

# Recursive Algorithms

## Recursive Functions

computing factorials recursively

computing factorials iteratively

## Accumulating Parameters

tracing recursive functions automatically

computing with accumulating parameters

## Recursive Problem Solving

check if a word is a palindrome

- 1 **Recursive Functions**
  - computing factorials recursively
  - computing factorials iteratively
- 2 **Accumulating Parameters**
  - tracing recursive functions automatically
  - computing with accumulating parameters
- 3 **Recursive Problem Solving**
  - check if a word is a palindrome

MCS 275 Lecture 8  
Programming Tools and File Management  
Jan Vershelde, 29 January 2010

# Recursive Algorithms

## Recursive Functions

computing factorials recursively

computing factorials iteratively

## Accumulating Parameters

tracing recursive functions automatically

computing with accumulating parameters

## Recursive Problem Solving

check if a word is a palindrome

- 1 Recursive Functions
  - computing factorials recursively
  - computing factorials iteratively
- 2 Accumulating Parameters
  - tracing recursive functions automatically
  - computing with accumulating parameters
- 3 Recursive Problem Solving
  - check if a word is a palindrome

# Computing Factorials Recursively

rule based programming

## Recursive Functions

computing factorials recursively

computing factorials iteratively

## Accumulating Parameters

tracing recursive functions automatically

computing with accumulating parameters

## Recursive Problem Solving

check if a word is a palindrome

Let  $n$  be a natural number.

By  $n!$  we denote *the factorial of  $n$* .

Its recursive definition is given by two rules:

- 1 for  $n \leq 1$ :  $n! = 1$
- 2 if we know the value for  $(n - 1)!$   
then  $n! = n \times (n - 1)!$

Recursion is similar to mathematical proof by induction:

- 1 first we verify the trivial or base case
- 2 assuming the statement holds for all values smaller than  $n$  – the induction hypothesis – we extend the proof to  $n$

# Computing Factorials Recursively

rule based programming

## Recursive Functions

computing factorials recursively

computing factorials iteratively

## Accumulating Parameters

tracing recursive functions automatically

computing with accumulating parameters

## Recursive Problem Solving

check if a word is a palindrome

Let  $n$  be a natural number.

By  $n!$  we denote *the factorial of  $n$* .

Its recursive definition is given by two rules:

- 1 for  $n \leq 1$ :  $n! = 1$
- 2 if we know the value for  $(n - 1)!$   
then  $n! = n \times (n - 1)!$

Recursion is similar to mathematical proof by induction:

- 1 first we verify the trivial or base case
- 2 assuming the statement holds for all values smaller than  $n$  – the induction hypothesis – we extend the proof to  $n$

# Computing Factorials Recursively

rule based programming

## Recursive Functions

computing factorials recursively

computing factorials iteratively

## Accumulating Parameters

tracing recursive functions automatically

computing with accumulating parameters

## Recursive Problem Solving

check if a word is a palindrome

Let  $n$  be a natural number.

By  $n!$  we denote *the factorial of  $n$* .

Its recursive definition is given by two rules:

- 1 for  $n \leq 1$ :  $n! = 1$
- 2 if we know the value for  $(n - 1)!$  then  $n! = n \times (n - 1)!$

Recursion is similar to mathematical proof by induction:

- 1 first we verify the trivial or base case
- 2 assuming the statement holds for all values smaller than  $n$  – the induction hypothesis – we extend the proof to  $n$

# Computing Factorials Recursively

rule based programming

## Recursive Functions

computing factorials recursively

computing factorials iteratively

## Accumulating Parameters

tracing recursive functions automatically

computing with accumulating parameters

## Recursive Problem Solving

check if a word is a palindrome

Let  $n$  be a natural number.

By  $n!$  we denote *the factorial of  $n$* .

