Recursion

CSCI 1030U - Intro to Computer Science
@IntroCS

Randy J. Fortier
@randy_fortier

Outline

- Recursion
  - Recursive calls and backtracking
  - Recursive function calls and the calling stack
  - Tail recursion and optimization

Recursion

Recursion - A Motivating Example

- Say you are in line and want to know what position you have in that line
    - You could count all of the people yourself, but you are lazy and don't want to do that
    - Any ideas?

Recursion - A Motivating Example

- You could ask the person in front of you the same question
  - "What position in line are you?"

Recursion - A Motivating Example

- You could ask the person in front of you the same question
  - "What position in line are you?"
  - Assuming that they are also too lazy to count, they may ask the same question of the person in front of them

Recursion - A Motivating Example

- … and so on

Recursion - A Motivating Example

- The person at the front of the line knows that they are first, so they pass that information back to the person behind them

Recursion - A Motivating Example

- The second person adds one (for themselves) and passes back the count

Recursion - A Motivating Example

- … and so on
- Finally, you know that there are 6 people in front of you, so you must be seventh in line

Recursion

- A recursive function is one that is defined in terms of itself
  - i.e. The function calls itself, directly or indirectly
- Example:

Python:

```python
def forever():
    print("hello")
    forever()
```

C++:

```cpp
void forever() {
    cout << "hello";
    forever();
}
```

OntarioTech UNIVERSITY

Recursion

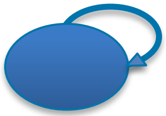
- The example below is called *direct recursion*, since the function `forever` calls itself directly

Python:

```
def forever():
    print("hello")
    forever()
```

C++:

```
void forever() {
    cout << "hello" << endl;
    forever();
}
```

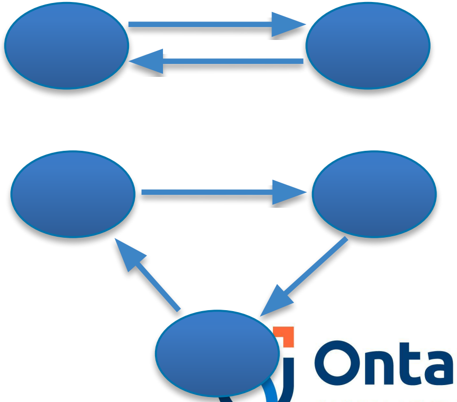


Recursion

- The example below is called *indirect recursion*, since the function `forever1` calls itself indirectly
  - This example has two functions calling each other in a cycle, but any number of functions can be involved

```
def forever1():
    print("hello")
    forever2()

def forever2():
    forever1()
```



OntarioTech UNIVERSITY

Recursion - Exit Conditions

- The most important thing to remember with recursion is to include an *exit condition*
  - An exit condition is a way to stop repetition
  - Similar to how a loop must always exit, so must recursion
- The example below has no exit condition
  - The result is *infinite recursion*

```
def forever(message):
    print(message)
    forever(message)
```

pythontutor.com/visualize.html

Infinite Recursion - Video Example

```
def infinite(x):
    y = 3
    print(x + y)
    infinite(x + 1)

infinite(1)
```

Recursion - Exit Conditions

- Let's define a function which has an exit condition:

```
def repeat_n_times(n, message):
    if n < 1:
        return
    print(message)
    repeat_n_times(n - 1, message)

repeat_n_times(10, "hello")
```

pythontutor.com/visualize.html

Recursion - Video Example

```python
def sum_list(elems):
    if len(elems) == 0:
        return 0
    sum_rest = sum_list(elems[1:])
    return sum_rest + elems[0]

total = sum_list([1,2,3])
```

0

Coding Exercise 05b.1

- Write a recursive implementation to calculate the Fibonacci numbers
- Recall the definition of the Fibonacci numbers is:

$$\text{Fib } n = \begin{cases} n, & \text{if } n == 0 \text{ or } n == 1 \\ \text{Fib}(n-1) + \text{Fib}(n-2), & \text{if } n > 1 \end{cases}$$

0, 1, 1, 2, 3, 5, 8, 13, 21, …

Coding Challenge 05b.1

- Create a recursive version of the factorial function
- Don't forget an exit condition!

$$n! = \begin{cases} 1, \text{ if } n == 0 \text{ or } n == 1 \\ n * (n - 1)!, \text{ if } n > 1 \end{cases}$$

OntarioTech UNIVERSITY

Coding Exercise 05b.2

- Write your own *recursive* version of the `filter` function, called `myfilter`, which:
  - Takes a unary function (`check`) and a list (`values`) as arguments
  - Applies the function `check` to successive each value from the list, and if the result is `True`, adds the value to the output list
- For example:

```
marks = [64.5, 87.0, 55.5, 94.0, 71.5, 46.0, 100.0]
a_grades = myfilter(lambda mark: mark >= 80.0, marks)
# a_grades should be [87.0, 94.0, 100.0]
```

