Programming in Python Lecture 5- Review

Plan for to day

- Lecture 1-Variables-Numbers,Strings, Computational Operators, Logical Operators
- Lecture 2-Lists, Tuples, Dictionaries
- Lecture 3-If statements, For loop, While loop
- Lecture 4-Functions, Lambda, Recursion

Homework

1. Create a Python script to create and print a list where the values are square of numbers between 1 and n (both included)

Input:

```
>>> 5
```

Output:

```
>>>  [1, 4, 9, 16, 25]
```

2. -Create a Python script to find all keys in the provided dictionary that have the given value.

Input:

```
>>> students = {'Theodore': 19,'Roxanne': 20,'Mathew': 21,'Betty': 20}
```

Output:

```
>>> ['Roxanne', 'Betty']
```

Homework

3. Create a Python script to calculate the value of the following expression by using lambda function.(x*10+(y/2)*z)

Input:

```
>>>  x = 5, y = 5, z = 5
```

Output:

```
>>>  62.5
```

4. Create a Python script to create a lambda function that multiplies argument x with argument y and print the result

Input:

```
>>>  a=4, b=12
```

Output:

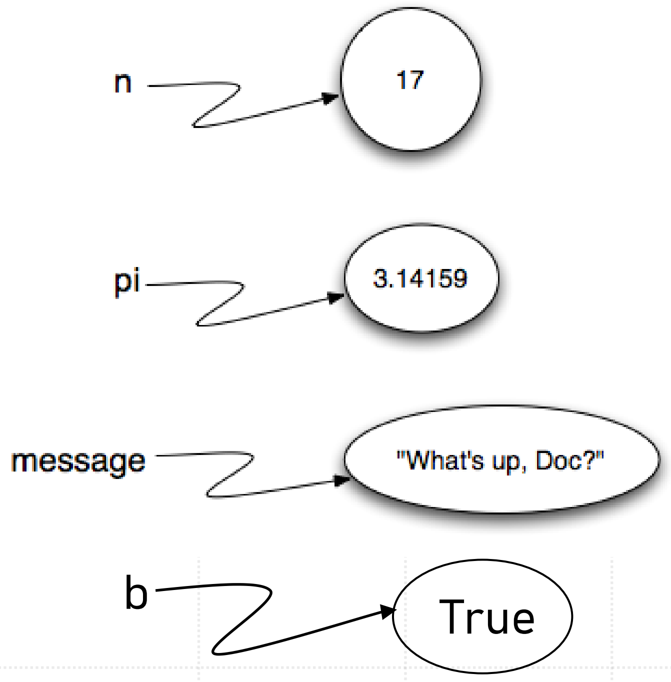
```
>>>  48
```

Lecture 1

- Variables
    - Numbers
    - Strings
- Computational Operators
- Logical Operators

Why Do We Need Different Types?

- Saving memory
- Execution speed
- Variables types:
  - Int(integer)
  - Float(numbers with decimal point)
  - Strings(text sequences)
  - Booleans(True or False)



Arithmetic Operators

Operator	Use	Description
+	x + y	Adds x to y
–	x – y	Subtracts x from y
*	x * y	Multiplies x by y
**	x ** y	X to the power y
/	x / y	Divides x by y
%	x % y	Computes the remainder of dividing x by y

Strings Slicing

```
str="51689”
print( str[1])
>>>‘1‘
print( str[0:3])
>>>‘516‘
print( str[1:])
>>> '1689"
print( str[-3:-1])
>>> ‘16’
print( str[:-3])
>>> ‘51"
print([::-1])
>>> ‘98615’
```

5	1	6	8	9
0	1	2	3	4
-5	-4	-3	-2	-1

Strings concatenation

```
word1 = "Hello"
word2 = “World"
print(word1 + word2)
>>> 'HelloWorld’
print(word1 + ‘ ‘+word2)
>>> 'Hello World’
```

Strings Built In Methods

https://www.w3schools.com/python/python_ref_string.asp

- Len(len(str))- the function returns the number of items (length) in an object.
- Upper(upper.str)- Converts a string into upper case.
- Lower(lower.str)- Converts a string into lower case.
- Replace(replace.str)- Returns a string where a specified value is replaced with a specified value.
- Count(count.str)- Returns the number of times a specified value occurs in a string.
- Split (str. Split())- split a string according to an argument(default-spices)

Strings Built In Methods

https://www.w3schools.com/python/python_ref_string.asp

- str.isalpha()- return true if a cher in string is a letter
- str.isdigit()-return true if a cher in string is a number

Comparison Operators

- Compares two variables and returns a Boolean type result/variable

Operator	Name	Description
x < y	Less than	true if x is less than y, otherwise false.
x > y	Greater than	true if x is greater than y, otherwise false.
x <= y	Less than or equal to	true if x is less than or equal to y, otherwise false.
x >= y	Greater than or equal to	true if x is greater than or equal to y, otherwise false.
x == y	Equal	true if x equals y, otherwise false.
x != y	Not Equal	true if x is not equal to y, otherwise false.

Logical Operators

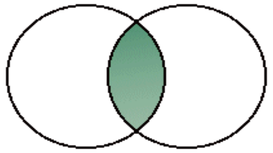
Operates on two Booleans and returns a Boolean

Operator	Description
x **and** y	Both True: **True**, otherwise: **False**.
x **or** y	At least one is True: **True**, Otherwise: **False**.
not x	x is False → **True**, x is True → **False**

And, or, not

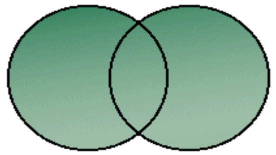
and

- The guy is tall **and** nice



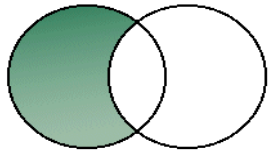
or

- The guy is either tall **or** nice



not

- The guy is **not** tall



Questions?

Hands On

Lecture 2

- Lists
- Tuples
- Dictionaries

Lists are Indexable

Remember this?

```
>>> str="We are learning Python!!!“
>>> str[1:3]
‘e ‘
>>> str[0:3]
‘We ‘
>>> str[1:]
"e are learning Python!!!’'
>>> str[-4:-2]
'n!’
>>> str[:-3]
'We are learning Python"
>>> str[-3:]
'!!!’
```

The same indexing + slicing works for lists!

Lists are Indexable

```
>>> list = [9,8,6,1,5]
>>> list[0]
9
>>> list[4]
5
>>> list[-3]
6
>>> list[::2]
[9,6,5]
>>> my_list[5]
Traceback (most recent call last):
File "<pyshell#7>", line 1, in <module>
my list[5]
IndexError: list index out of range
```

9	8	6	1	5
0	1	2	3	4
-5	-4	-3	-2	-1

Assignments of List Variables

```
>>> list_1= [1,2,3]
>>> list_2= list_1
>>> list_1 = [6,7,8,9]
>>> list_2
[1,2,3]
>>> list_1
[6,7,8,9]
```

So far - no surprises

Assignments of List Variables

```
>>> list_2= list_1
>>> list_1[0] = 1000
>>> list_1
[1000,7,8,9]
>>> list_2
[1000,7,8,9]
```

Surprise!

Nested Lists

```
NL = [ [3, 7, 2],[6, 0, 1] ]

Print(NL[1])

>>>[6, 0, 1]

Print(NL[1][0])

>>> 6

len(NL)

>>> 2
```

List Methods

Function	Description
L. append(elem)	Adds an element at the end of the list.
L. clear()	Removes all the elements from the list
L. copy()	Returns a copy of the list
L. count()	Returns the number of elements with the specified value
L. extend(elem)	Add the elements of a list (or any iterable), to the end of the current list
L. index(num)	Returns the index of the first element with the specified value
L. insert(elem)	Adds an element at the specified position
L. pop(elem,index)	Removes the element at the specified position
L. remove(elem)	Removes the first item with the specified value
L. reverse()	Reverses the order of the list
L. sort()	Sorts the list

Tuples are immutable lists

```
>>> list = [1,2,3]
>>> list [1]=10
>>> tuple = (1,2,3)
>>> tuple[1] = 10
Traceback (most recent call last):
  File "<pyshell#20>", line 1, in <module>
   my_tuple[1] = 10
TypeError: 'tuple' object does not support item assignment
```

Tuples

A tuple is similar to a list, but it is immutable.