Its recursive definition is given by two rules:

- 1 for  $n \leq 1$ :  $n! = 1$
- 2 if we know the value for  $(n - 1)!$   
then  $n! = n \times (n - 1)!$

Recursion is similar to mathematical proof by induction:

- 1 first we verify the trivial or base case
- 2 assuming the statement holds for all values smaller than  $n$  – the induction hypothesis – we extend the proof to  $n$

# Computing Factorials Recursively

rule based programming

## Recursive Functions

computing factorials recursively

computing factorials iteratively

## Accumulating Parameters

tracing recursive functions automatically

computing with accumulating parameters

## Recursive Problem Solving

check if a word is a palindrome

Let  $n$  be a natural number.

By  $n!$  we denote *the factorial of  $n$* .

Its recursive definition is given by two rules:

- 1 for  $n \leq 1$ :  $n! = 1$
- 2 if we know the value for  $(n - 1)!$   
then  $n! = n \times (n - 1)!$

Recursion is similar to mathematical proof by induction:

- 1 first we verify the trivial or base case
- 2 assuming the statement holds for all values smaller than  $n$  – the induction hypothesis – we extend the proof to  $n$

# The Function `factorial`

recursive programming in Python

## Recursive Functions

computing factorials recursively

computing factorials iteratively

## Accumulating Parameters

tracing recursive functions automatically

computing with accumulating parameters

## Recursive Problem Solving

check if a word is a palindrome

```
def factorial(n):
    """
    computes the factorial of n recursively
    """
    if n <= 1:
        return 1
    else:
        return n*factorial(n-1)

def main():
    n = input('give a number : ')
    f = factorial(n)
    print 'n! = ', f
    print 'len(n!) = ', len(str(f))
```

# The Function `factorial`

recursive programming in Python

## Recursive Functions

computing factorials recursively

computing factorials iteratively

## Accumulating Parameters

tracing recursive functions automatically

computing with accumulating parameters

## Recursive Problem Solving

check if a word is a palindrome

```
def factorial(n):
    """
    computes the factorial of n recursively
    """
    if n <= 1:
        return 1
    else:
        return n*factorial(n-1)

def main():
    n = input('give a number : ')
    f = factorial(n)
    print 'n! = ', f
    print 'len(n!) = ', len(str(f))
```

# The Function `factorial`

recursive programming in Python

## Recursive Functions

computing factorials recursively

computing factorials iteratively

## Accumulating Parameters

tracing recursive functions automatically

computing with accumulating parameters

## Recursive Problem Solving

check if a word is a palindrome

```
def factorial(n):
    """
    computes the factorial of n recursively
    """
    if n <= 1:
        return 1
    else:
        return n*factorial(n-1)

def main():
    n = input('give a number : ')
    f = factorial(n)
    print 'n! = ', f
    print 'len(n!) = ', len(str(f))
```

# Tracing Recursive Functions

call #, input and output parameters

## Recursive Functions

computing factorials recursively

computing factorials iteratively

## Accumulating Parameters

tracing recursive functions automatically

computing with accumulating parameters

## Recursive Problem Solving

check if a word is a palindrome

Calling factorial for  $n = 5$ :

```
factorial(5) call #0: call for n-1 = 4
  factorial(4) call #1: call for n-1 = 3
    factorial(3) call #2: call for n-1 = 2
      factorial(2) call #3: call for n-1 = 1
        factorial(1) call #4: base case, return 1
      factorial(2) call #3: returning 2
    factorial(3) call #2: returning 6
  factorial(4) call #1: returning 24
factorial(5) call #0: returning 120
```

Computes in returns:

```
return 1, 1*2, 1*2*3, 1*2*3*4, 1*2*3*4*5
```

# Tracing Recursive Functions

call #, input and output parameters

## Recursive Functions

computing factorials recursively

computing factorials iteratively

## Accumulating Parameters

tracing recursive functions automatically

computing with accumulating parameters

## Recursive Problem Solving

check if a word is a palindrome

Calling factorial for  $n = 5$ :

```
factorial(5) call #0: call for n-1 = 4
factorial(4) call #1: call for n-1 = 3
factorial(3) call #2: call for n-1 = 2
factorial(2) call #3: call for n-1 = 1
factorial(1) call #4: base case, return 1
factorial(2) call #3: returning 2
factorial(3) call #2: returning 6
factorial(4) call #1: returning 24
factorial(5) call #0: returning 120
```

