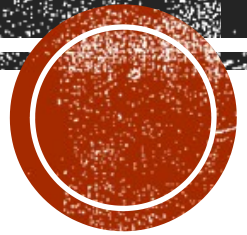


FUNCTIONS



OPTIONAL PARAMETERS

```
def func_name(par1, par2 = 5):  
    return par1 + par2
```

```
>>> func_name(4,9)
```

```
>>> func_name(3)
```



REVIEW: FUNCTION CALLING:

Positional

Keyword arguments



REVIEW: BOOLEAN FUNCTIONS

Functions that return Boolean values

```
def is_divisible_by_2(num):  
    return (num % 2 == 0)
```



REVIEW: DEBUGGING IN FUNCTIONS

- Preconditions
- Postconditions



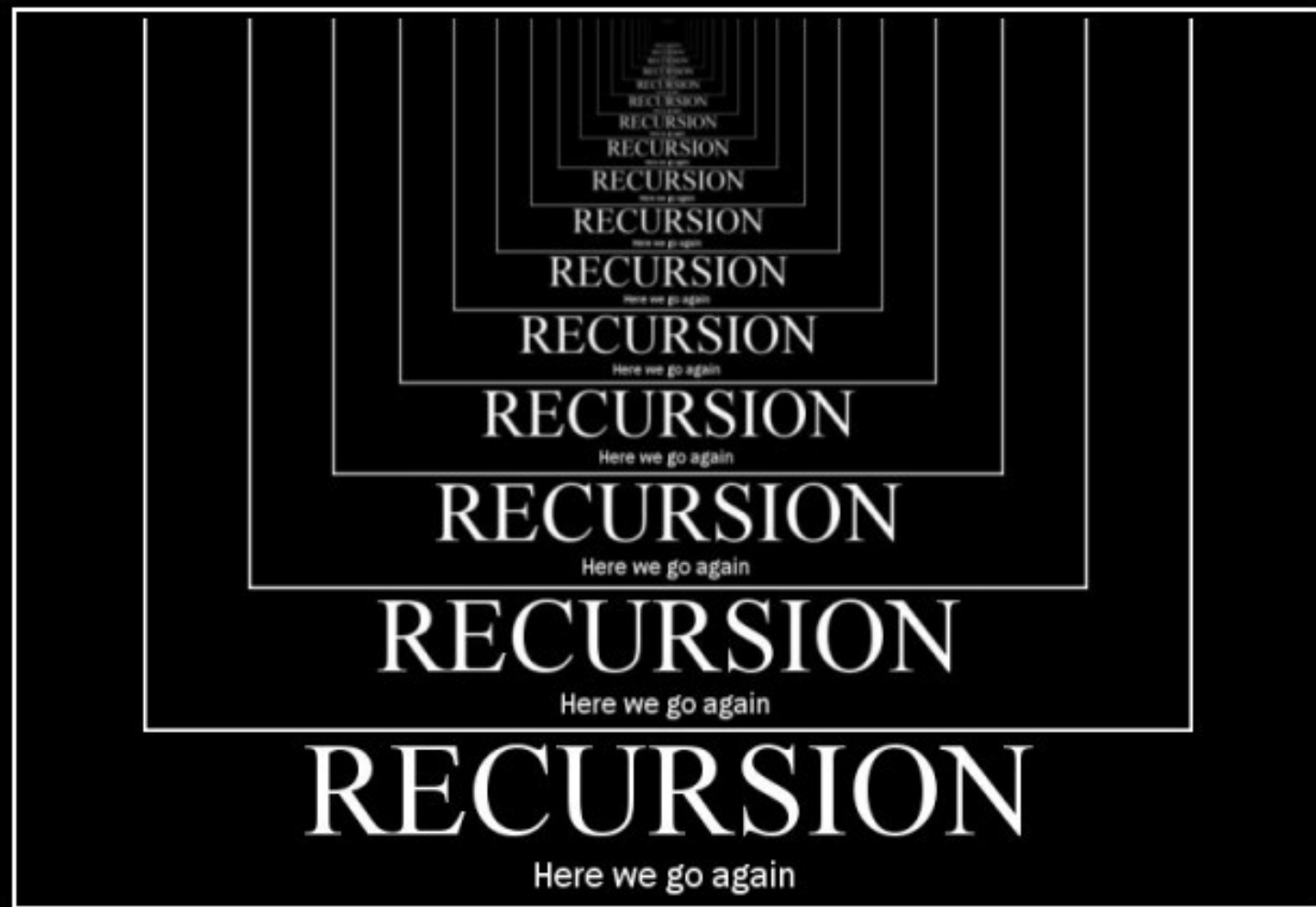
UNKNOWN NUMBER OF PARAMETERS

```
def add(*params):  
    total = 0  
    for i in params:  
        total += i  
    return total
```

```
>>> add(1,2,4)
```

```
>>> add(1,2,3,5,4,3)
```





RECURSION

Here we go again

RECURSION

A technique that solves problem
by solving smaller versions of the
same problem!