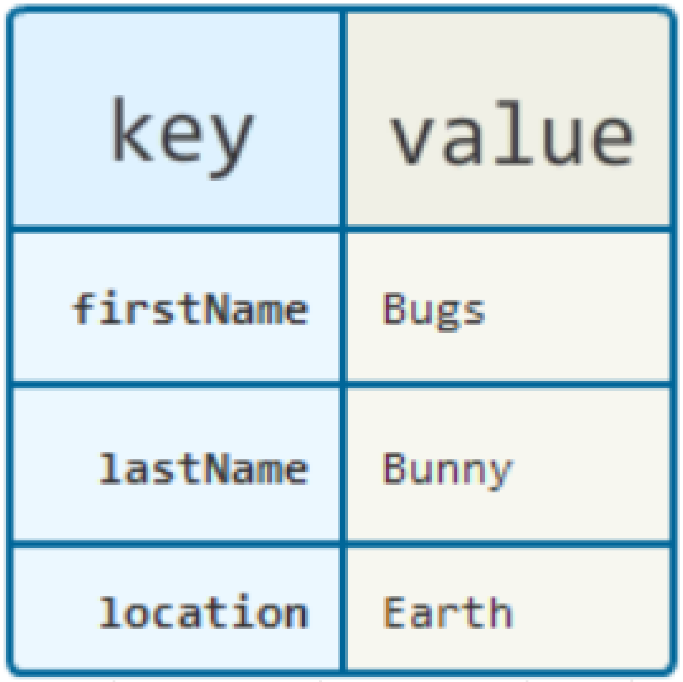
```
>>> B = (“Let", “It", "be") # definition
>>> B
(“Let", “It", "be")
>>> B[0]       # indexing
"Let"
>>> B[-1]      # backwords indexing
‘Be'
>>> B[1:2]     # slicing
(‘It')
```

Tuples

- Fixed size
- Immutable (similarly to Strings)
- What are they good for (compared to list)?
    - Simpler (“light weight”)
    - Staff multiple things into a single container
    - Immutable (e.g., records in database, safe code)

Dictionaries

- Key – Value mapping
  - No order
- Fast!
- Usage examples:
  - Database
  - Dictionary
  - Phone book



key	value
firstName	Bugs
lastName	Bunny
location	Earth

Dictionaries

Access to the data in the dictionary:

- A key is one-to-one function
- Given a key, it is easy to get the value.
- Given a value, you need to go over all the dictionary to get the key.

Dictionaries

Example: ID list- Map names to IDs:

```
>>> ID_list = {'Eric': ‘30145', ‘Shlomi': ‘38171',
  ‘Kobi': ‘85736'}
>>> print(ID_list)
{'Eric': ‘30145', ‘Shlomi': ‘38171', ‘Kobi':
  ‘85736'}
```

Note:The pairs order changed!

Dictionaries

Access dictionary Items:

```
>>> ID_list ['Eric’]
'30145'
```

Add a new person:

```
>>> ID_list[‘David'] = ‘84759'
>>> print(ID_list )
{'Eric': ‘30145', ‘Shlomi': ‘38171', ‘Kobi':
  ‘85736’, ‘David’:‘84759' }
```

Dictionaries

What happens when we add a key that already exists?

```
>>> ID_list [‘David']= ‘75647'

>>> print(ID_list)

{'Eric': ‘30145', ‘Shlomi': ‘38171', ‘Kobi':
  ‘85736’, ‘David’:‘75647 ' }
```

How can we add another Kenny McCormick in the phone book?

Dictionary Methods

Function	Description
clear()	Removes all the elements from the dictionary
copy()	Returns a copy of the dictionary
fromkeys()	Returns a dictionary with the specified keys and values
get()	Returns the value of the specified key
keys()	Returns a list containing the dictionary's keys
pop()	Removes the element with the specified key
items()	Returns a list containing a tuple for each key value pair
popitem()	Removes the last inserted key-value pair
update()	Updates the dictionary with the specified key-value pairs
values()	Returns a list of all the values in the dictionary

Questions?

Hands On

Lecture 3

- If statements
- For loop
- While loop

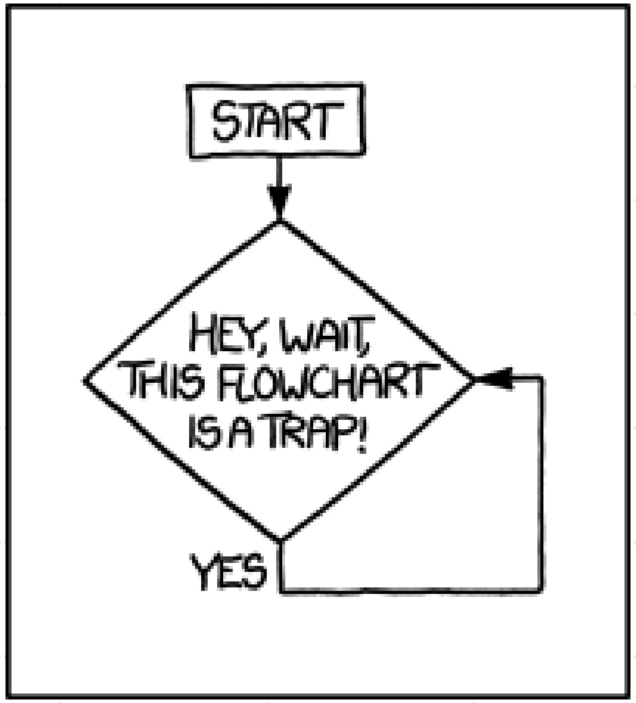
Flow Control

Different inputs →Different execution order

- Computer games
- Illegal input

Control structures

- **if-else**
- **for loop**
- **while** loop



http://xkcd.com/1195/

Conditional Statement: if

Used to execute statements conditionally

<u>Syntax</u>

```
if condition:
    statement1
    statement2
```

*If condition is **True**, statements are executed*
Condition *= expression that evaluates to a Boolean*

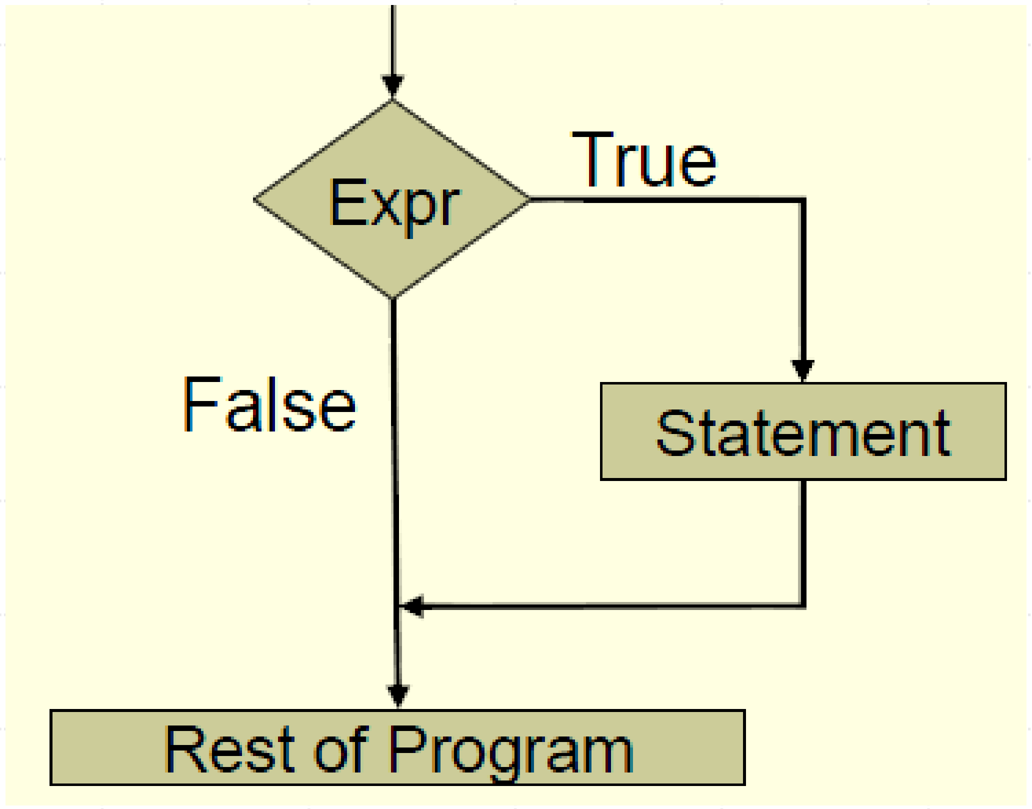
Indentation:

Following the if statement:

Open a new scope = one tab to the right.

Indicates the commands within the scope of this if.

