Python
WEEK 3
Recursion

AI Academy

COMPUTER PROGRAMMING WITH PYTHON

Instructor - James E. Robinson, III

Teaching Assistant - Travis Martin

LIGHTNING REVIEW

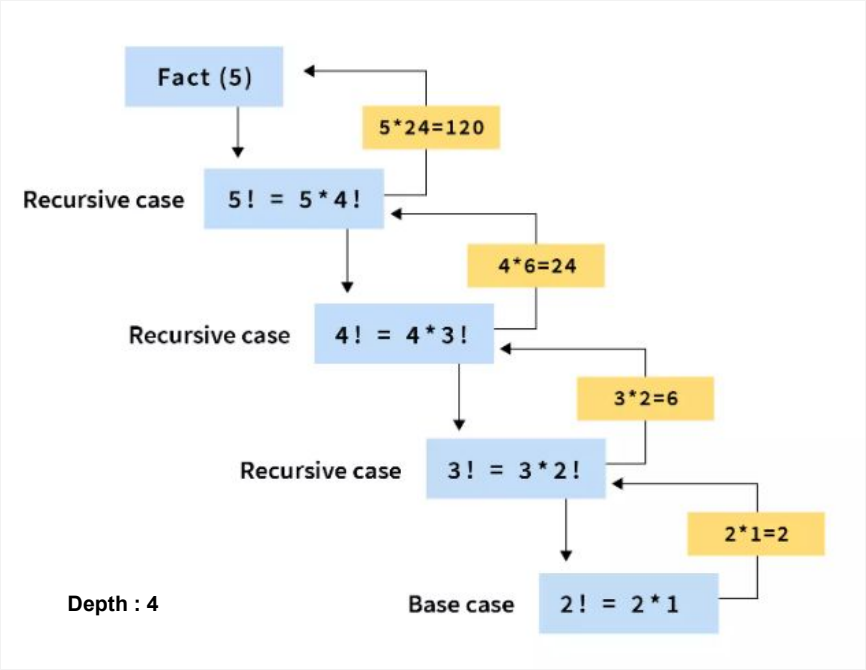
- Variables
- Input / Output
- Expressions
- Functions
- Conditional Control
- Looping
- Data Types
- Logging
- Functions
- Scope
- Decorators

TOPICS COVERED

- Recursion
  - Concept
  - Structure
  - Recursion v/s Iteration
- Dynamic Programming
- Exception Handling
  - Structure
  - Clauses

RECURSION - CONCEPT

SOLVING A PROBLEM BY SOLVING SMALLER AND SIMPLER SUB-PROBLEMS OF THE ORIGINAL



Base Case
It defines when the function should return instead of calling itself again.

Depth
The number of times a function calls itself

Note :

Recursion can only be performed if the problem can be divided into *smaller & similar* sub-problems.

Recursion problems require a base case to control the depth of recursion (stop it from going infinitely)

RECURSION - STRUCTURE

EVERY RECURSIVE PROBLEM HAS A BASIC SIMPLE STRUCTURE

Pseudocode

```
recursive(param):
    if base_case:
        return (val)
    else recursive_case:
        #actions
        recursive(param_mod)
```

Note :

base_case must be a definitive condition.

recursive_case performs any required #actions before or after recursive call.

param_mod are modified parameters for the next recursive call.

param_mod must eventually get value where base_case becomes true (or else recursion depth = infinity)

RECURSION - STRUCTURE

EVERY RECURSIVE PROBLEM HAS A BASIC SIMPLE STRUCTURE

Example

```
def printall(x):
    if x <= 0:
        return
    else:
        (1)print(x)
        printall(x-1)
        (2)print(x)

printall(10)
```

OUTPUT(1) :

10
9
8
7
6
5
4
3
2
1

OUTPUT(2) :

1
2
3
4
5
6
7
8
9
10

Note:

Recursive calls are expensive as they take up a lot of memory and time since it uses a call stack

Recursive functions are hard to debug since you won't know at what juncture the recursive call breaks

