Problem Solving and Programming CSE1001

Prof. Tulasi Prasad Sariki

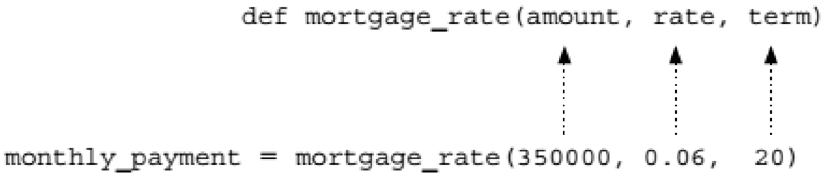
October 8, 2019

Functions

FUNCTIONS

Keyword Arguments in Python

- The functions we have looked at so far were called with a fixed number of positional arguments.
- **A positional argument** is an argument that is assigned to a particular parameter based on its position in the argument list, as illustrated below.

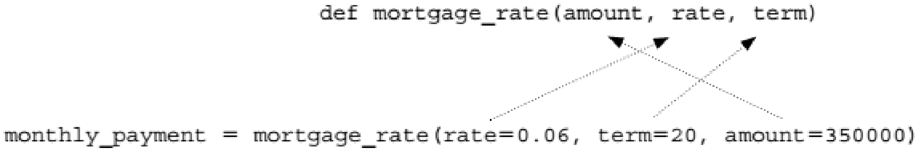


```
def mortgage_rate(amount, rate, term)

monthly_payment = mortgage_rate(350000, 0.06, 20)
```

Keyword Arguments in Python Contd...

- A **keyword argument** is an argument that is specified by parameter name, rather than as a positional argument as shown below

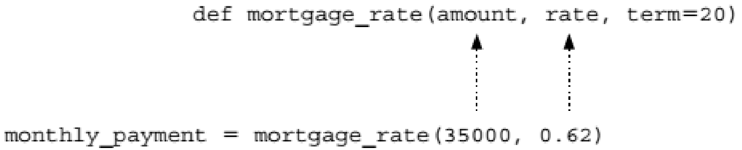


```
def mortgage_rate(amount, rate, term)

monthly_payment = mortgage_rate(rate=0.06, term=20, amount=350000)
```

Default Arguments in Python

- A **default argument** is an argument that can be optionally provided,



```
def mortgage_rate(amount, rate, term=20)

monthly_payment = mortgage_rate(35000, 0.62)
```

- In this case, the third argument in calls to function mortgage_rate is optional. If omitted, parameter term will be assigned by default to the value 20 (years) as shown. On the other hand, if a third argument is provided, the value passed will replace the default parameter value.

Functions

Keyword and Default Argument Examples

-
  ```
  >>> def f(a, b, c):
          print(a, b, c)
  ```

-
  ```
  >>> f(1, 2, 3)
   1 2 3
  ```

-
  ```
  >>> f(c=3, b=2, a=1)
   1 2 3
  ```

-
  ```
  >>> f(1, c=3, b=2)
  ```
- # a gets 1 by position, b and c passed by name 1 2 3

Default Examples

-
  ```
  >>> def f(a, b=2, c=3):
          print(a, b, c)
  ```
- # a required, b and c optional

-
  ```
  >>> f(1)              # Use defaults
  1 2 3
  ```

-
  ```
  >>> f(a=1)
  1 2 3
  ```

-
  ```
  >>> f(1, 4)           # Override defaults
  1 4 3
  ```

-
  ```
  >>> f(1, 4, 5)
  1 4 5
  ```

-
  ```
  >>> f(1, c=6)         # Choose defaults
  1 2 6
  ```

Combining keywords and defaults

- def func(spam, eggs, toast=0, ham=0):   # First 2 required
      print((spam, eggs, toast, ham))
- >>>func(1, 2)   # Output: (1, 2, 0, 0)
- >>>func(1, ham=1, eggs=0)   # Output: (1, 0, 0, 1)
- >>>func(spam=1, eggs=0)   # Output: (1, 0, 0, 0)
- >>>func(toast=1, eggs=2, spam=3)   # Output: (3, 2, 1, 0)
- >>> func(1, 2, 3, 4)   # Output: (1, 2, 3, 4)

Arbitrary Arguments Examples

- Use of '*'
- Collects unmatched positional arguments into a tuple:
  ```
  >>> def f(*args):
          print(args)
  ```

-
  ```
  >>> f()
  ()
  ```

-
  ```
  >>> f(1)
  (1,)
  ```

-
  ```
  >>> f(1, 2, 3, 4)
  (1, 2, 3, 4)
  ```

