Recursion

Lab 5 - March 12, 2024

Lab #4 Quiz Task)

Print out the first N integers, one per line, where N is read in from the user. When printing the numbers, apply the following rules:

1. For multiples of 3, print "Fizz" instead of the number.
2. For multiples of 5, print "Buzz" instead of the number.
3. For numbers that are multiples of both 3 and 5, print "FizzBuzz".

- If the user enters a number less than 1, print out “N must be greater than 1”
- If the user enters a number greater than 100, print out “Too much work, no thanks”

```
def print_nums():
    """
    For multiples of:
        - 3, print "Fizz" instead of the number.
        - 5, print "Buzz" instead of the number.
        - both 3 and 5, print "FizzBuzz".

    Otherwise, print the number.

    Other rules:
       - If N <= 1, print "N must be greater than 1"
       - If N > 100, print "Too much work, no thanks"

    """
```

Example: N = 15

```
1
2
Fizz
4
Buzz
Fizz
7
8
Fizz
Buzz
11
Fizz
13
14
FizzBuzz
```

print_nums() body:

```
"""
N = int(input("Enter an integer: "))

if N <= 1:
    print("N must be greater than one")
    return None

elif N > 100:
    print("Too much work, no thanks")
    return None

else:
    for num in range(1, N+1):
        if num % 3 == 0 and num % 5 == 0:
            print("FizzBuzz")
        elif num % 3 == 0:
            print("Fizz")
        elif num % 5 == 0:
            print("Buzz")
        else:
            print(num)

return None
```

Example: N = 15

```
1
2
Fizz
4
Buzz
Fizz
7
8
Fizz
Buzz
11
Fizz
13
14
FizzBuzz
```

Test:

```
In [46]: print_nums()
Enter an integer: 15
1
2
Fizz
4
Buzz
Fizz
7
8
Fizz
Buzz
11
Fizz
13
14
FizzBuzz
```

In Python, we know that a function can call *other* functions...

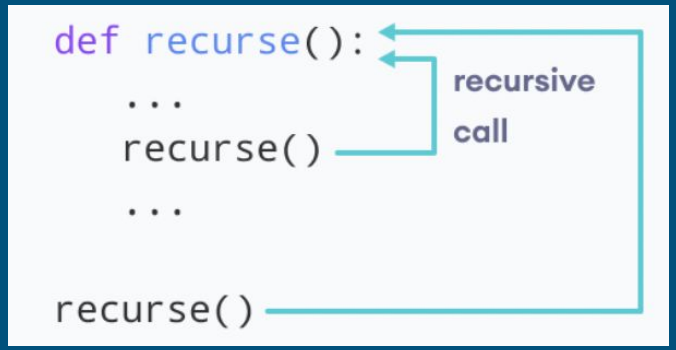
```
## ---------------------------- ##
#  Helper Function: get_squared  #
## ---------------------------- ##
def get_squared(x):
    """ Returns the number squared.

    Examples:
    ---------
    >>> get_squared(1)
    1
    >>> get_squared(2)
    4
    >>> get_squared(3)
    9

    """
    return x * x
```

```
## ---------------------------------- ##
#   Calling HF in another function body  #
## ----------------------------------##
def get_sum_of_squares(my_list: list):
    """
    Returns the sum of squared numbers
    in my_list.

    Examples:
    ---------
    >>> get_sum_of_squares([1, 2, 3])
    14

    """
    sum = 0
    for num in my_list:
        sum += get_squared(num)
    return sum
```

```
In [11]: get_sum_of_squares([1, 2, 3])
Out[11]: 14
```

Recursion: A function can also call *itself*

- A function that calls itself is a **recursive function.**
- Recursion is a common mathematical concept!



- We also use recursion as a **programming technique**
  - Allows us to ***loop* through our function** to reach a result.

A Common Example of Mathematical Recursion: The Factorial (!) Function

- The **factorial function** (symbol: **!**) says to **multiply all whole numbers** from a specified number down to 1:
  - ! = “The factorial of”
- Examples:

  4! = 4 * 3 * 2 * 1 = 24

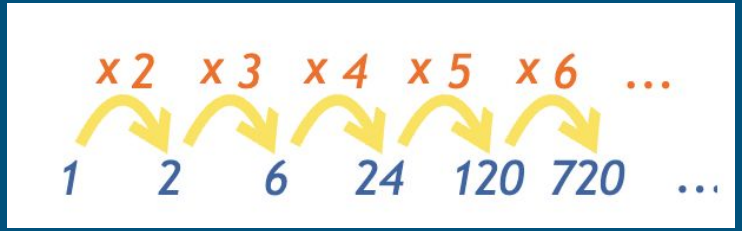
  6! = 6 * 5 * 4 * 3 * 2 * 1 =720

  1! = 1
- Recursive Definition: **n! = n * (n-1)!**

Factorials (!)