Tail Recursion

Tail Recursion

- Tail recursion is a special case of recursion where the recursive call is the last thing to happen before the function returns
  - It is noteworthy since this kind of recursion can be easily optimized:
    1. Converting to an iterative equivalent
    2. Simplifying the calling stack

```
def print_n_times(n, message):
    if n == 0:
        return
    print(message)
    print_n_times(n - 1, message)

print_n_times(5, ´Hello´)
```

Tail Recursion - Discussion

- Is the following Fibonacci function tail-recursive?

```
def fibonacci(n):
    if n <= 1:
        return n
    return fibonacci(n-1) + fibonacci(n-2)
```

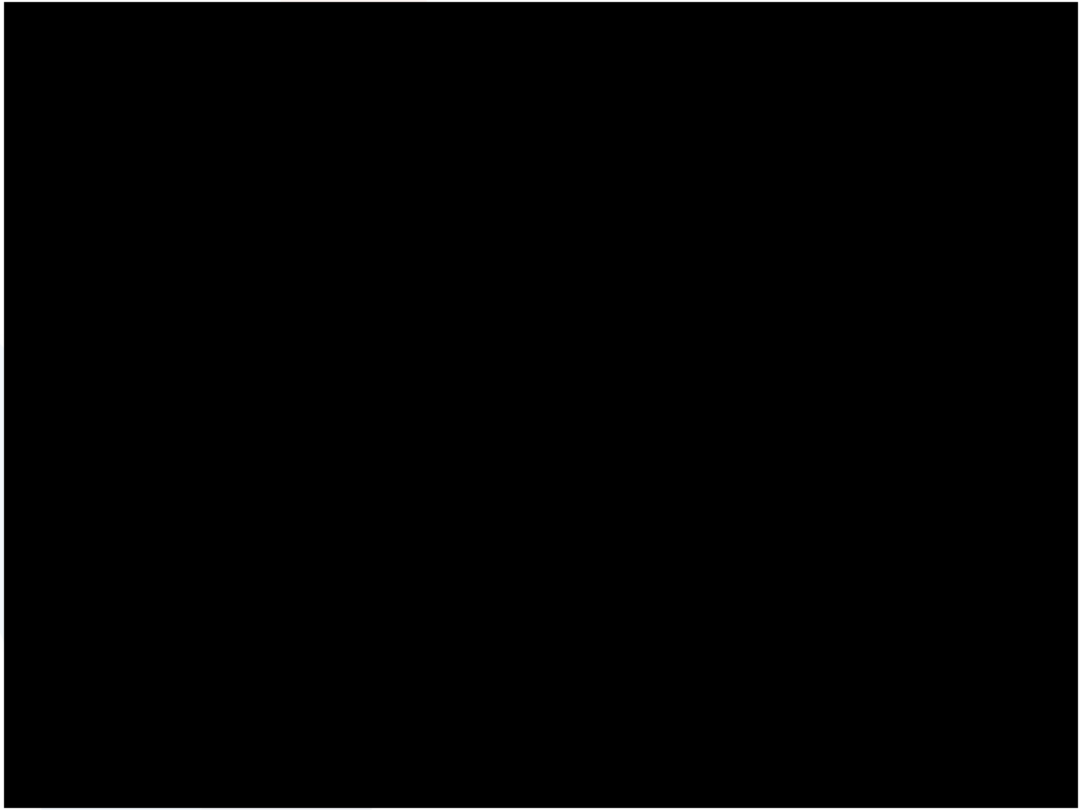
pythontutor.com/visualize.html

Tail Recursion - Calling Stack

- Each stack frame on the calling stack remembers the *return address*, which is the address of the instruction immediately following the function call
  - Since the very next action after the recursive call is a return, we can simplify the stack significantly
  - Instead of each recursive call waiting for other recursive calls to exit, only to return, we can just have the innermost recursive call return to the original return address

Ontario Tech UNIVERSITY

Tail Recursion - Video Example



Tail Recursion - Video Example

0

```
def sum_list_tail(elems, base = 0):
    if len(elems) == 0:
        return base
    return sum_list_tail(elems[1:], base + elems[0])

total = sum_list_tail([1,2,3])
```

Hacker's Corner: Iterative Conversion

- Converting a tail-recursive function to iteration is usually straightforward:

```
def print_n_times(n, message):
    if n == 0:
        return
    print(message)
    print_n_times(n - 1, message)
```

```
def print_n_times(n, message):
    for i in range(n):
        print(message)
```

Wrap-up

- Recursion
    - Recursive calls and backtracking
    - Recursive function calls and the calling stack
    - Tail recursion and optimization

OntarioTech UNIVERSITY

Coming Up

- Motivation
  - What are classes?
  - What are objects?
  - Objects and classes in real life
  - Member variables and functions
- Writing object-oriented programs
  - Python syntax