Arbitrary Arguments Examples

- ** feature is similar, but it only works for keyword arguments - it collects them into a new dictionary

-
  ```
  >>> def f(**args):
          print(args)
  ```

-
  ```
  >>> f()
  {}
  ```

-
  ```
  >>> f(a=1, b=2)
  {'a': 1, 'b': 2}
  ```

-
  ```
  >>> def f(a, *pargs, **kargs):
          print(a, pargs, kargs)
  ```

-
  ```
  >>> f(1, 2, 3, x=1, y=2)
  1 (2, 3) {'y': 2, 'x': 1}
  ```

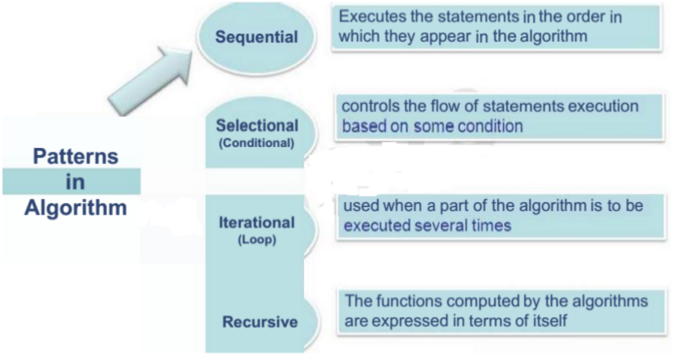
Calls: Unpacking arguments

- `>>> def func(a, b, c, d):`
  `        print(a, b, c, d)`
- `>>> args = (1, 2)`
- `>>> args += (3, 4)`
- `>>> func(*args)          # Same as func(1, 2, 3, 4)`
  `1 2 3 4`
- `>>> args = {'a': 1, 'b': 2, 'c': 3}`
- `>>> args['d'] = 4`
- `>>> func(**args)          # Same as func(a=1, b=2, c=3, d=4)`
  `1 2 3 4`

Function argument-matching forms

Syntax	Location	Interpretation
func(value)	Caller	Normal argument:Matched by position
func(name = value)	Caller	Normal argument:Matched by name
func(*iterable)	Caller	Pass all object in *iterable* as individual position argument
func(**dict)	Caller	Pass all key/value pairs in *dict* as individual keyword argument
def func(name)	function	Normal argument: matches any passed value by position or name
def func(name = value)	function	Default argument value, if not passed in call
def func(*name)	function	Matches and collects remaining positional arguments in tuple
def func(*other, name)	function	Arguments that must be passed by keywords only in calls(3 , X)
def func(*, name=value)	function	Arguments that must be passed by keywords only in calls(3 , X)

Different patterns in Algorithm


Patterns in Algorithm
Sequential
Executes the statements in the order in which they appear in the algorithm
Selectional (Conditional)
controls the flow of statements execution based on some condition
Iterational (Loop)
used when a part of the algorithm is to be executed several times
Recursive
The functions computed by the algorithms are expressed in terms of itself

Recursive Functions

MOTIVATION-Recursion

- Almost all computation involves the repetition of steps.
- Iterative control statements, such as the for and while statements, provide one means of controlling the repeated execution of instructions.
- Another way is by the use of recursion.

Recursive Algorithms

- In recursive problem solving, a problem is repeatedly broken down into similar sub problems, until the sub problems can be directly solved without further breakdown.
- The functions computed by the algorithms are expressed in terms of itself
- Example

  Task: Find the Factorial of a positive integer

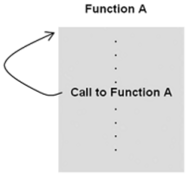
Recursive Algorithms

ALGORITHM:

```
Algorithm Factorial(n)
    Begin
      if (n==1) then return 1
      else return(n * Factorial(n-1))
    End
```

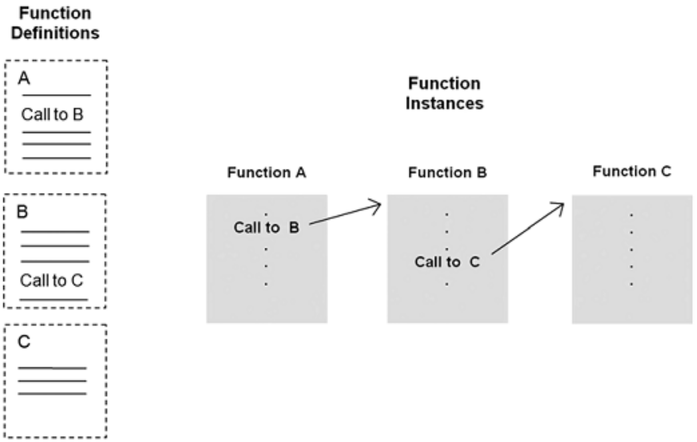
What Is a Recursive Function?