RECURSION

When you turn this into a program,
you end up with functions that call
themselves (i.e., *recursive functions*)



FACTORIAL

$$n! = n * (n-1)!$$

$$f(n) = n * f(n-1)$$



FACTORIAL

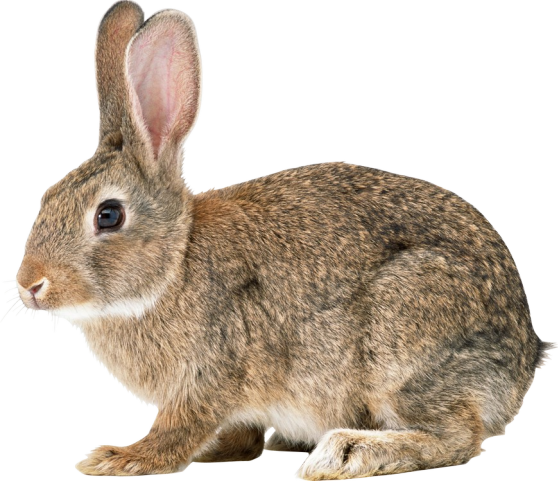
```
def factorial(n):  
    assert(n > 0)  
    if n == 1:  
        return 1  
    else:  
        return n * factorial(n - 1)
```



RECURSION PARTS

1. Base case
2. Recursive Call

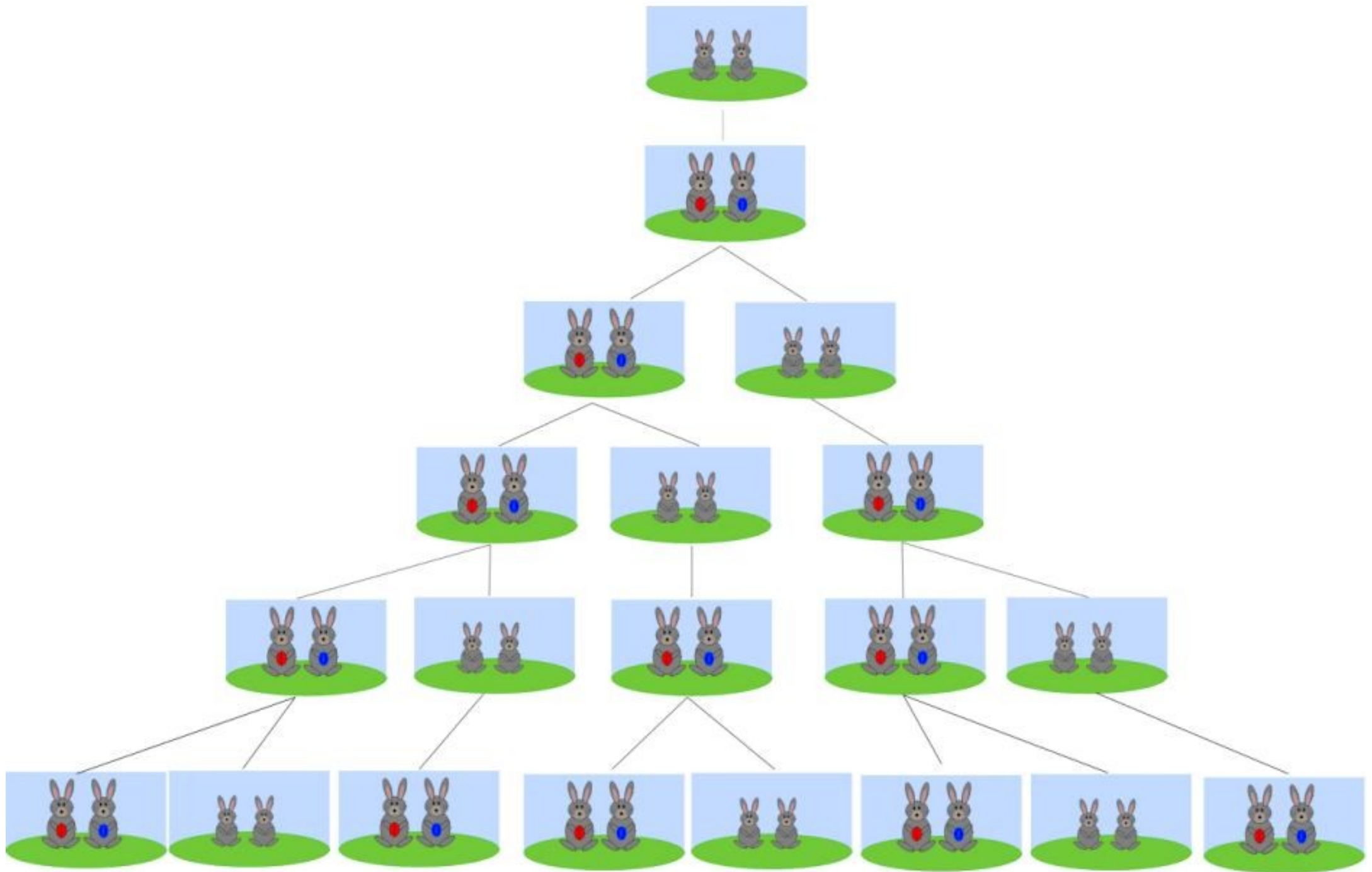




FIBONACCI SEQUENCE

0, 1, 1, 2, 3, 5, 8, 13...