RECURSION - CLASSIC EXAMPLE

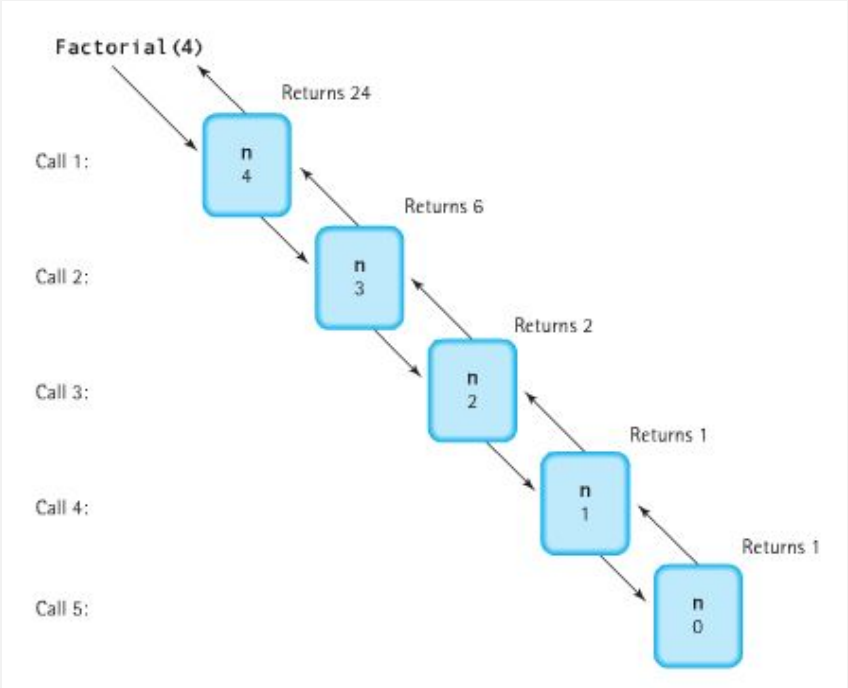
CALCULATING FACTORIAL OF A GIVEN NUMBER

Example

```
def Factorial(n):
    if n == 0:
        return 1
    else:
        return n * Factorial(n-1)
print("4! = ", Factorial(4))
```

OUTPUT :

4! = 24



AI Academy

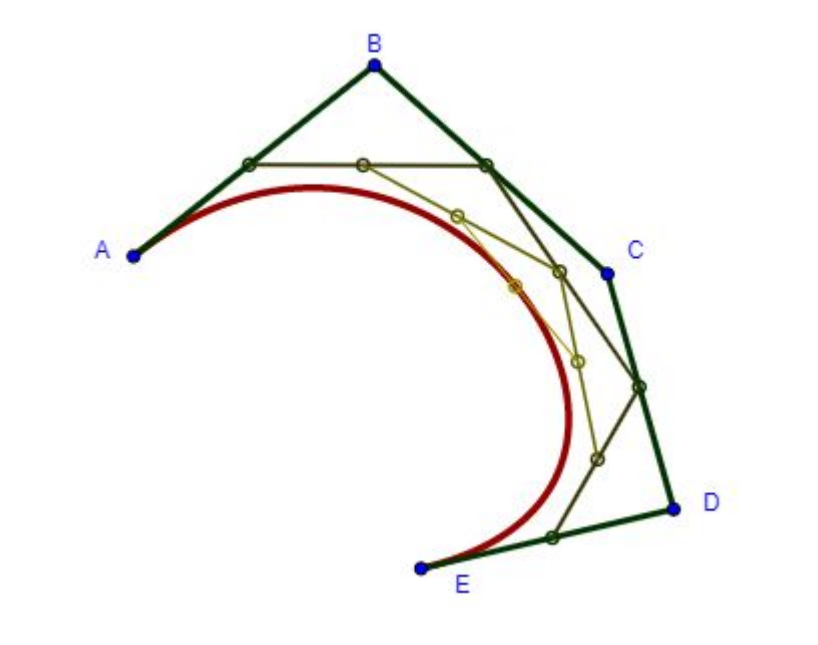
NC STATE

RECURSION - MODERN EXAMPLE

EVALUATING A BEZIER CURVE OF NTH ORDER

De Casteljau's algorithm

It is a recursive method to evaluate polynomials in Bernstein form or Bézier curves



RECURSION V/S ITERATION

RECURSION AND ITERATION WHILE SIMILAR AREN'T EXACTLY SAME

"All Recursive problems can be solved using Iteration, but not all Iterative problems can be solved by Recursion"

Example (Add 1 to N numbers)

Iteration:

```
n = 10
s = 0
for i in range(n+1):
    s += i
print(s)
```

Recursion:

```
n = 10
s = 0
def add(s, n):
    if n <= 0:
        return s
    else:
        s += n
        return add(s, n-1)
print(add(s, n))
```

Note:
Example shows how loops can be converted to recursive calls

RECURSION V/S ITERATION

RECURSION AND ITERATION WHILE SIMILAR AREN'T EXACTLY SAME

There are several factors where Recursion and Iteration vary. Hence choosing the right one is helpful in making the code efficient.