Conditional Statements



elif

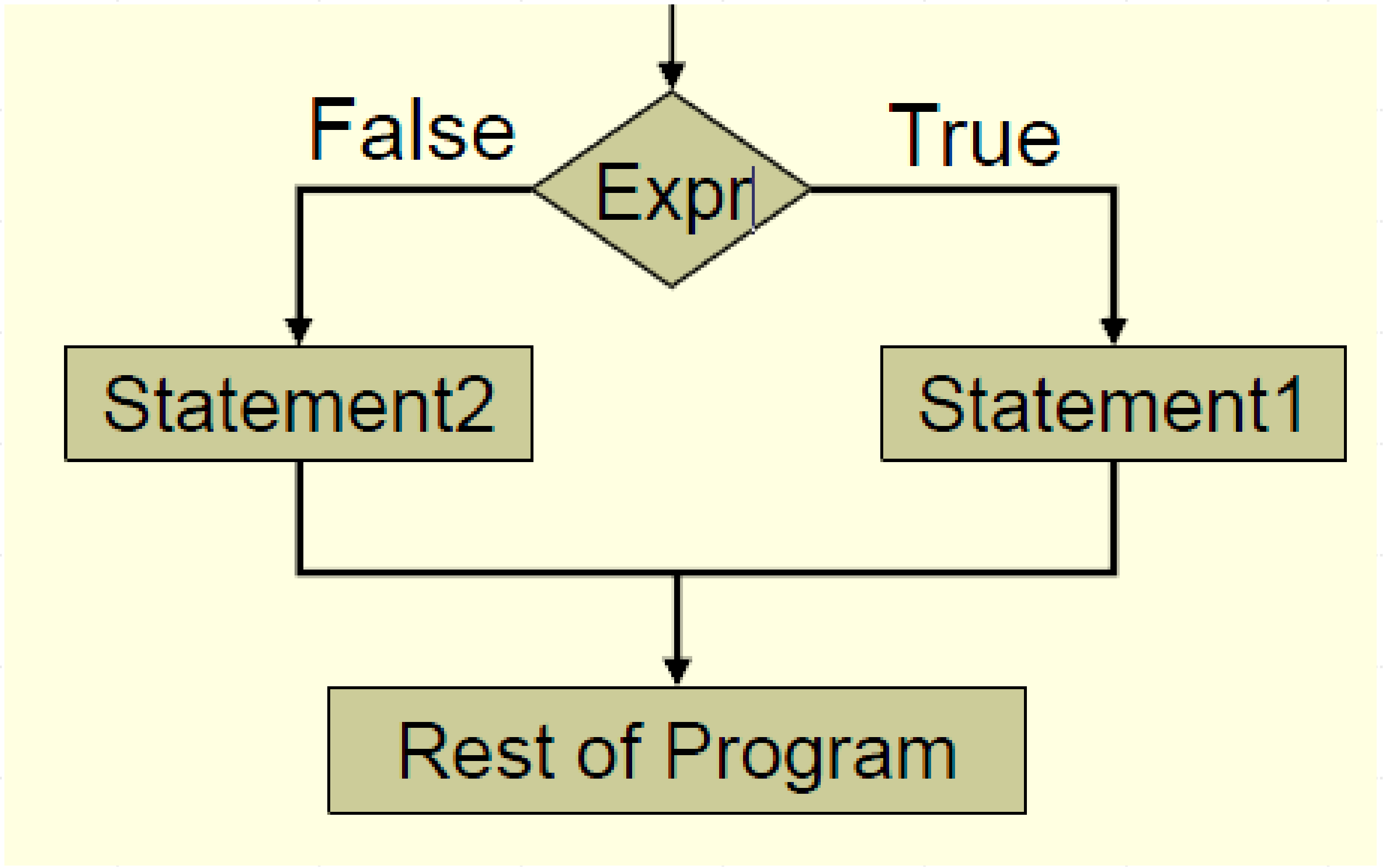
```
if  condition1:
    statement1
elif condition2:
    statement2
else:
    statement3
```

condition1 is true → execute ***statement1***

condition1 false and **condition2** true → execute ***statement2***

both conditions are false → execute ***statement3***

elif



For Loop

```
for element in iterable:
  statement1
  statement2
    …
```

Run over all elements in the object (list, string, etc.)

`Iteration 0:` Assign element = object[0]

- Execute the statements

`Iteration 1:` Assign *element* = object*[1]*

- Execute the statements

…

For Loop

Indentation determines the scope of the iteration.

Note

No infinite lists in Python !!!

No infinite for loops!!!

Range

An ordered list of integers in the range.

```
>>> range(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

range(from, to) contains all integers k satisfying from ≤ k **<** to.

range(to) is a shorthand for range(0, to).

```
>>> range(2,10)
[ 2, 3, 4, 5, 6, 7, 8, 9]
>>> range(-2,2)
[-2, -1, 0, 1]
>>> range(4,2)
[]
```

Range

```
>>> type(range(3))
<type 'list'>
```

Step size:

`range(from, to, step)` returns:

from, *from+step*, *from+2*step*,…, *from+i*step*

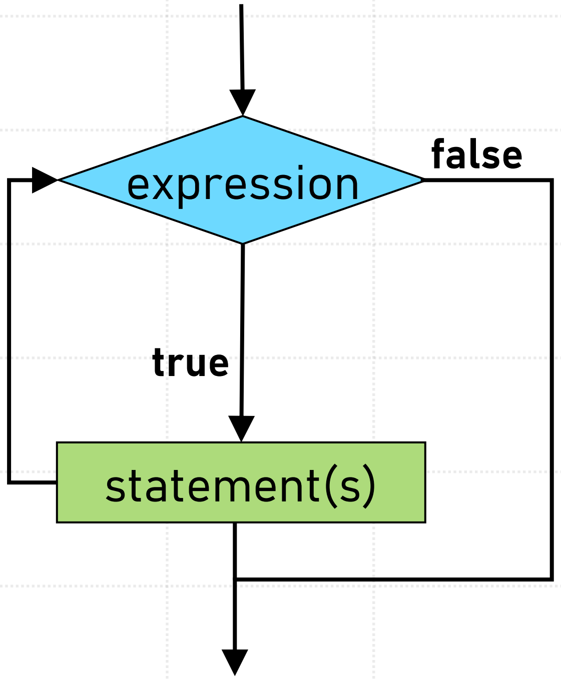
until *to* is reached, not including to itself.

```
>>> range(0,10,2)
[0, 2, 4, 6, 8]
>>> range(10,0,-2)
[10, 8, 6, 4, 2]
```

While Loop

Used to repeat the same instructions until a stop criterion is met

