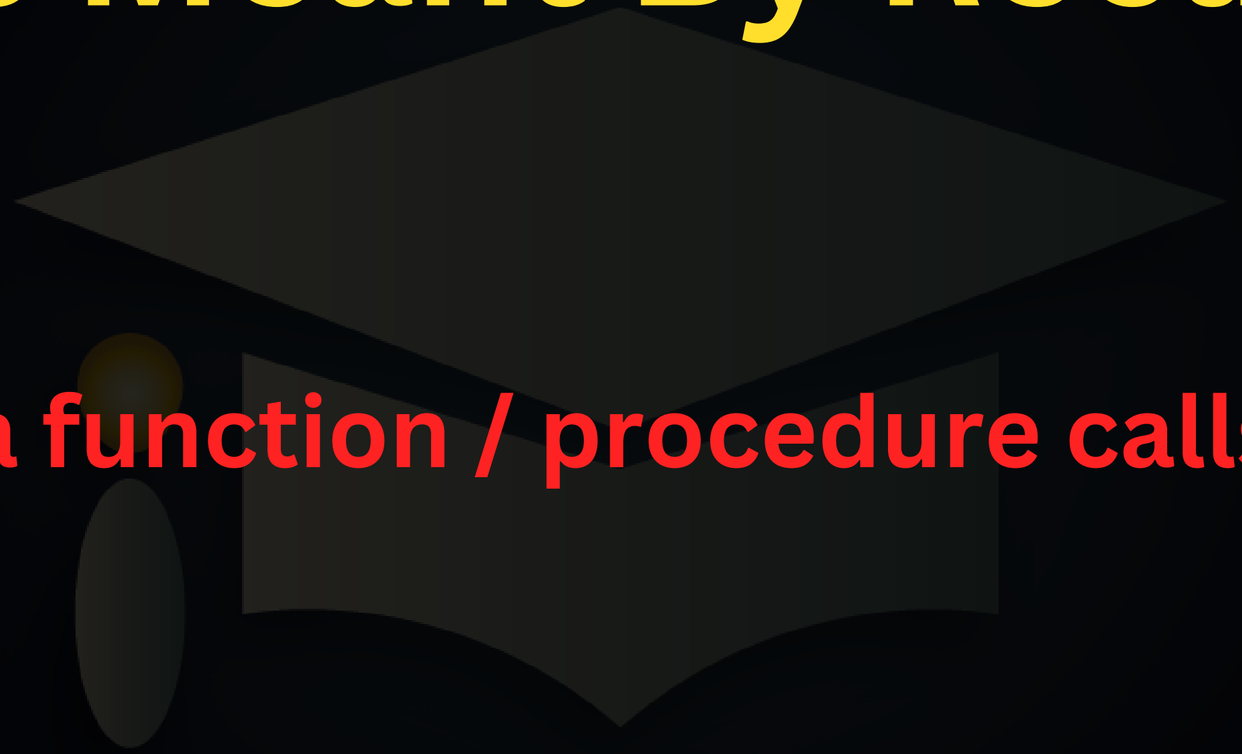
Papers Dock

PYTHON

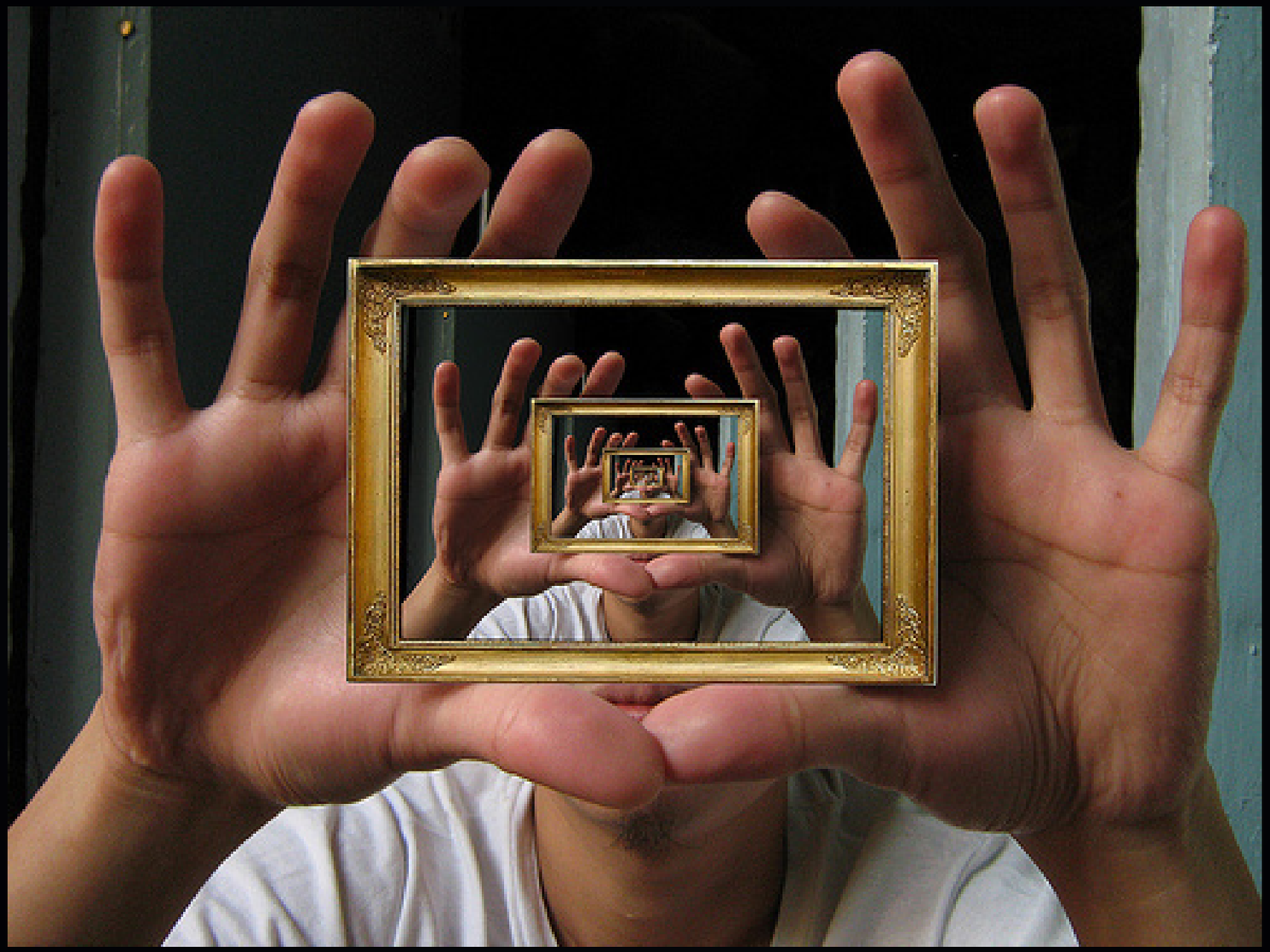
9618

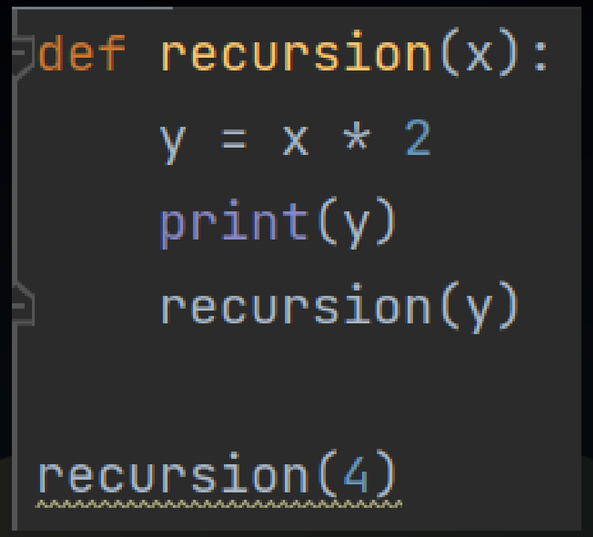
RECURSION

What Is Meant By Recursion ?



