Recursion

Welcome back to CS 2100!

Prof. Rasika Bhalerao

Recursion

Recursion: Solving a problem by solving a smaller version of the problem

(The function calls itself to do the smaller version of itself)

Let's trace this function:

```
def power(base: int, exp: int) -> int:
    """Returns the base raised to the power of the exponent"""
    if exp == 0:
        return 1
    else:
        returned = power(base, exp-1)
        return base * returned
        # base ^ exp = base * base ^ (exp-1)
```

Notes from tracing the recursive function:

- In the debugger, we saw:
  - Call stack: the stack of function calls that are currently running
  - Stack frame:
    - Each function call in the call stack is a stack frame
    - Each stack frame includes its own copy of the variables / parameters

Poll: What does this function do?

```
def something(inp: int) -> int:
    if inp == 0:
        return 0
    else:
        digit = inp % 10
        rest = inp // 10
        return digit + something(rest)
```

1. Returns a number added to itself 10 times
2. Returns the sum of the digits in a number
3. Returns a number added to itself as many times as digits in the number
4. Returns 0

Structure of a recursive function:

- Base case(s)
  - The simplest version of the problem you are trying to solve
  - E.g., 0, empty string, empty list
- Recursive case(s)
  - Do one step
    - E.g., process one number, print one character
  - Recursively call the function to do the rest of the steps

What if no base case?

Let's remove the base case from `something` and trace it!

Implementing a recursive function

Let's write a recursive function that returns the sum of the numbers in a list.

Twist: sometimes an element in the list is another list (use `Any` to satiate MyPy)

- Use `isinstance()` to differentiate the case
- Two base cases
  - Empty list (return 0)
  - `None` / not a list (return 0)
- Two recursive cases:
  - First element is a number: add the first element to the sum of the rest of the list
  - First element is a list: sum the elements in that first list and add that to the sum of the rest of the list

```
def sum_list(inp: list[Any]) -> Any:
    if inp is None or not isinstance(inp, list):
        return 0
    elif len(inp) == 0:
        return 0
    else:
        element = inp[0]
        if isinstance(element, list):
            return sum_list(element) + sum_list(inp[1:])
        else:
            return element + sum_list(inp[1:])

print(sum_list([1, 2, [3, 4], 5]))
```

Poll: What goes in the ??? ? We want the `size()` function to return the size of a string.

```
def size(inp: str) -> int:
    if inp == '':
        return 0
    else:
        return ???
```

1. `1 + size(inp)`
2. `size(inp)`
3. `0 + size(inp)`
4. `0`

Poll: What goes in the ??? ? We want the `reverse()` function to return the reversed version of the string.

```
def reverse(inp: str) -> str:
    if inp == '':
        return ???
    else:
        return ???
```

1. `''`, `inp[0] + reverse([1:])`
2. `''`, `reverse(inp[1:]) + inp[0]`
3. `None`, `inp[0] + reverse([1:])`
4. `None`, `reverse(inp[1:]) + inp[0]`

Poll: What goes in the ??? ?

```
def reverse_inputs(n: int) -> None:
    """Takes n inputs from the user, and prints them in reverse order.
    Example: user types in "cat" "elephant" "fish"
    Output:
       fish
       elephant
       cat"""
    if n > 0:
        ???
```

1.

```
word = input('Please enter a word: ')
print(word)
reverse_inputs(n - 1)
```

2.

```
reverse_inputs(n - 1)
word = input('Please enter a word: ')
print(word)
```

3.

```
word = input('Please enter a word: ')
reverse_inputs(n - 1)
print(word)
```

Exercise: Let's write a recursive function `try_all()` that calls this function using all possible passcodes:

```
def unlock(passcode: str) -> None:
    """Unlocks the vault if the passcode is correct"""
    if passcode == '0004':
        print("Unlocked!")
```

```
def try_all(num_digits: int, passcode_so_far: str = '') -> None:
    if len(passcode_so_far) == num_digits:
        unlock(passcode_so_far)
    else:
        for digit in range(10):
            try_all(num_digits, f'{passcode_so_far}{digit}')
```

In-class exercise: debate

Split the class into three groups, one for each of the **three problems** (on the next slide).

And within each of those three groups, split the group further into three sub-groups.

For each of the three problems:

- One sub-group will argue that we should use a **for loop** to solve the problem
- One sub-group will argue that we should use a **while loop** to solve the problem
- One sub-group will argue that we should use **recursion** to solve the problem

You will have 5 minutes to prepare, and then send two representatives from your group to argue your assigned position. They must argue their assigned position.
You may use any tool to come up with an argument.

Problem 1: is_palindrome()

The problem: write a function called `is_palindrome()` that takes a string and returns `True` if it is a palindrome (and returns `False` otherwise).
A string is a palindrome if it is the same backwards and forwards.

Problem 2: greatest_common_denominator()

The problem: write a function called `greatest_common_denominator()` that takes two `int`s and returns the largest integer that is a factor of both `int`s.

Problem 3: flatten()

The problem: write a function called flatten() which takes a list (where the elements may be lists, or they may be individual things) and returns a "flattened" version of the list (where the elements are not lists, and they must just be the individual things).

Example: `flatten([4, 5, [1, 2, [3]], [0, 6]]) = [4, 5, 1, 2, 3, 0, 6]`

Poll:

1. What is your main takeaway from today?

2. What would you like to revisit next time?