- A recursive function is often defined as "a function that calls itself."



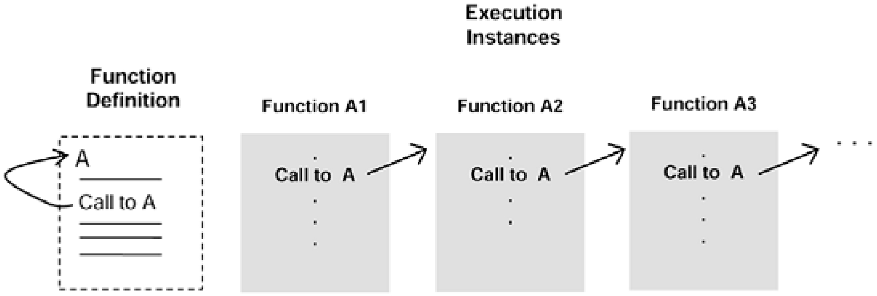
Recursive Functions

General mechanism of non-recursive function



Recursive function execution instances

- Execution of a series of recursive function instances is similar to the execution of series of non-recursive instances, except that the execution instances are "clones" of each other (that is, of the same function definition).



Recursive Functions

LET'S TRY IT!

-
  ```
  >>>def rfunc(n):
          print (n)
          if n>0:
             rfunc(n-1)
  ```
- >>> rfunc(4) → ???
- >>> rfunc(0) → ???
- >>> rfunc(100) → ???

-
  ```
  >>>def rfunc(n):
          if n==0:
             return 1
          else:
             return n * rfunc(n-1)
  ```
- >>> rfunc(1) → ???
- >>> rfunc(3) → ???
- >>> rfunc(100) → ???

Example: Factorial

Problem:

The factorial function is an often-used example of the use of recursion. The computation of the factorial of 4 is given as,

- factorial(4) = 4 * 3 * 2* 1= 24
- In general, the computation of the factorial of any (positive, nonzero) integer n is,
- factorial(n) = n . (n-1). (n-2) ... 1
- The one exception is the factorial of 0, defined to be 1.

Logic

- The factorial of n can be defined as n times the factorial of n - 1

$$\text{factorial}(n) = n \cdot \underbrace{(n-1) \cdot (n-2) \cdot \cdots \cdot 1}_{\text{factorial}(n-1)}$$

Thus, the complete definition of the factorial function is,

$$\begin{aligned} \text{factorial}(n) &= 1, && \text{if } n = 0 \\ &= n \cdot \text{factorial}(n-1), && \text{otherwise} \end{aligned}$$

Recursive Functions

A Recursive Factorial Function Implementation

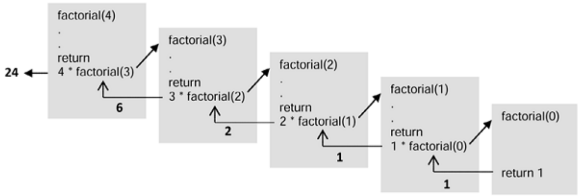
Factorial Function Implementation

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

Input

factorial(4)

Factorial Recursive Instance Calls



Factorial Recursive Instance Calls

LET'S TRY IT!

-
  ```
  >>>def factorial(n):
          if n==0:
              return 1
          return n * factorial(n-1)
  ```
- >>> factorial(4) → ???
- >>> factorial(0) → ???
- >>> factorial(100) → ???
- >>> factorial(10000) → ???

-
  ```
  >>> def ifactorial(n):
  result =1
  if n==0:
          return result
  for k in range(n,0,-1):
          result = result * k
  return result
  ```
- >>> ifactorial(0) → ???
- >>> ifactorial(100) → ???
- >>> ifactorial(10000) → ???

Recursive Functions

Command line arguments in Linux

```
command.py
import sys
print(len(sys.argv))
print(sys.argv[0])
```

In terminal:

```
python command.py 2 3 4
3
command.py
```

Recursive Functions

Exercise 1

An artificial rabbit population, satisfying the following conditions:
1. A newly born pair of rabbits, one male, one female, build the initial population.
2. These rabbits are able to mate at the age of one month so that at the end of its second month a female can bring forth another pair of rabbits.
3. These rabbits are immortal
4. A mating pair always produces one new pair (one male, one female) every month from the second month onwards.
Find the number of rabbit pairs after n-months

Recursive Functions

Exercise 2

Ram is planning to give chocolates to two of his brother. He comes to a shop that has 'n' chocolates of type 1 and 'm' chocolates of type 2, and Ram wants to buy largest but equal number of both. Write a program to determine the number using recursive function.

Thank you