Notice how we can calculate a factorial of any given number based the factorial of the previous number:

n	n!		
1	**1**	1	1
2	2 × **1**	= 2 × **1!**	= 2
3	3 × **2 × 1**	= 3 × **2!**	= 6
4	4 × **3 × 2 × 1**	= 4 × **3!**	= 24
5	5 × **4 × 3 × 2 × 1**	= 5 × **4!**	= 120
6	etc	etc	



To get:

2!, we multiply 2 by 1 to get 2.

3!, we multiply 3 by 2 to get 6

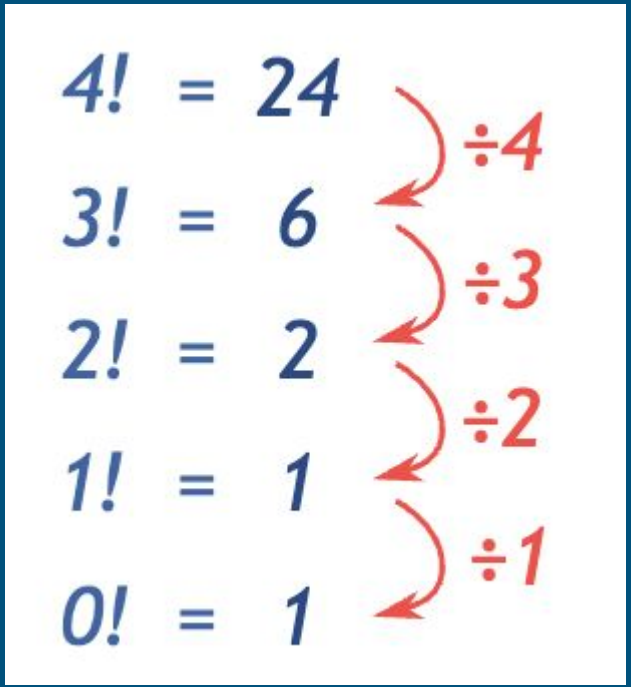
4!, we multiply 4 by 6 to get 24

5!, we multiply 5 by 24 to get 120

6!, we multiply 6 by 120 to get 720

etc.

What about 0!



Mathematical convention: 0! = 1.

Make it make sense:

- If we follow the recursive pattern backwards, it makes sense that we define 0! as 1.

Recursive definition of factorial

Rule: “The **factorial** of a number will always be that number times the **factorial of that number minus one**”

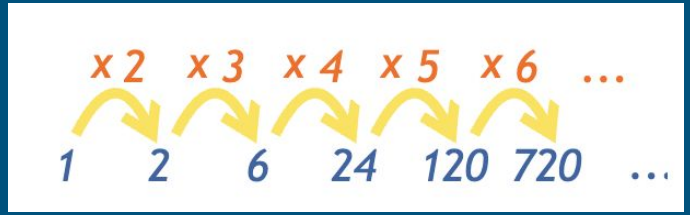
$$1! = 1$$
$$2! = 2 \times 1 = 2$$
$$3! = 3 \times 2 \times 1 = 6$$
$$4! = 4 \times 3 \times 2 \times 1 = 24$$
$$5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$$

Iterative definition: **n! = n x (n-1) x (n-2) x … x 1**

In recursive terms:: **n! = n x (n-1)!**

Iterative Implementation of factorials

```python
def factorial_iterative(n):
    """
    This is an iterative function
    to find the factorial of an integer.
    """
    result = 1
    while n > 1:
        result = result * n
        n -= 1
    return result
```




```python
print("0! = ", factorial_iterative(0))
print("1! = ", factorial_iterative(1))
print("2! = ", factorial_iterative(2))
print("3! = ", factorial_iterative(3))
```

→

```
In [36]:
0! =  1
1! =  1
2! =  2
3! =  6
```

Recursive Implementation

```
def factorial(n):
    """
    This is a recursive function
    to find the factorial of an integer.
    """
    if n == 1:
        # BASE CASE
        return n
    else:
        # RECURSIVE CASE
        return n * factorial(n - 1)
```