Usage

Usage of either of these techniques is a trade-off between time complexity and size of code.

While recursion decreases size of code and increases simplicity, it has a higher time complexity that iteration.

Infinite Repetition

Infinite Repetition in recursion will lead to program halting when recursion limit is exceeded; however, in iteration it will only stop when memory is exhausted.

Overhead

Recursion has a large amount of Overhead as compared to Iteration.

DYNAMIC PROGRAMMING

DYNAMIC PROGRAMMING IS AN OPTIMIZATION OF PLAIN RECURSION

Example

```
def fibonacci(n):
    f = [0, 1]
    for i in range(2, n+1):
        f.append(f[i-1] + f[i-2])
    return f[-1]
print(fibonacci(6))
```

OUTPUT :

[0, 1, 1, 2, 3, 5, 8]

Note:

Dynamic Programming is based on maintaining a Data Structure for saving previous answers instead of recursive calls.

Dynamic programming is faster than recursion but uses more space to store data.

DYNAMIC PROGRAMMING

REAL WORLD EXAMPLE

Calculate minimum number of coins required to get amount(A) with given denominations of currency(C)

For Amount (A) = 12 For denominations (C) = [1,3,6,10]

Table (M):

i/j	0	1	2	3	4	5	6	7	8	9	10	11	12
1	0	1	2	3	4	5	6	7	8	9	10	11	12
3	0	1	2	1	2	3	2	3	4	3	4	5	4
6	0	1	2	1	2	3	1	2	3	2	3	4	2
10	0	1	2	1	2	3	1	2	3	2	1	2	**2**

Hence Answer is M[d][n] = 2

∴ For change of 12, minimum of 2 coins could be used

EXCEPTION HANDLING

AN EXCEPTION IS AN ERROR THAT OCCURS WHILE A PROGRAM IS RUNNING

Traceback

When an exception occurs, a traceback is displayed. It is an error message that

- Gives information regarding line numbers that caused the exception
- Indicates the type of exception
- Shows description of the error that caused exception to be raised

Exception handler

Code Block that responds when exceptions are raised and prevents program from crashing/halting.

EXCEPTION HANDLING -STRUCTURE

EXCEPTION HANDLER STRUCTURE

Syntax

```
try:
    # statements that might raise exception
except exceptionName:
    # statements to perform if exception is raised
```

Note:

exceptionName is the type of expected error to occur - i.e IOError, SyntaxError etc

exceptionName can be a list of exceptions or it can be left blank (if left blank, it will perform if any kind of exception is raised)

Code can contain multiple exception handlers with multiple exception blocks

EXCEPTION HANDLING - CLAUSES

CLAUSES IF EXCEPTION HANDLER PROVIDE FOR ROBUST HANDLING OF ANY ERROR IN CODE

else Clause
A block of statements executed after statements in try suite, only if no exceptions were raised

finally Clause
A block of statements executed after the statements in try suite (and else) regardless of exception raised

EXCEPTION HANDLING

EXAMPLE

```
try:
    a = int(input("Enter numerator, a:"))
    b = int(input("Enter denominator, b:"))
    c = a/b
    print(f"Result of a/b={c:.3f}")
except ValueError:
    print("Entered value must be a number. Please try again.")
except ZeroDivisionError:
    print("Unable to divide by zero. Please try again.")
else:
    print("This will execute if the try executes.")
finally:
    print("This will always execute.")
```

EXERCISE

Problem

Print all the permutations of string “academy”

- Use For loops, Recursion and Dynamic Programming
- Reason which method is feasible and efficient

WEEK SUMMARY

- Learned the concept of Recursion
- Learned how to use recursion in real world problems
- Learned the difference between recursion and iteration
- Learned how to implement dynamic programming
- Learned about exceptions and implementing exception handling

THANK YOU

FOR ADDITIONAL QUERIES OR DOUBTS
CONTACT:
jerobins@ncsu.edu

AI Academy