Recursive Functions

Sometimes when solving a problem, we can compute the solution of a simpler version of the same problem. Eventually we reach the most basic version, for which the answer is known.

Manager: Recorder:

Presenter: Reflector:

Content Learning Objectives

After completing this activity, students should be able to:

- Identify the base case and recursive step of the factorial function.
- Trace a recursive function by hand to predict the number of calls.
- Write short recursive functions based on mathematical sequences.

Process Skill Goals

During the activity, students should make progress toward:

- Evaluating mathematical functions to gain insight on recursion. (Information Processing)

Model 1 Factorial Function

"In mathematics, the factorial of a non-negative integer n, denoted by $n!$, is the product of all positive integers less than or equal to n. For example, $5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$."

Source: https://en.wikipedia.org/wiki/Factorial

n	$n!$
0	1
1	1
2	2
3	6
4	24
5	120

Questions (15 min) **Start time:**

1. Consider how to calculate $4! = 24$.

a) Write out all the numbers that need to be multiplied:

$4! =$

b) Rewrite the expression using $3!$ instead of $3 \times 2 \times 1$:

$4! =$

2. Write expressions similar to #1b showing how each factorial can be calculated in terms of a smaller factorial. Each answer should end with a factorial (!).

a) $2! =$

b) $3! =$

c) $100! =$

d) $n! =$

3. What is the value of $0!$ based on Model 1? Does it make sense to define $0!$ in terms of a simpler factorial? Why or why not?

If we repeatedly break down a problem into smaller versions of itself, we eventually reach a basic problem that can't be broken down any further. Such a problem, like 0!, is referred to as the **base case**.

4. Consider the following Python function that takes n as a parameter and returns $n!$:

```
1 def factorial(n):
2     # base case
3     if n == 0:
4         return 1
5     # general case
6     product = 1
7     for i in range(n, 0, -1):
8         product *= i
9     return product
```

a) Review your answer to #2c that shows how to compute 100! using a smaller factorial. Convert this expression to Python by using the function above instead of the ! operator.

b) Now rewrite your answer to #2d in Python using the variable `n` and the function above.

c) In the source code above, replace the "1" on Line 6 with your answer from b). Then cross out Lines 7 and 8. Test the resulting function in a Python Shell. Does it still work?

d) What specific function is being called on Line 6?

e) Why is the `if` statement required on Line 3?

5. A function that refers to itself is called **recursive**. What two steps were necessary to define the recursive version of `factorial`?

6. Was a loop necessary to cause the recursive version of `factorial` to run multiple times? Explain your reasoning.

Model 2 Fibonacci Numbers

The Fibonacci numbers are a sequence where every number (after the first two) is the sum of the two preceding numbers: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, . . .

Source: https://en.wikipedia.org/wiki/Fibonacci_number

We can define a recursive function to compute Fibonacci numbers. Enter the following code into a Python Editor, and run the program to see the sequence.

```
1  def fibonacci(n):
2      # base case
3      if n == 1 or n == 2:
4          return 1
5      # general case
6      return fibonacci(n - 1) + fibonacci(n - 2)
7
8  if __name__ == "__main__":
9      for i in range(1, 6):
10         print(fibonacci(i))
```

Questions (10 min) Start time:

7. Based on the source code:

a) How many function calls are needed to compute `fibonacci(3)`? Identify the value of the parameter `n` for each of these calls.

b) How many function calls are needed to compute `fibonacci(4)`? Identify the value of the parameter `n` for each of these calls.

c) How many function calls are needed to compute `fibonacci(5)`? Identify the value of the parameter `n` for each of these calls.

8. Check your answers for the previous question by adding the following `print` statements to the code and rerunning the program:

- Insert `print("n is", n)` at Line 2, before the `# base case` comment
- Insert `print("fib(%d) is..." % i)` at Line 10, before the `print` statement

9. What happens if you try to compute `fibonacci(0)` in the Python Shell?

10. How could you modify the code so that this situation doesn't happen?

Model 3 Summation

"In mathematics, summation (capital Greek sigma symbol: Σ) is the addition of a sequence of numbers; the result is their sum or total."

$$\sum_{i=1}^{100} i = 1 + 2 + 3 + \ldots + 100 = 5050$$

Source: https://en.wikipedia.org/wiki/Summation

Questions (20 min) Start time:

11. Consider how to calculate $\sum_{i=1}^{4} i = 10$.

a) Write out all the numbers that need to be added:

$\sum_{i=1}^{4} i =$

b) Show how this sum can be calculated in terms of a smaller summation.

$\sum_{i=1}^{4} i =$

12. Write an expression similar to #11b showing how any summation of n integers can be calculated in terms of a smaller summation.

$$\sum_{i=1}^{n} i =$$

13. What is the base case of the summation? (Write the complete formula, not just the value.)

Here are important questions to consider before writing a recursive function:

- *How can you define the problem in terms of a smaller similar problem?*
- *What is the base case, where you solve an easy problem in one step?*
- *For the recursive call, how will you make the problem size smaller?*

To avoid infinite recursion, make sure that each recursive call brings you closer to the base case!

14. Implement a recursive function named `summation` that takes a parameter `n` and returns the sum $1 + 2 + \ldots + n$. It should only have an `if` statement and two `return` statements (no loops).

15. Enter your code into a Python Editor, and test the function. Make sure that `summation(100)` correctly returns 5050.

16. Implement a recursive function named `geometric` that takes three parameters (`a`, `r`, and `n`) and returns the sum "$a + ar + ar^2 + ar^3 \ldots$" where $n + 1$ is the total number of terms.

a) What is the base case?
`geometric(a, r, 0)` returns:

b) What is the recursive case?
`geometric(a, r, n)` returns:

c) Write the function in Python:

17. Enter your code into a Python Editor, and test the function. For example, if $a = 10$ and $r = 3$, the first five terms would be 10, 30, 90, 270, and 810. Make sure that `geometric(10, 3, 4)` correctly returns 1210 (the sum of those five terms).