This is a recursive function because it calls itself (i.e., in its function body)

When we call this function on a positive integer, it will *recursively* call itself by decreasing the number until the number reaches 1.

```
num = 3
print("The factorial of", num, "is", factorial(num))
```

```
The factorial of 3 is 6
```

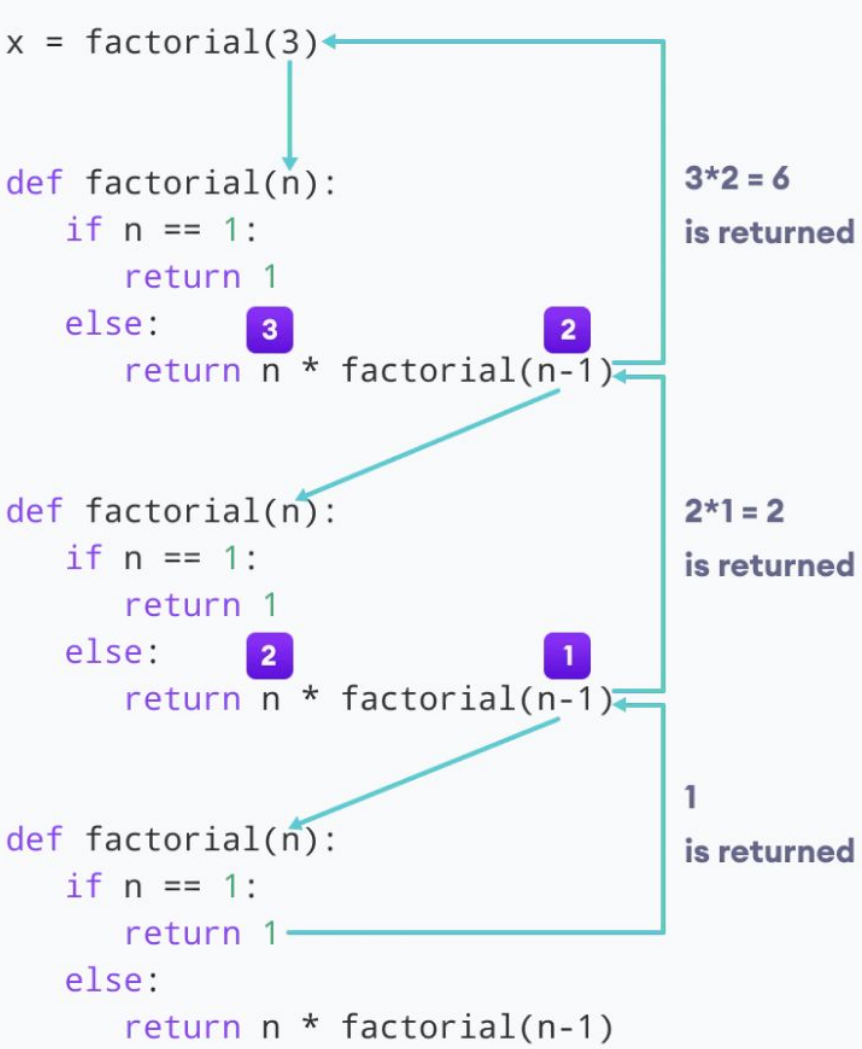
Each function multiples the number with the factorial of the number below it until it is equal to one (i.e., BASE CASE).

This recursive call can be explained in the following steps:

```
factorial(3)          # 1st call with 3
3 * factorial(2)      # 2nd call with 2
3 * 2 * factorial(1)  # 3rd call with 1
3 * 2 * 1             # return from 3rd call as number=1
3 * 2                 # return from 2nd call
6                     # return from 1st call
```

Step-by-step process: factorial(3)

Out recursion ends when the number reduces to 1. This is called the **base condition**.



The Base condition

- Every recursive function must have a **base condition** that stops the recursion OR ELSE the function will call itself *infinitely*.
- By default, the maximum depth of recursion is 1000 (calling the function 1000 times). If this limit is crossed, it results in **RecursionError**.

```
def recursor():
    recursor()
recursor()
```

Output:

```
Traceback (most recent call last):
  File "<string>", line 3, in <module>
  File "<string>", line 2, in a
  File "<string>", line 2, in a
  File "<string>", line 2, in a
  [Previous line repeated 996 more times]
RecursionError: maximum recursion depth exceeded
```