Computes in returns:

```
return 1, 1*2, 1*2*3, 1*2*3*4, 1*2*3*4*5
```

# Tracing Recursive Functions

call #, input and output parameters

## Recursive Functions

computing factorials recursively

computing factorials iteratively

## Accumulating Parameters

tracing recursive functions automatically

computing with accumulating parameters

## Recursive Problem Solving

check if a word is a palindrome

Calling factorial for  $n = 5$ :

```
factorial(5) call #0: call for n-1 = 4
factorial(4) call #1: call for n-1 = 3
factorial(3) call #2: call for n-1 = 2
factorial(2) call #3: call for n-1 = 1
factorial(1) call #4: base case, return 1
factorial(2) call #3: returning 2
factorial(3) call #2: returning 6
factorial(4) call #1: returning 24
factorial(5) call #0: returning 120
```

Computes in returns:

```
return 1, 1*2, 1*2*3, 1*2*3*4, 1*2*3*4*5
```

# Tracing Recursive Functions

call #, input and output parameters

## Recursive Functions

computing factorials recursively

computing factorials iteratively

## Accumulating Parameters

tracing recursive functions automatically

computing with accumulating parameters

## Recursive Problem Solving

check if a word is a palindrome

Calling factorial for  $n = 5$ :

```
factorial(5) call #0: call for n-1 = 4
factorial(4) call #1: call for n-1 = 3
factorial(3) call #2: call for n-1 = 2
factorial(2) call #3: call for n-1 = 1
factorial(1) call #4: base case, return 1
factorial(2) call #3: returning 2
factorial(3) call #2: returning 6
factorial(4) call #1: returning 24
factorial(5) call #0: returning 120
```

Computes in returns:

```
return 1, 1*2, 1*2*3, 1*2*3*4, 1*2*3*4*5
```

# Tracing Recursive Functions

call #, input and output parameters

## Recursive Functions

computing factorials recursively

computing factorials iteratively

## Accumulating Parameters

tracing recursive functions automatically

computing with accumulating parameters

## Recursive Problem Solving

check if a word is a palindrome

Calling factorial for  $n = 5$ :

```
factorial(5) call #0: call for n-1 = 4
factorial(4) call #1: call for n-1 = 3
factorial(3) call #2: call for n-1 = 2
factorial(2) call #3: call for n-1 = 1
factorial(1) call #4: base case, return 1
factorial(2) call #3: returning 2
factorial(3) call #2: returning 6
factorial(4) call #1: returning 24
factorial(5) call #0: returning 120
```

Computes in returns:

```
return 1, 1*2, 1*2*3, 1*2*3*4, 1*2*3*4*5
```

# Tracing Recursive Functions

call #, input and output parameters

## Recursive Functions

computing factorials recursively

computing factorials iteratively

## Accumulating Parameters

tracing recursive functions automatically

computing with accumulating parameters

## Recursive Problem Solving

check if a word is a palindrome

Calling factorial for  $n = 5$ :

```
factorial(5) call #0: call for n-1 = 4
  factorial(4) call #1: call for n-1 = 3
    factorial(3) call #2: call for n-1 = 2
      factorial(2) call #3: call for n-1 = 1
        factorial(1) call #4: base case, return 1
      factorial(2) call #3: returning 2
    factorial(3) call #2: returning 6
  factorial(4) call #1: returning 24
factorial(5) call #0: returning 120
```

Computes in returns:

```
return 1, 1*2, 1*2*3, 1*2*3*4, 1*2*3*4*5
```

# Tracing Recursive Functions

call #, input and output parameters

## Recursive Functions

computing factorials recursively

computing factorials iteratively

## Accumulating Parameters

tracing recursive functions automatically

computing with accumulating parameters

## Recursive Problem Solving

check if a word is a palindrome

Calling factorial for  $n = 5$ :

```
factorial(5) call #0: call for n-1 = 4
factorial(4) call #1: call for n-1 = 3
factorial(3) call #2: call for n-1 = 2
factorial(2) call #3: call for n-1 = 1
factorial(1) call #4: base case, return 1
factorial(2) call #3: returning 2
factorial(3) call #2: returning 6
factorial(4) call #1: returning 24
factorial(5) call #0: returning 120
```

