]
```

Problem 11: T-I-Double Guh-ER

Signs in the Hundred Acre Wood have been losing letters as Tigger bounces around stealing any letters he needs to spell out his name. Write a function `tiggerfy()` that accepts a string `s`, and returns a new string with the letters `t`, `i`, `g`, `e`, and `r` from it.

```
def tiggerfy(s):
	pass
```

Example Usage

```
s = "suspicerous"
tiggerfy(s)

s = "Trigger"
tiggerfy(s)

s = "Hunny"
tiggerfy(s)
```

Example Output:

```
"suspcous"
""
"Hunny"
```

▶ 💡 **Hint: String Methods**

Problem 12: Thistle Hunt

Pooh, Piglet, and Roo are looking for thistles to gift their friend Eeyore. Write a function `locate_thistles()` that takes in a list of strings `items` and returns a list of the indices of any elements with value `"thistle"`. The indices in the resulting list should be ordered from least to greatest.

```
def locate_thistles(items):
	pass
```

Example Usage

```
items = ["thistle", "stick", "carrot", "thistle", "eeyore's tail"]
locate_thistles(items)

items = ["book", "bouncy ball", "leaf", "red balloon"]
locate_thistles(items)
```

Example Output:

```
[0, 3]
[]
```

Close Section

▾ Standard Problem Set Version 2

Problem 1: Batman

Write a function `batman()` that prints the string `"I am vengeance. I am the night. I am Batman!"`.

```
def batman():
        pass
```

Example Usage:

```
batman()
```

Example Output:

```
I am vengeance. I am the night. I am Batman!
```

▶ 💡 **Hint: Python Functions**

▶ 💡 **Hint: Python Strings**

▶ 💡 **Hint: `print()` function**

Problem 2: Mad Libs

Write a function `madlib()` that accepts one parameter, a string `verb`. The function should print the sentence: `"I have one power. I never <verb>. - Batman"`.

```
def madlib(verb):
        pass
```

Example Usage

```
verb = "give up"
madlib(verb)

verb = "nap"
madlib(verb)
```

Example Output:

```
"I have one power. I never give up. - Batman"
"I have one power. I never nap. - Batman"
```

▶ 💡 **Hint: Variables**

▶ 💡 **Hint: Parameters**

▶ 💡 **Hint: Formatted Strings**

Problem 3: Trilogy

Write a function `trilogy()` that accepts an integer `year` and prints the title of the Batman trilogy movie released that year as outlined below.

Year	Movie Title
`2005`	`"Batman Begins"`
`2008`	`"The Dark Knight"`
`2012`	`"The Dark Knight Rises"`

If the given `year` does not match one of the years in the table above, print `"Christopher Nolan did not release a Batman movie in <year>."`

```
def trilogy(year):
	pass
```

Example Usage:

```
year = 2008
trilogy(year)

year = 1998
trilogy(year)
```

Example Output:

```
"The Dark Knight"
"Christopher Nolan did not release a Batman movie in 1998."
```

▶ 💡 **Hint: Conditionals**

Problem 4: Last

Implement a function `get_last()` that accepts a list of items `items` and **returns** the last item in the list. If the list is empty, return `None`.

```
def get_last(items):
        pass
```

Example Usage

```
items = ["spider man", "batman", "superman", "iron man", "wonder woman", "black adam
get_last(items)

items = []
get_last(items)
```

Example Output:

```
"black adam"
None
```

▶ 💡 **Hint: List Indexing**

▶ 💡 **Hint: Return Values**

Problem 5: Concatenate

Write a function `concatenate()` that takes in a list of strings `words` and returns a string `concatenated` that concatenates all elements in `words`.

```
def concatenate(words):
        pass
```

Example Usage

```
words = ["vengeance", "darkness", "batman"]
concatenate(words)

words = []
concatenate(words)
```

Example Output:

```
"vengeancedarknessbatman"
""
```

▶ 💡 **Hint: For Loops**

▶ 💡 **Hint: Accumulator Variable**

Problem 6: Squared

Write a function `squared()` that accepts a list of integers `numbers` as a parameter and squares each item in the list. Return the squared list.