```
while expression:
  statement1
  statement2
  ...
```



Questions?

Hands On

Lecture 4

- Functions
  - What are they good for
  - Built-in Functions
  - Defining New Functions
  - Functions call functions
- Lambda
- Recursion

Why use functions ?

- **We** wrote code calculating $4! + 7! + 9!$.

To use the code in 3 different calculations, he:

  - copy & pasted
  - assigned arguments

3 times.

The calculation took 0.0046 microseconds.

- There is a more efficient algorithm to calculate $4! + 7! + 9!$.
- To update the code, he went over the 3 calculations ☹
- After the update, the calculation took only 0.0006 microseconds – over 7 times faster!

→ Don’t duplicate code, use functions !

Modularity enables code reuse !

Definition

Modularity is the degree to which a system's components may be separated and recombined (Wikipedia).

- Top-down design
- Improves maintainability
- Enforces logical boundaries between components

Scope of a function

- Variables defined within a function are considered **local**, and can be used only within the function's block of code.
- Local variables can mask variables with the same name defined outside of a function (**Global** variables).

Built-in Functions

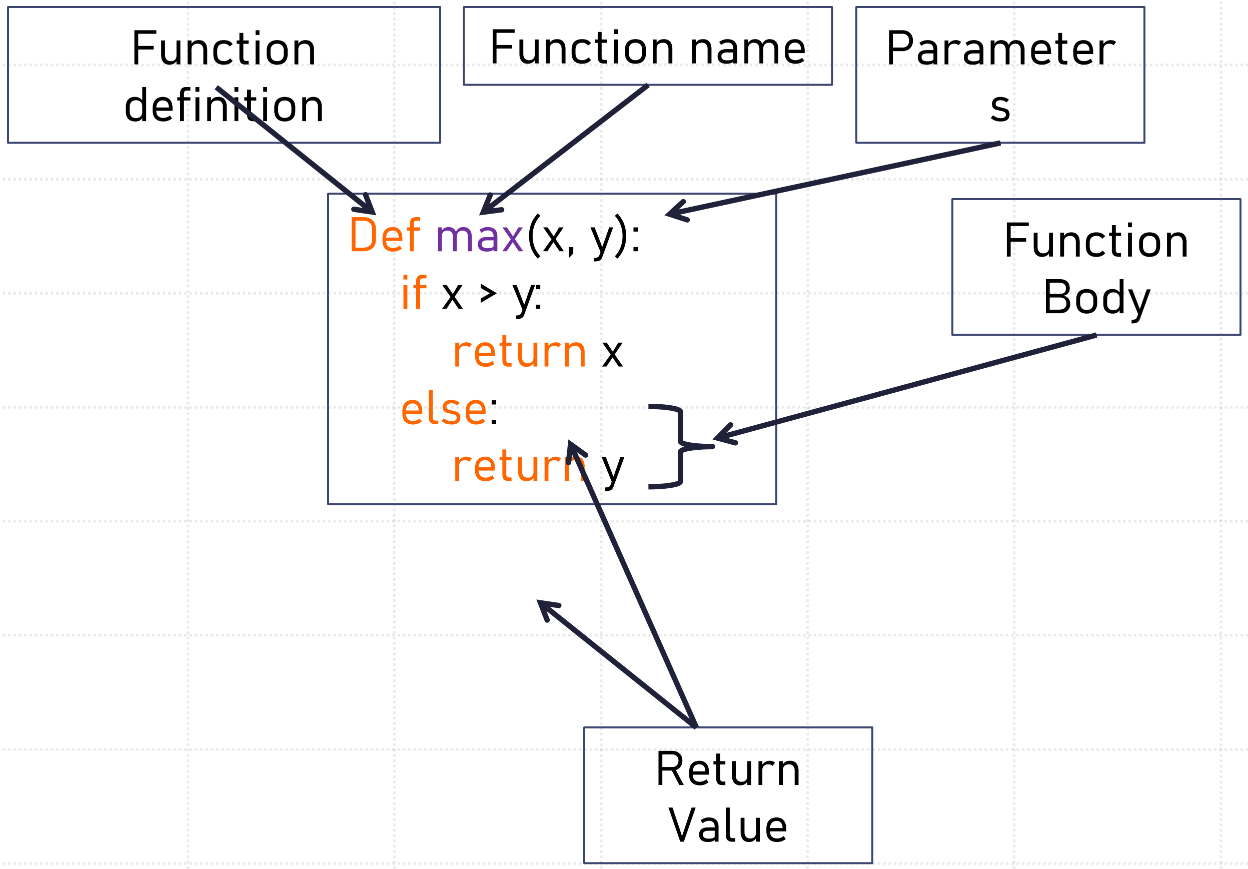
http://docs.python.org/library/functions.html

We already used built-in functions:

```
>>> type(5)
<type 'int'>


>>> len(range(8,100, 8))
12
```

Function Definition in Python



```
Def max(x, y):
    if x > y:
        return x
    else:
        return y
```

Function’s Input / Output

Input: Arguments

Can be of any type - int / float / str / Boolean / list

Output: The *Return* statement

Returns a value to the function caller.

- Returned value can be any Python type
  - Multiple values are wrapped in list
- If no *Return* or no values is specified - returns *None*
- **Different from print**
- ***Return* stops the function’s execution and returns to the caller**

Functions call functions

```
def print_text():
        print “Is this the real life”
        print “Is this the real life or it’s just fantasy”
def text_2():
        print_text()
        print_text()
>>> text_2()
Is this the real life
Is this the real life or it’s just fantasy
Is this the real life
Is this the real life or it’s just fantasy
```

Default Arguments For Functions

- We can specify default values for the arguments of the function.
- The default value will be used only when the function is called **without** specifying a value for the argument.

```
def f1(x, y=1):
    return x+y

>>> f1(1,2)     # In this call: x = 1, y = 2
3
>>> f1(3)    # In this call: x = 3, y = 1 (the default value)
4
>>> f1()     # x doesn't have a default value, it must be specified