When a function / procedure calls itself



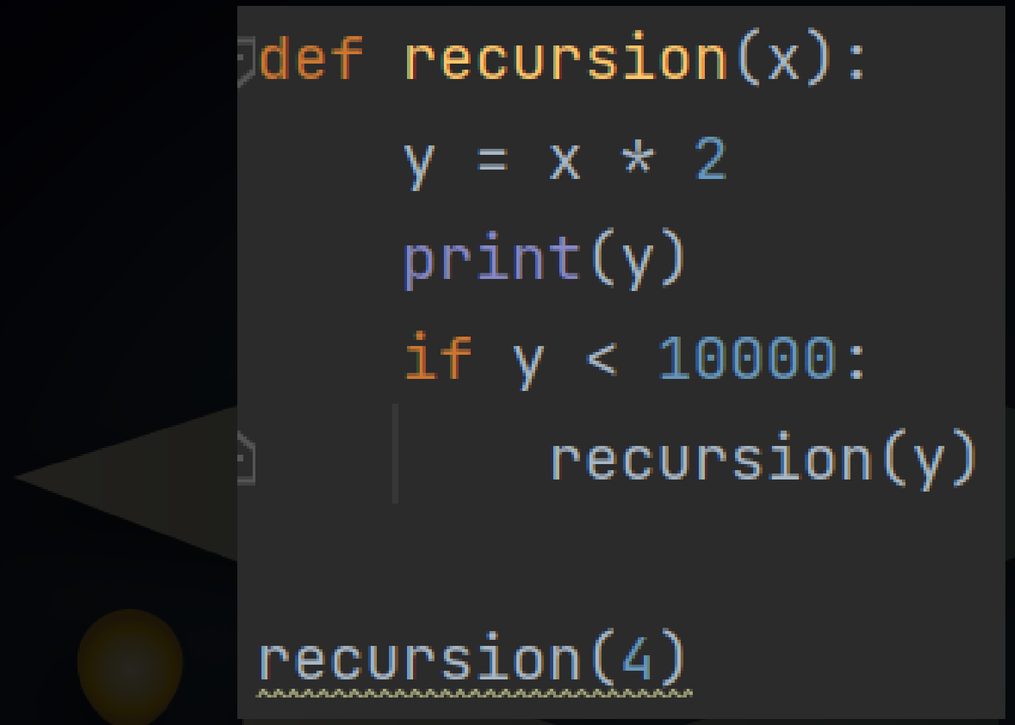


This is a recursive function in Python that takes an argument x, doubles it, and then calls itself with the result as the new argument. The function will keep calling itself recursively, doubling the argument each time, and printing out the result, until it encounters an error due to reaching Python's maximum recursion depth.

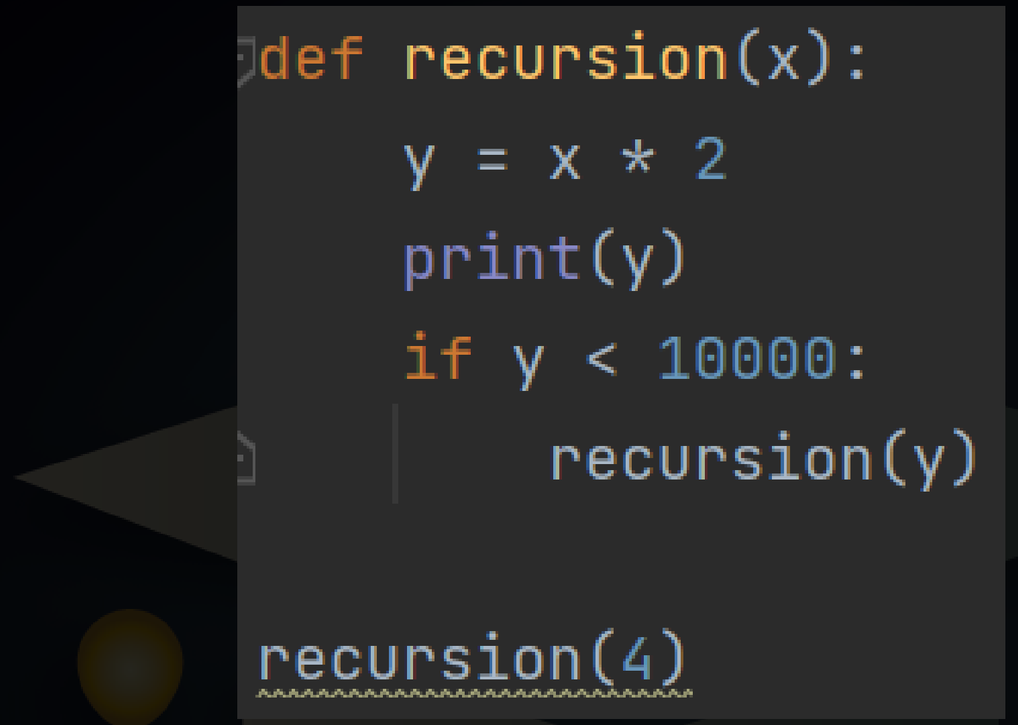
The base case and the general case are two important concepts in recursion that are used to control the flow of a recursive function.