```
def squared(numbers):
	pass
```

Example Usage

```
numbers = [1, 2, 3]
squared(numbers)
```

Example Output:

```
[1, 4, 9]
```

Problem 7: NaNaNa Batman!

Write a function `nanana_batman()` that accepts an integer `x` and prints the string `"nanana batman!"` where `"na"` is repeated `x` times. Do not use the `*` operator.

```
def nanana_batman(x):
	pass
```

Example Usage

```
x = 6
nanana_batman(x)

x = 0
nanana_batman(x)
```

Example Output:

```
"nananananana batman!"
"batman!"
```

▶ 💡 **Hint: `range()` function**

Problem 8: Find the Villain

Write a function `find_villain()` that accepts a list `crowd` and a value `villain` as parameters and returns a list of all indices where the `villain` is found hiding in the `crowd`.

```
def find_villain(crowd, villain):
        pass
```

Example Usage

```
crowd = ['Batman', 'The Joker', 'Alfred Pennyworth', 'Robin', 'The Joker', 'Catwoman
villain = 'The Joker'
find_villain(crowd, villain)
```

Example Output:

```
[1, 4, 6]
```

Problem 9: Odd

Write a function `get_odds()` that takes in a list of integers `nums` and returns a new list containing all the odd numbers in `nums`.

```
def get_odds(nums):
        pass
```

Example Usage

```
nums = [1, 2, 3, 4]
get_odds(nums)

nums = [2, 4, 6, 8]
get_odds(nums)
```

Example Output:

```
[1, 3]
[]
```

▶ 💡 **Remainders with Modulus Division**

Problem 10: Up and Down

Write a function `up_and_down()` that accepts a list of integers `lst` as a parameter. The function should return the number of odd numbers minus the number of even numbers in the list.

```
def up_and_down(lst):
        pass
```

Example Usage

```
lst = [1, 2, 3]
up_and_down(lst)

lst = [1, 3, 5]
up_and_down(lst)

lst = [2, 4, 10, 2]
up_and_down(lst)
```

Example Output:

```
1
3
-4
```

Problem 11: Running Sum

Write a function `running_sum()` that accepts a list of integers `superhero_stats` representing the number of crimes Batman has stopped each month in Gotham City. The function should modify the list to return the running sum such that `superhero_stats[i] = sum(superhero_stats[0]...superhero_stats[i])`. You must modify the list in place; you may not create any new lists as part of your solution.

```
def running_sum(superhero_stats):
        pass
```

Example Usage

```
superhero_stats = [1, 2, 3, 4]
running_sum(superhero_stats)

superhero_stats = [1, 1, 1, 1, 1]
running_sum(superhero_stats)

superhero_stats = [3, 1, 2, 10, 1]
running_sum(superhero_stats)
```

Example Output:

```
[1, 3, 6, 10]
[1, 2, 3, 4, 5]
[3, 4, 6, 16, 17]
```

Problem 12: Shuffle

Write a function `shuffle()` that accepts a list `cards` of `2n` elements in the form `[x1,x2,...,xn,y1,y2,...,yn]`. Return the list in the form [x1,y1,x2,y2,...,xn,yn].

```
def shuffle(cards):
        pass
```

Example Usage

```
cards = ['Joker', 'Queen', 2, 3, 'Ace', 7]
shuffle(cards)

cards = [9, 2, 3, 'Joker', 'Joker', 3, 2, 9]
shuffle(cards)

cards = [10, 10, 2, 2]
shuffle(cards)
```

Example Output:

```
['Joker', 3, 'Queen', 'Ace', 2, 7]
[9, 'Joker', 2, 3, 3, 2, 'Joker', 9]
[10, 2, 10, 2]
```

Close Section

▸ Advanced Problem Set Version 1

▸ Advanced Problem Set Version 2