FIBONACCI SEQUENCE

$$f(n) = f(n-1) + f(n-2)$$

$$\text{where } f(0) = 1$$

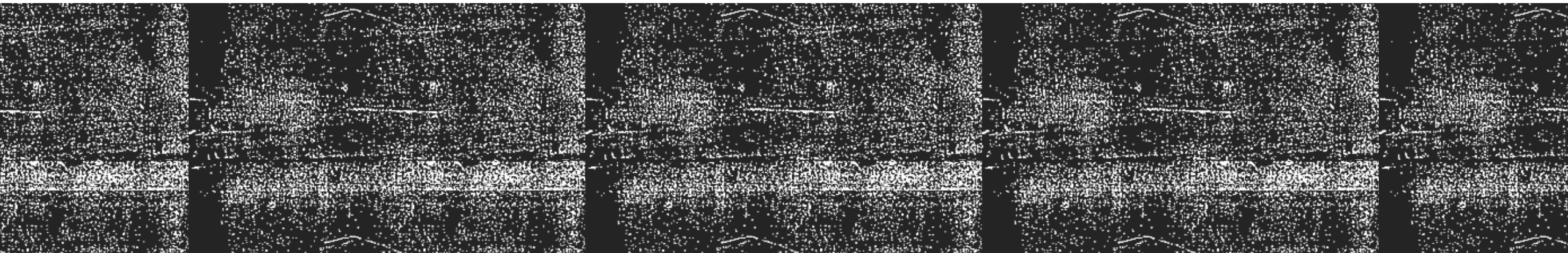
$$f(1) = 1$$



INFINITE RECURSION

```
def infinite(n):  
    if n == 1:  
        return 0  
    return n * infinite(n + 1)
```





STRING DATA-TYPE

- A string is a **sequence of characters**
- A string literal uses quotes 'Hello' or “Hello”
- For strings, + means “concatenate”
- When a string contains numbers, it is still a string



TRAVERSING

Computations on string start at the beginning, select each character in turn, do something to it, and continue until the end. This pattern of processing is called a **traversal**.



TRAVERSAL: FOR LOOP

```
for letter in  
'banana':  
    print(letter)
```



The **iteration variable** “iterates” through the **string** and the **block (body)** of code is executed once for each value in the **sequence**



LOOKING INSIDE STRINGS

We can get at any single character in a string using an index specified in square brackets

b	a	n	a	n	a
0	1	2	3	4	5

```
>>> fruit = 'banana'  
>>> print(fruit[1])
```



INDEXING OPERATOR

The expression in brackets is called an **index**

The index value must be an integer and starts at zero

```
>>> fruit = 'banana'
```

```
>>> letter = fruit[1]
```

```
>>> print(letter)
```

```
a
```

```
>>> n = 3
```

```
>>> w = fruit[n - 1]
```



INDEXING OPERATOR

b	a	n	a	n	a
0	1	2	3	4	5

```
>>> print(fruit[-1])
```



LENGTH OF A STRING

- The len function, when applied to a string, returns the number of characters in a string

```
>>> fruit = "banana"
```

```
>>> len(fruit)
```

```
6
```



TRAVERSAL: WHILE LOOP

```
fruit = 'banana'
```

```
ctr = 0
```

```
while (ctr < len(fruit))  
    print(fruit[ctr])  
    ctr += 1
```



SLICING STRINGS

- A *substring* of a string is obtained by taking a **slice**.
- Need to get continuous section of a string using a **colon operator**.



SLICING

M	o	n	t	y		P	y	t	h	o	n
0	1	2	3	4	5	6	7	8	9	10	11

```
>>> s = 'Monty Python'
```

```
>>> print(s[0:2])
```

```
print(s[:2])
```

```
Mo
```



SLICING

M	o	n	t	y		P	y	t	h	o	n
0	1	2	3	4	5	6	7	8	9	10	11

```
>>> print s[8:]
```

Thon

```
>>> print s[:]
```

Monty Python



A CHARACTER TOO FAR

```
>>> zot = 'abc'
```

```
>>> print(zot[5])
```

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

IndexError: string index out of range



A CHARACTER TOO FAR

```
>>> zot = 'abc'
```

```
>>> print(zot[len(zot)])
```

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

IndexError: string index out of range