The base case is the condition that determines when the recursion should stop. It is the condition that prevents the function from calling itself again and again, and allows the function to return a value. In other words, the base case is the stopping point for the recursion. It is the case where the problem is simple enough that the function does not need to call itself anymore.

The general case, on the other hand, is the opposite of the base case. It is the condition that determines when the recursion should continue. In the general case, the function is not yet at the stopping point and needs to call itself again with a smaller or simpler version of the problem. This continues until the base case is reached.



the base case is when the value of y exceeds or equals 10000, which means that the recursion stops and the function does not call itself anymore.



The base case is when y >= 10000

The general case is when y < 10000

What Is Meant By Recursive Algorithm ?

Any algorithm that have a base case, work towards a base case, have a general case and calls itself.

(b) The following recursive procedure outputs every even number from the positive parameter value down to and including 2.

The procedure checks if the integer parameter is an even or an odd number. If the number is odd, the procedure converts it to an even number by subtracting 1 from it.

The function `MOD(ThisNum : INTEGER, ThisDiv : INTEGER)` returns the remainder value when `ThisNum` is divided by `ThisDiv`.

Complete the **pseudocode** for the recursive procedure.

```
PROCEDURE Count(BYVALUE ..... : INTEGER)

    IF .....(Number, 2) <> 0

        THEN

            Number ← Number - 1

    ENDIF

    OUTPUT .....

    IF Number > 0

        THEN

            .....(..... - 1)

    ENDIF

ENDPROCEDURE
```

[5]

(b) The following recursive procedure outputs every even number from the positive parameter value down to and including 2.

The procedure checks if the integer parameter is an even or an odd number. If the number is odd, the procedure converts it to an even number by subtracting 1 from it.

The function `MOD(ThisNum : INTEGER, ThisDiv : INTEGER)` returns the remainder value when `ThisNum` is divided by `ThisDiv`.

Complete the **pseudocode** for the recursive procedure.

```
PROCEDURE Count(BYVALUE Number : INTEGER)
    IF MOD(Number, 2) <> 0
        THEN
            Number ← Number - 1
    ENDIF
    OUTPUT Number
    IF Number > 0
        THEN
            CALL Count(Number - 1)
    ENDIF
ENDPROCEDURE
```

62

[5]

Concept of winding and unwinding

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

temp = factorial(5)
print(temp)
```

winding : Is done in general case in which the function calls are getting pushed in the stack no calculation is done in this phase

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

temp = factorial(5)
print(temp)
```

Unwinding : When the base case is reached all the data stored in stack gets popped out from top of the stack

```python
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n-1)

temp = factorial(5)
print(temp)
```

Explain what a compiler has to do to implement recursion?

When the recursive call is made all value are put on the stack which is known as winding when the base case is met the algorithm unwinds. The last set of values are taken off the stack un reverse order

1 Study the following pseudocode for a recursive function.

```
FUNCTION Unknown(BYVAL X, BYVAL Y : INTEGER) RETURNS INTEGER
  IF X < Y THEN
    OUTPUT X + Y
    RETURN (Unknown(X + 1, Y) * 2)
  ELSE
    IF X = Y THEN
      RETURN 1
    ELSE
      OUTPUT X + Y
      RETURN (Unknown(X - 1, Y) DIV 2)
    ENDIF
  ENDIF
ENDFUNCTION
```

The operator `DIV` returns the integer value after division e.g. `13 DIV 2` would give `6`

(a) Write program code to declare the function `Unknown()`.

```
def Unknown(X, Y):
    if X < Y:
        print(X + Y)
        return(Unknown(X + 1, Y) * 2)
    elif X == Y:
        return 1
    else:
        print(X + Y)
        return (Unknown(X -1, Y) // 2)
```

the function `Unknown()` as an iterative function, `Iterative`

```
def IterativeUnknown(X, Y):
    Total = 1
    while X != Y:
        if X < Y:
            print(str(X + Y))
            X = X + 1
            Total = Total * 2
        else:
            print(str(X + Y))
            X = X - 1
            Total = Total // 2
    return Total
```

```
print("10 and 15")
print(str(IterativeUnknown(10, 15)))
print("10 and 10")
print(str(IterativeUnknown(10, 10)))
print("15 and 10")
print(str(IterativeUnknown(15, 10)))
```