Traceback (most recent call last):
  File "<pyshell#12>", line 1, in <module>
    f1()
TypeError: f1() takes at least 1 argument (0 given)
```

Lambda

- Lambda is a special command that is used to quickly create in-line functions for specific usecases.
- The lambda operator or lambda function is a way to create small anonymous functions, i.e. functions without a name .
- These functions are throw-away functions, i.e. they are just needed where they have been created .
- Lambda functions are mainly used in combination with the functions filter(), map() and reduce .()
- These kind of functions are heavily used in Apache Spark for example.

Recursion

Recursive function:

A function whose implementation calls itself (with different arguements).

Recursive Solution

A solution to a “large” problem using solutions to “small” problems that assemble it.

Recursion Example in Python

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

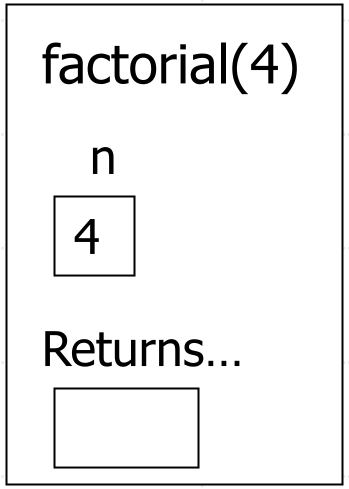
stop condition

calculate the result
using a recursive call

advance towards base case

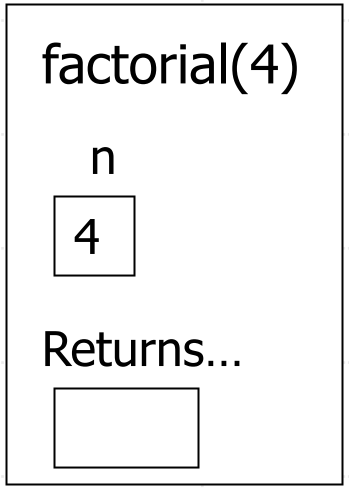
Recursive factorial – step by step

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



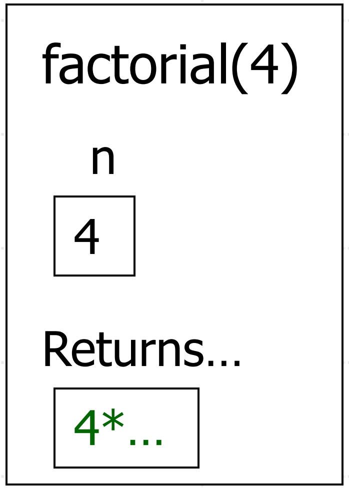
