Recursive Functions

HIT137 Software Now

Week 05

We extend our respect to Elders - past, present and emerging - and to all First Nations people.

Outline

- Understanding of:
  - Explain why functions are useful in structuring code in a program
  - **Define a recursive function**
  - Explain the use of the namespace in a program and exploit it effectively
  - Define a function with required and optional parameters

Functions as Abstraction Mechanisms

- An abstraction hides detail
  - **Allows a person to view many things as just one thing**
- We use abstractions to refer to the most common tasks in everyday life
  - For example, the expression “doing my laundry”
- Effective designers must invent useful abstractions to control complexity

Functions Eliminate Redundancy

- Functions serve as abstraction mechanisms by eliminating **redundant**, or **repetitious** code

```
def sum(lower, upper):
  """
  Arguments: A lower bound and an upper bound
  Returns: the sum of the numbers between the arguments
    and including them
  """
  result = 0
  while lower <= upper:
    result += lower
    lower += 1
  return result

print(sum(1, 4)) # The summation of the numbers 1..4
```

Function calls

- A function call is an expression that invokes a named function, passing argument(s).
- E.g.:

```
abs(10 - n)
max(n1, n2)
input('Enter a number: ')
```

- The function call contains sub-expression(s) enclosed in parentheses. These sub-expressions are evaluated to determine the argument(s), which are passed to the named function.

Function Calls - Built-in Functions

- Python provides a variety of “built-in” functions. You have tried in previous weeks.
- Arithmetic functions:
  - abs(x) returns the absolute value of x
  - max(x, y) returns the maximum of x and y
  - min(x, y) returns the minimum of x and y
  - round(x) returns the value of x rounded to the nearest integer

Design with Recursive Functions

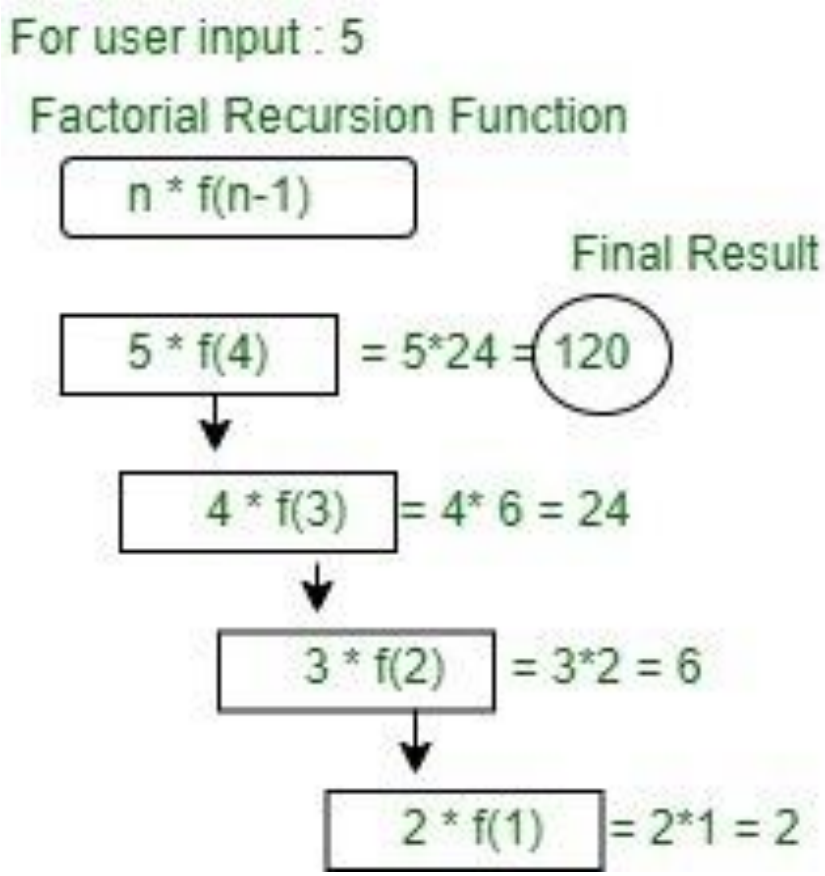
- In top-down design, you decompose a complex problem into a set of simpler problems and solve these with different functions
- In some cases, you can decompose a complex problem into smaller problems of the same form
  - Subproblems can be solved using the same function
  - This design strategy is called **recursive design**
  - Resulting functions are called **recursive functions**

Defining a Recursive Function

- A recursive function is a function that calls itself ☺
- To prevent function from repeating itself indefinitely, it must contain at least one selection statement
- Statement examines **base case** to determine whether to stop or to continue with another **recursive case**

```
def factorial(n):
    if n == 1:
        return 1  # Base Case
    else:
        return n * factorial(n - 1)  # Recursive Case
```

How a Recursive Function Works?



Computation Time: Recursive and Iterative

Guess - Which one is better?

```python
import time

def factorial_recursive(n):
    if n == 0:
        return 1
    else:
        return n * factorial_recursive(n-1)

def factorial_iterative(n):
    result = 1
    for i in range(2, n+1):
        result *= i
    return result

n = 1000
```

```python
# Measure time for recursive approach
start_time = time.time()
factorial_recursive(n)
end_time = time.time()
recursive_time = end_time - start_time

# Measure time for iterative approach
start_time = time.time()
factorial_iterative(n)
end_time = time.time()
iterative_time = end_time - start_time

print(f"Recursive approach took {recursive_time:.10f} seconds")
print(f"Iterative approach took {iterative_time:.10f} seconds")
```

```
Recursive approach took 0.0011270046 seconds
Iterative approach took 0.0008859634 seconds
```

Defining a Recursive Function

- To convert displayRange to a recursive function:

- You can replace loop with a selection statement and assignment statement with a **recursive call**

Tutorial Session

Resources

Some great resources to start your work :

- Fundamentals of Python: First Programs
- https://www.w3schools.com/python/default.asp
- https://python.swaroopch.com/