Recursion

You will learn the definition of recursion as well as seeing how simple recursive programs work.

What Is Recursion?

“the determination of a succession of elements by operation on one or more preceding elements according to a rule or formula involving a finite number of steps”
(Merriam-Webster online)
What This Really Means

Breaking a problem down into a series of steps. The final step is reached when some basic condition is satisfied. **The solution for each step is used to solve the previous step.** The solution for all the steps together form the solution to the whole problem.

(The “Tam” translation)

Definition Of Philosophy

“…state of mind of the wise man; practical wisdom...” 

*See Metaphysics*
Metaphysics

“…know the ultimate grounds of being or what it is that really exists, embracing both psychology and ontology.”

Result Of Lookup, Possibility One: Success

• I know what Ontology means!
Result Of Lookup, Possibility One

Philosophy?

Metaphysics?

Success! I’ll take a Philosophy option.

Ontology!

Result Of Lookup, Possibility Two: Failure

- Lookup ‘loops’ back.
Result Of Lookup, Possibility Two

Philosophy?

Metaphysics?

Rats!!!

See previous

Ontology?

Ontology

“…equivalent to metaphysics.”

3 The New Webster Encyclopedic Dictionary of the English Language

Wav file from “The Simpsons”
Result Of Lookup, Possibility Three: Failure

- You’ve looked up everything and still don’t know the definition!

Looking Up A Word

```python
if (you completely understand a definition) then
    return to previous definition (using the definition that’s understood)
else
    lookup (unknown word(s))
```
Recursion In Programming

“A programming technique whereby a function calls itself either directly or indirectly.”

Direct Call

```python
def fun ():
    ...
    fun ()
    ...
```
Indirect Call

\[ f_1 \rightarrow f_2 \]

\[ f_1 \rightarrow f_2 \rightarrow f_3 \rightarrow \ldots \rightarrow f_n \]
Indirect Call (2)

Name of the example program: recursive.1py

def fun1():
    fun2()

def fun2():
    fun1()

fun1()

Requirements For *Sensible* Recursion

1) Base case
2) Progress is made (towards the base case)
Example Program: sumSeries.py

def sum(no):
    if (no == 1):
        return 1
    else:
        return (no + sum(no-1) )

def start():
    last = input ("Enter the last number: ")
    last = (int)last
    total = sum(last)
    print ("The sum of the series from 1 to", last, "is", total)

start()
When To Consider Alternatives To Recursion

- When a loop will solve the problem just as well
- Types of recursion (for both types a return statement is excepted)
  - Tail recursion
    - The last statement in the function is another recursive call to that function
      This form of recursion can easily be replaced with a loop.
  - Non-tail recursion
    - The last statement in the recursive function is not a recursive call.
    - This form of recursion is very difficult (read: impossible) to replace with a loop.

Example: Tail Recursion

- Tail recursion: A recursive call is the last statement in the recursive function.
- Name of the example program: tail.py

```python
def tail(no):
    if (no <= 3):
        print (no)
        tail(no+1)
    return()

tail(1)
```
Example: Non-Tail Recursion

- Non-Tail recursion: A statement which is not a recursive call to the function comprises the last statement in the recursive function.
- Name of the example program: nonTail.py

```python
def nonTail(no):
    if (no < 3):
        nonTail(no+1)
    print(no)
    return()

nonTail(1)
```

Error Handling Example Using Recursion

- Name of the example program: errorHandling.py

```python
-- Iterative/looping solution (month must be between 1 – 12)
month = -1
while ((month < 1) or (month > 12)):
    month = int(input("Enter birth month (1-12): "))
```
Error Handling Example Using Recursion (2)

- Iterative/looping solution (day must be between 1 – 31)

```python
def promptDay():
    day = int(input("Enter day of birth (1-31): "))
    if ((day < 1) or (day > 31)):
        day = promptDay()
    return(day)

...
day = promptDay()
```

Drawbacks Of Recursion

- Function calls can be costly
  - Uses up memory
  - Uses up time
Benefits Of Using Recursion

• Simpler solution that’s more elegant (for some problems)
• Easier to visualize solutions (for some people and certain classes of problems – typically require either: non-tail recursion to be implemented or some form of “backtracking”)

Common Pitfalls When Using Recursion

• These three pitfalls can result in a runtime error
  – No base case
  – No progress towards the base case
  – Using up too many resources (e.g., variable declarations) for each function call
No Base Case

```python
def sum(no):
    return(no + sum(no - 1))
```

No Base Case

```python
def sum(no):
    return (no + sum(no - 1))  # When does it stop???
```
No Progress Towards The Base Case

def sum (no):
    if (no == 1):
        return 1
    else:
        return (no + sum (no))

No Progress Towards The Base Case

The recursive case doesn’t make any progress towards the base (stopping) case
Using Up Too Many Resources

- Name of the example program: recursiveBloat.py

```python
def fun(no):
    print(no)
    aList = []
    for i in range (0, 1000000, 1):
        aList.append("*")
    no = no + 1
    fun(no)

fun(1)
```

Undergraduate Student Definition Of Recursion

Word: re·cur·sion

Pronunciation: ri-'kər-zhən

Definition: See recursion
Recursion: Job Interview Question


You Should Now Know

- What is a recursive computer program
- How to write and trace simple recursive programs
- What are the requirements for recursion/What are the common pitfalls of recursion