Recursive factorial – step by step

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



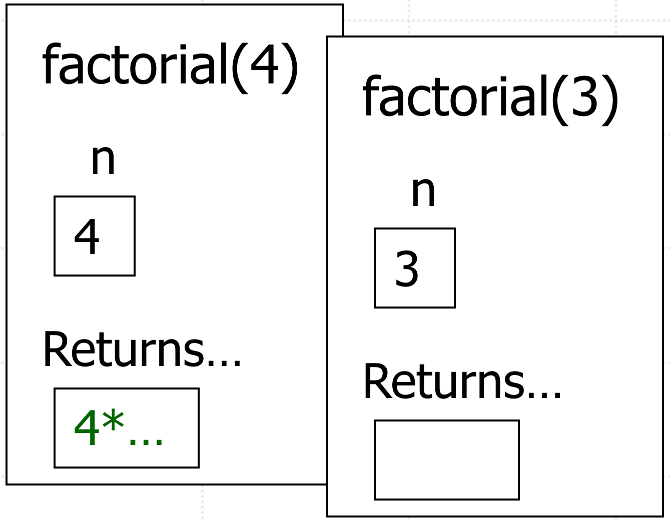
Recursive factorial – step by step

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



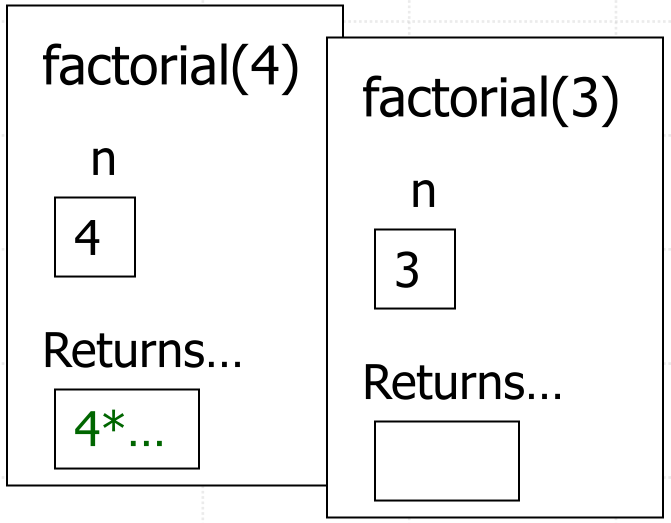
Recursive factorial – step by step

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



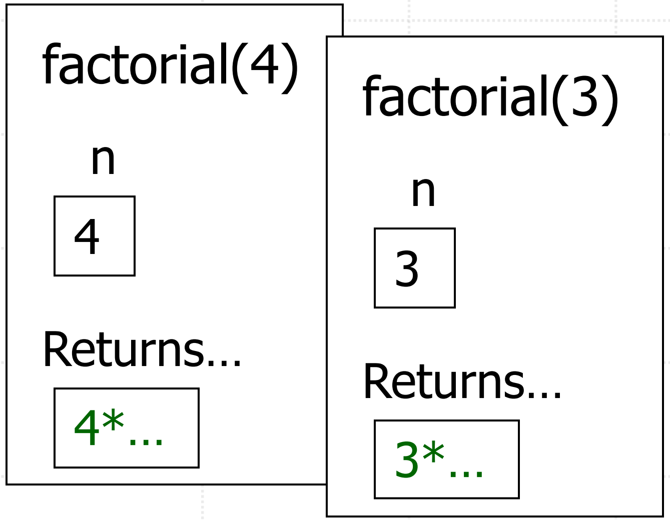
Recursive factorial – step by step

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



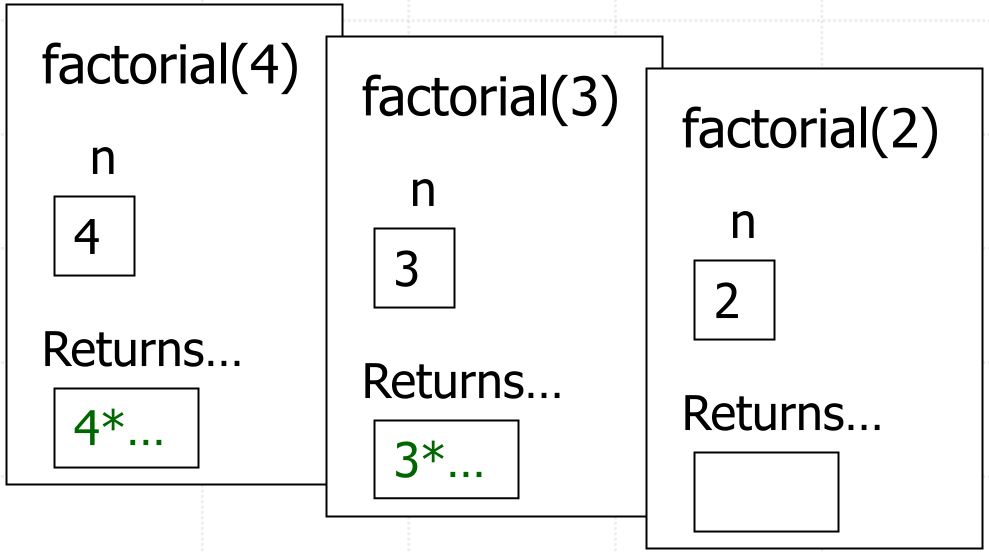
Recursive factorial – step by step

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



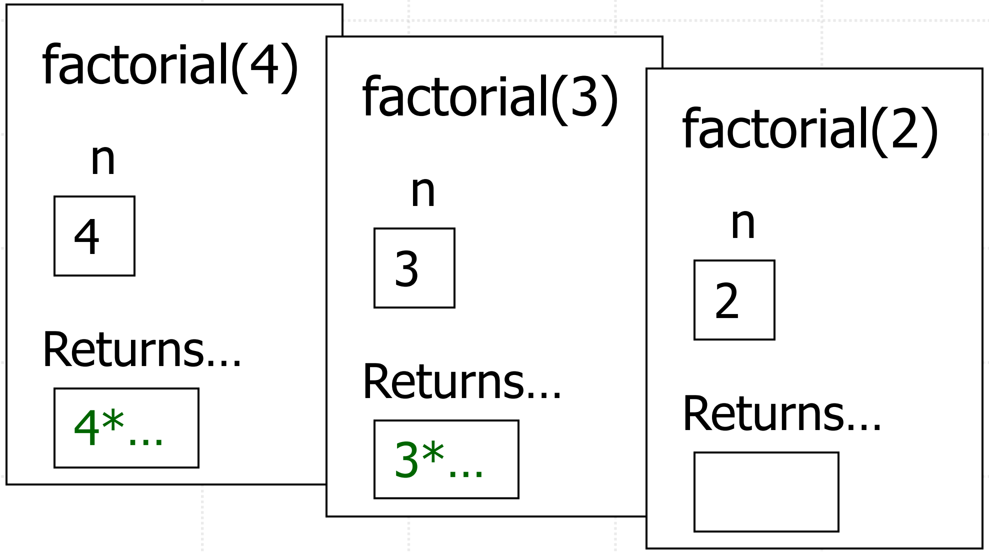
Recursive factorial – step by step

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



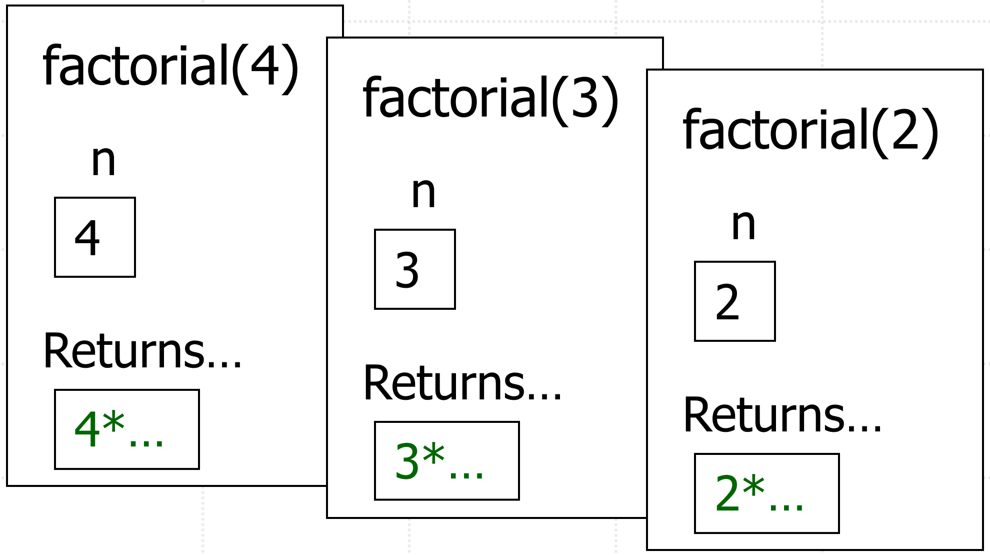
Recursive factorial – step by step

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



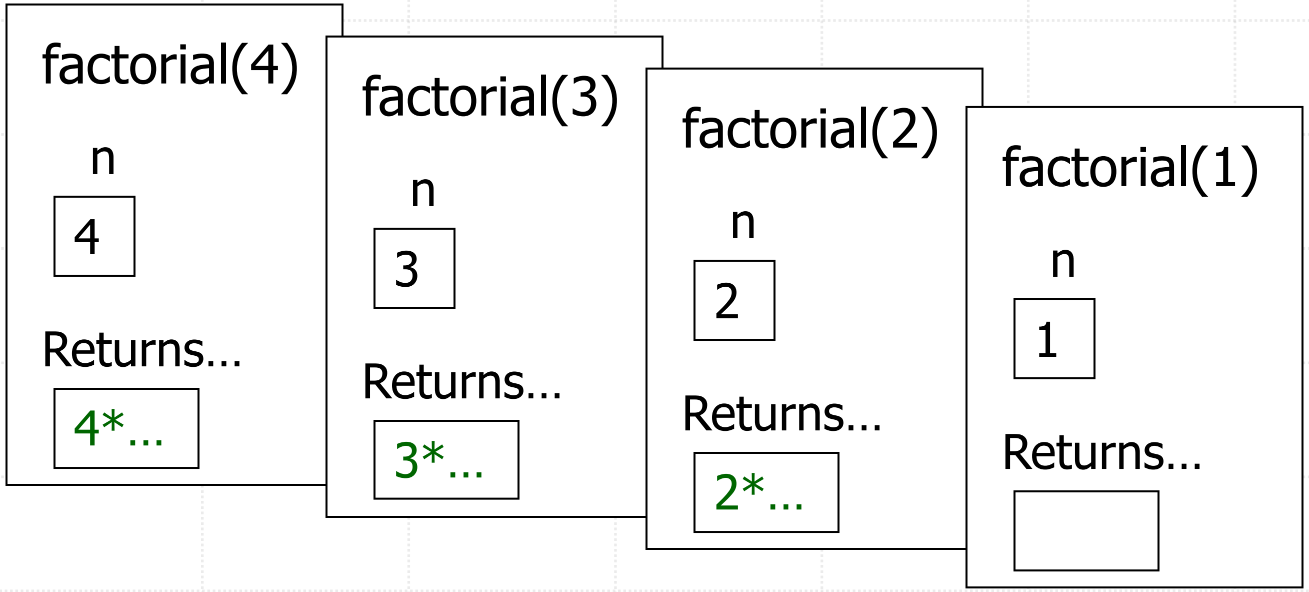
Recursive factorial – step by step

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



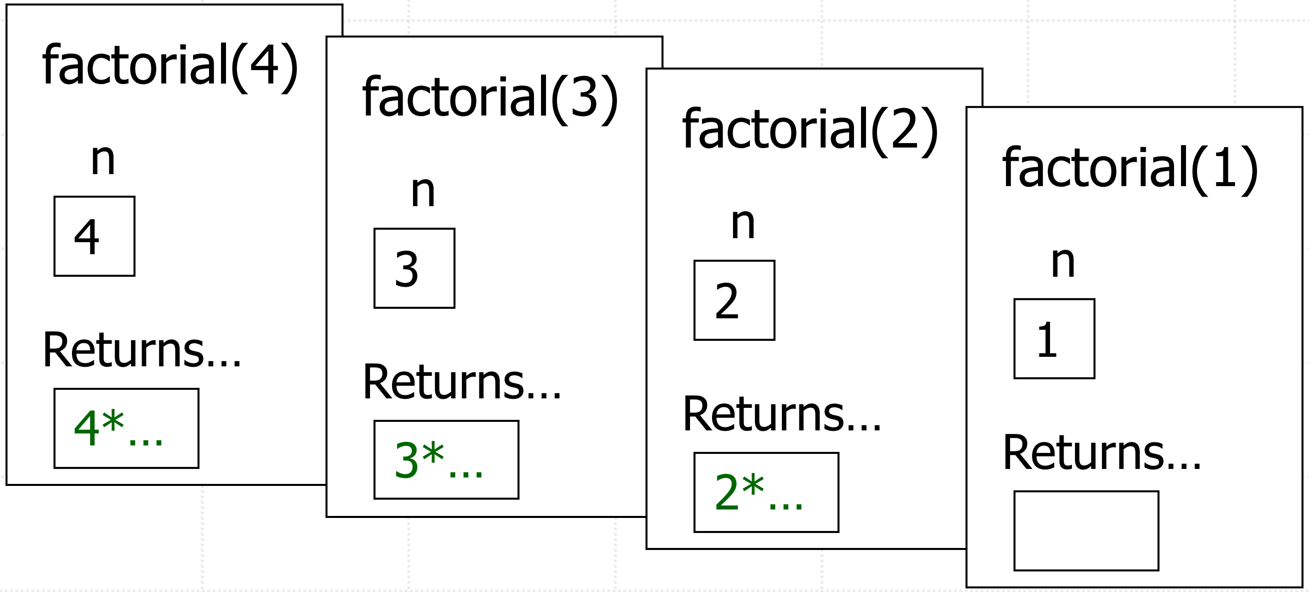
Recursive factorial – step by step

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



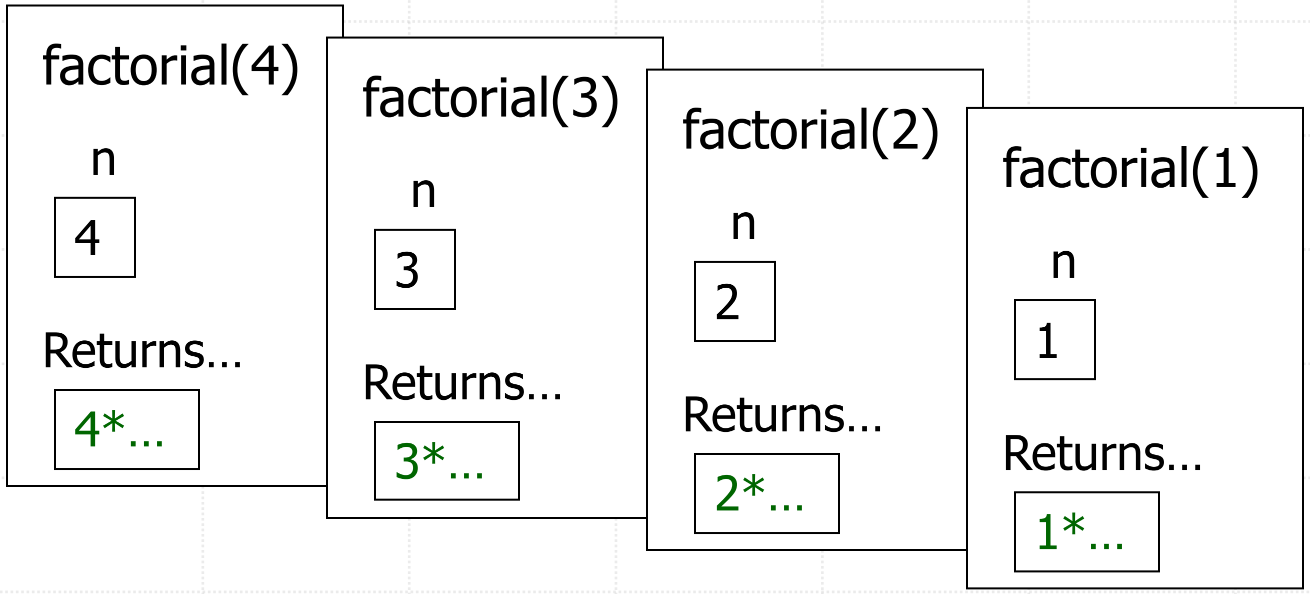
Recursive factorial – step by step

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



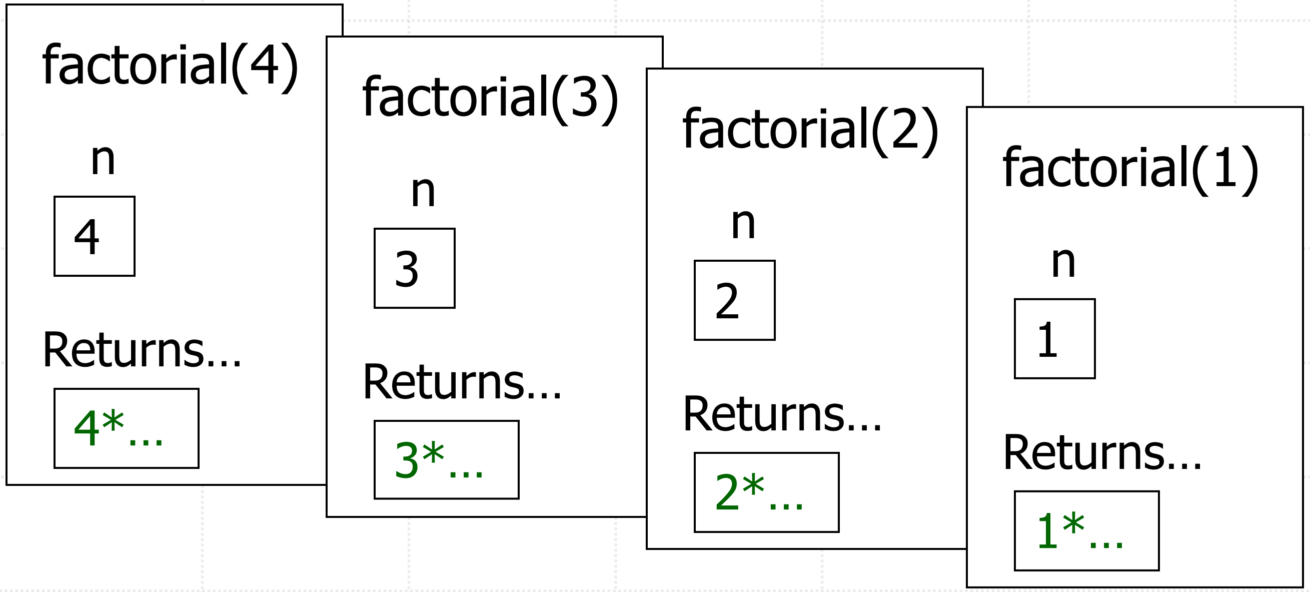
Recursive factorial – step by step

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



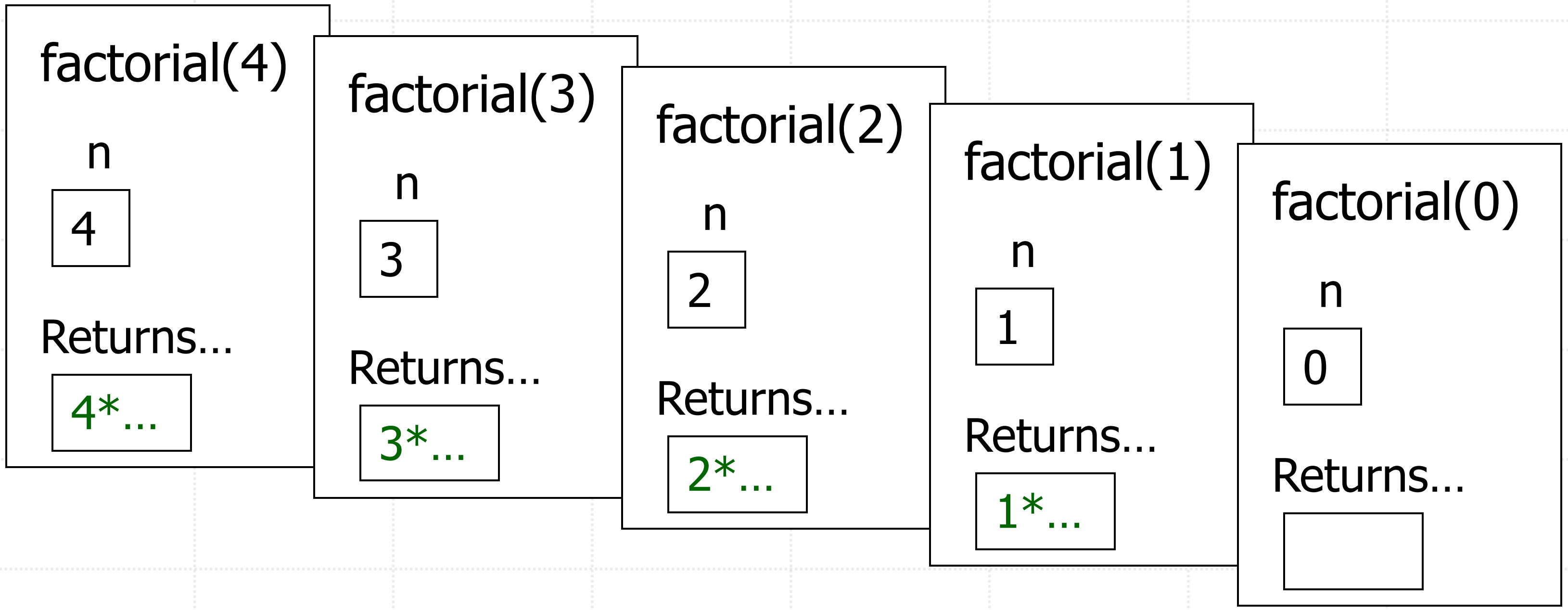
Recursive factorial – step by step

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



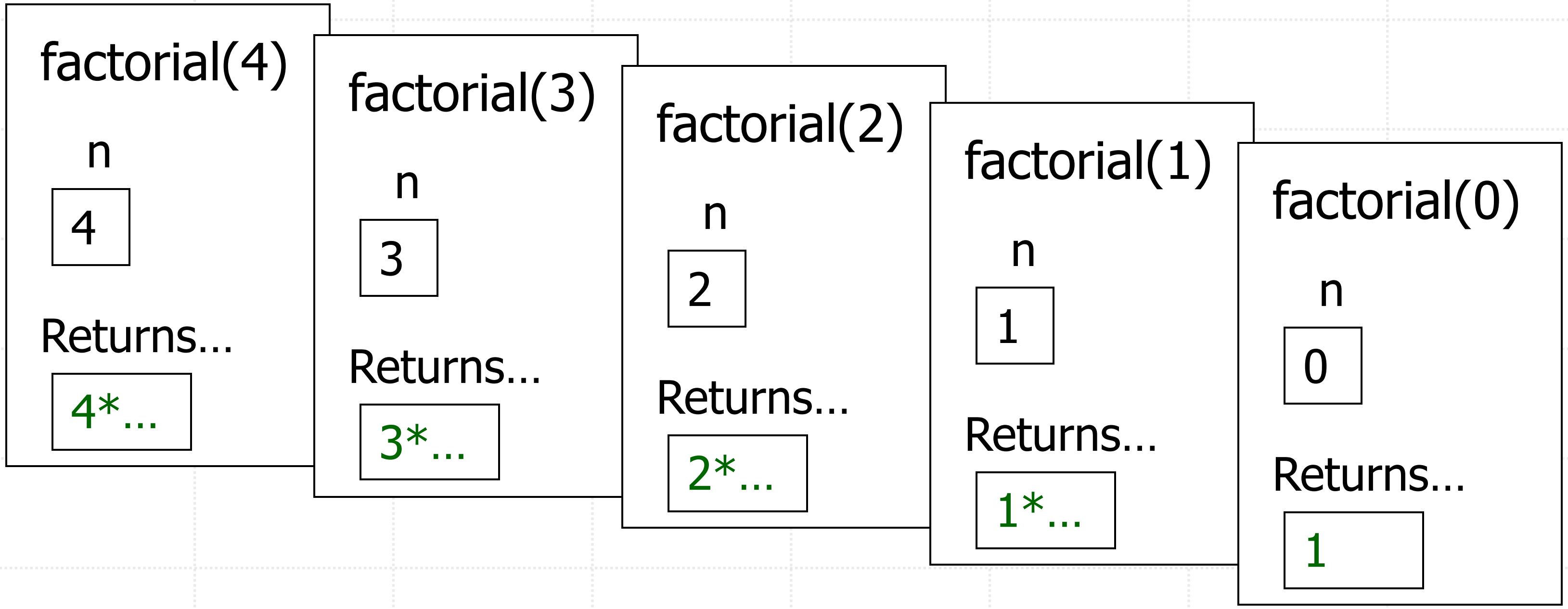
Recursive factorial – step by step

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



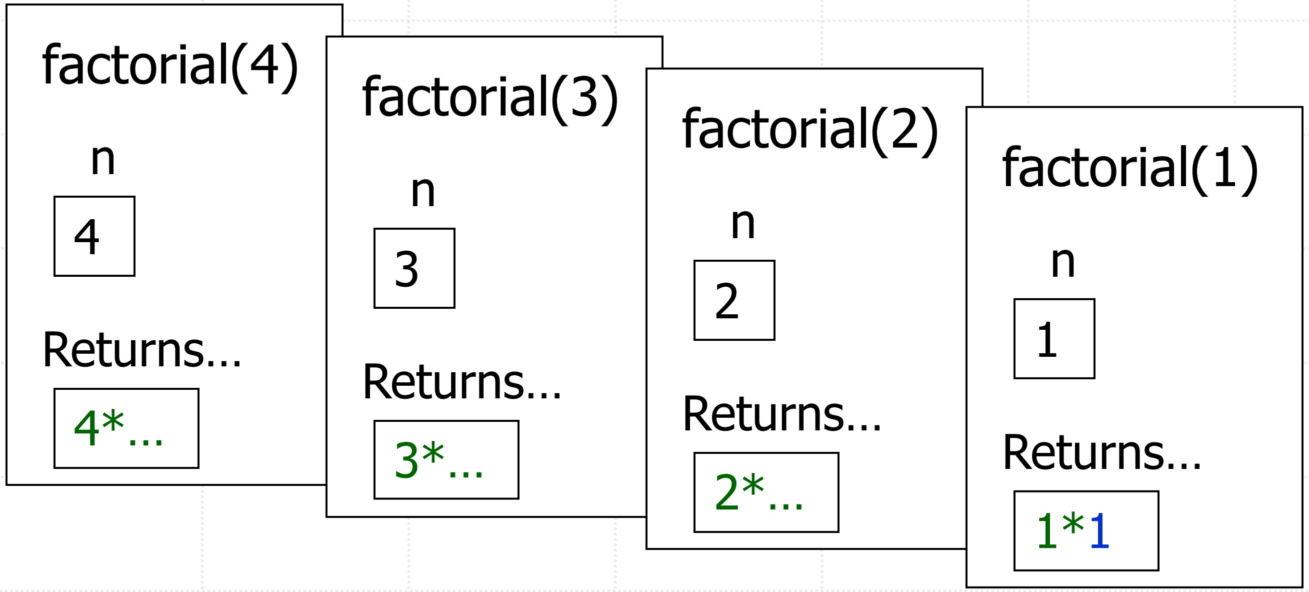
Recursive factorial – step by step

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



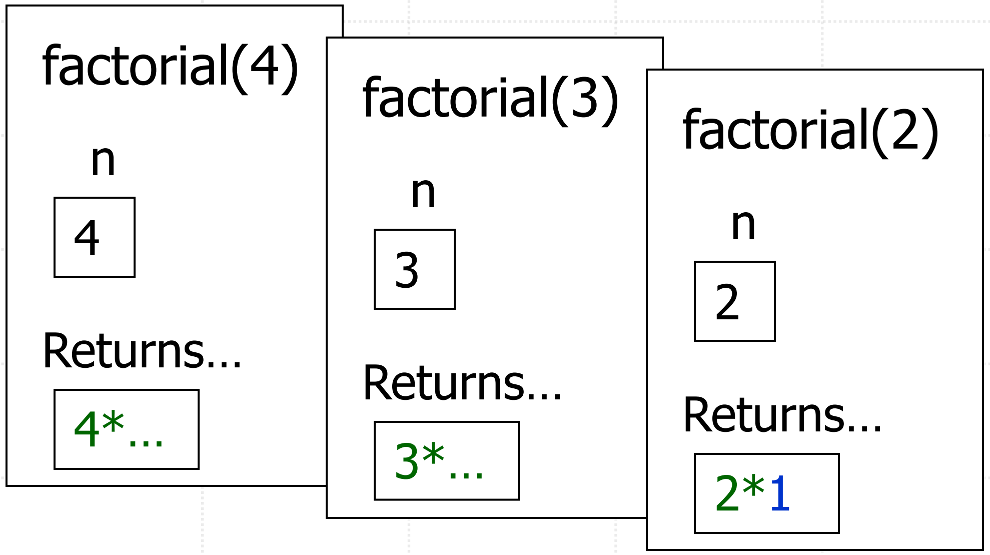
Recursive factorial – step by step

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



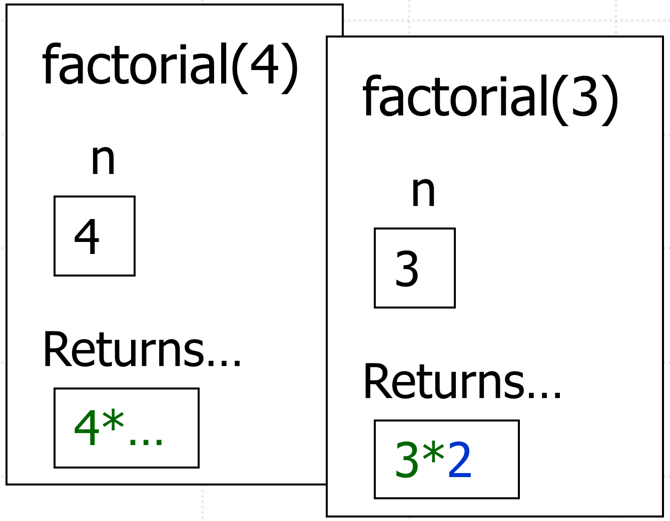
Recursive factorial – step by step

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



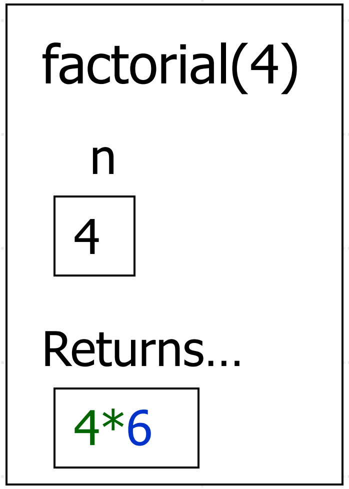
Recursive factorial – step by step

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



Recursive factorial – step by step

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



Homework

1. Create a Python function get 1 argument a returns 3 results(number+1, number*3 and (number*3)**number

Input:

```
>>> 5
```

Output:

```
>>>  (6, 15, 759375)
```

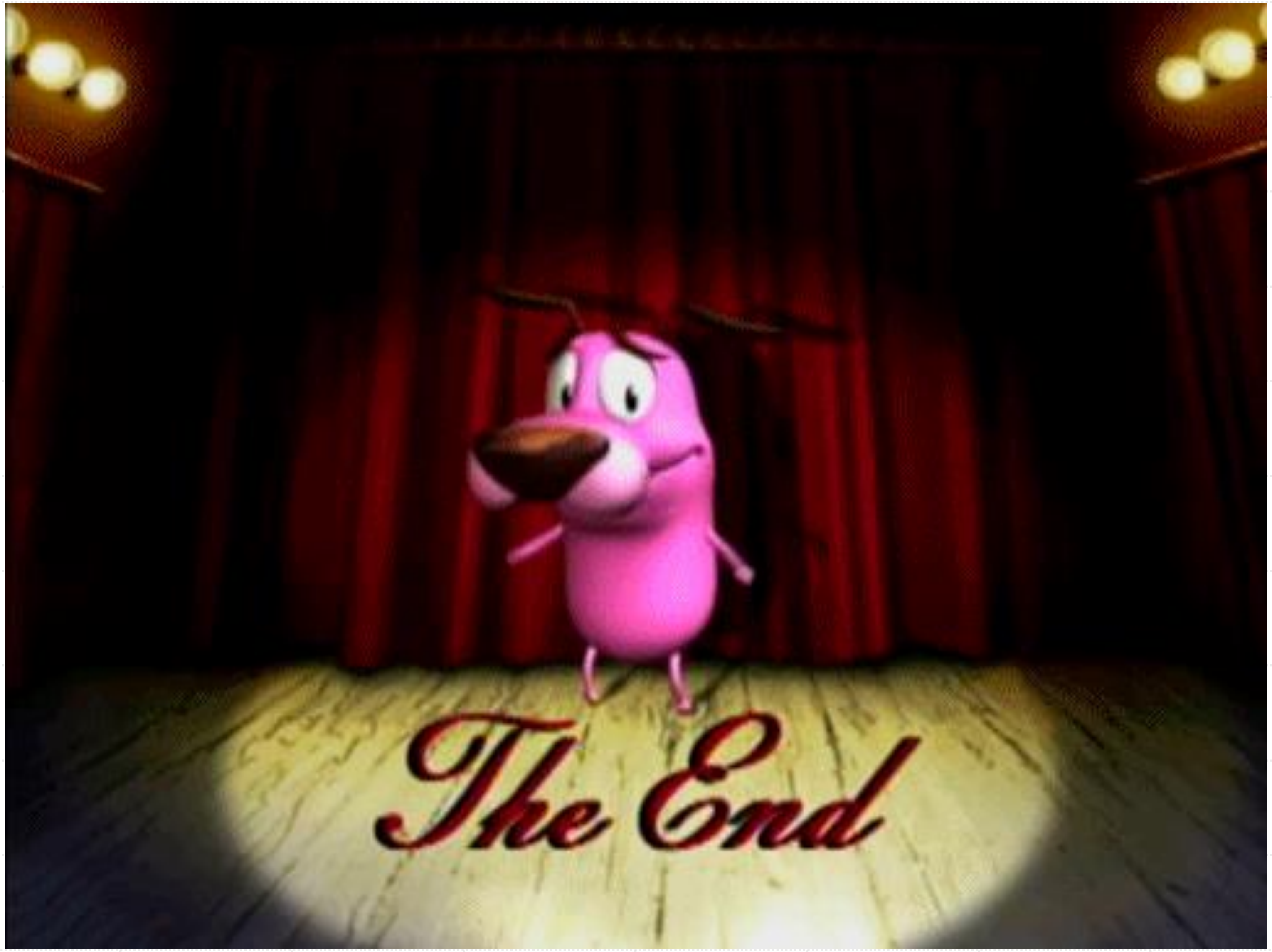
2. Create a Python function get amount of loan ,rate, a 2 durations,

and if the gap between the monthly payments is less than 200 and the

gap between the total returns are more than 1000, the first loan in better

Monthly payment Formula =round((amount*((1+rate)**duration_1))/(duration_1*12))

Total payment Formula =round(amount*((1+rate)**duration_1))


The End