Computes in returns:

```
return 1, 1*2, 1*2*3, 1*2*3*4, 1*2*3*4*5
```

# Tracing Recursive Functions

call #, input and output parameters

## Recursive Functions

computing factorials recursively

computing factorials iteratively

## Accumulating Parameters

tracing recursive functions automatically

computing with accumulating parameters

## Recursive Problem Solving

check if a word is a palindrome

Calling factorial for  $n = 5$ :

```
factorial(5) call #0: call for n-1 = 4
  factorial(4) call #1: call for n-1 = 3
    factorial(3) call #2: call for n-1 = 2
      factorial(2) call #3: call for n-1 = 1
        factorial(1) call #4: base case, return 1
      factorial(2) call #3: returning 2
    factorial(3) call #2: returning 6
  factorial(4) call #1: returning 24
factorial(5) call #0: returning 120
```

Computes in returns:

```
return 1, 1*2, 1*2*3, 1*2*3*4, 1*2*3*4*5
```

# Tracing Recursive Functions

call #, input and output parameters

## Recursive Functions

computing factorials recursively

computing factorials iteratively

## Accumulating Parameters

tracing recursive functions automatically

computing with accumulating parameters

## Recursive Problem Solving

check if a word is a palindrome

Calling factorial for  $n = 5$ :

```
factorial(5) call #0: call for n-1 = 4
factorial(4) call #1: call for n-1 = 3
factorial(3) call #2: call for n-1 = 2
factorial(2) call #3: call for n-1 = 1
factorial(1) call #4: base case, return 1
factorial(2) call #3: returning 2
factorial(3) call #2: returning 6
factorial(4) call #1: returning 24
factorial(5) call #0: returning 120
```

Computes in returns:

```
return 1, 1*2, 1*2*3, 1*2*3*4, 1*2*3*4*5
```

# Tracing Recursive Functions

call #, input and output parameters

## Recursive Functions

computing factorials recursively

computing factorials iteratively

## Accumulating Parameters

tracing recursive functions automatically

computing with accumulating parameters

## Recursive Problem Solving

check if a word is a palindrome

Calling factorial for  $n = 5$ :

```
factorial(5) call #0: call for n-1 = 4
factorial(4) call #1: call for n-1 = 3
factorial(3) call #2: call for n-1 = 2
factorial(2) call #3: call for n-1 = 1
factorial(1) call #4: base case, return 1
factorial(2) call #3: returning 2
factorial(3) call #2: returning 6
factorial(4) call #1: returning 24
factorial(5) call #0: returning 120
```

Computes in returns:

return 1, 1\*2, 1\*2\*3, 1\*2\*3\*4, 1\*2\*3\*4\*5

# Recursive Algorithms

## Recursive Functions

computing factorials recursively

computing factorials iteratively

## Accumulating Parameters

tracing recursive functions automatically

computing with accumulating parameters

## Recursive Problem Solving

check if a word is a palindrome

### 1 Recursive Functions

computing factorials recursively

computing factorials iteratively

### 2 Accumulating Parameters

tracing recursive functions automatically

computing with accumulating parameters

### 3 Recursive Problem Solving

check if a word is a palindrome

# Running the Recursive factorial

long integers are no problem, but ...

## Recursive Functions

computing factorials recursively

computing factorials iteratively

## Accumulating Parameters

tracing recursive functions automatically

computing with accumulating parameters

## Recursive Problem Solving

check if a word is a palindrome

```
$ python factorial.py
give a number : 79
n! = 894618213078297528685144171539831652
069808216779571907213868063227837990693501
860533361810841010176000000000000000000
len(n!) = 117
```

Exploiting Python long integers:

```
$ python factorial.py
give a number : 1234
...
RuntimeError: maximum recursion depth exceeded
```

An exception handler will compute  $n!$  iteratively.

# Running the Recursive factorial

long integers are no problem, but ...

## Recursive Functions

computing factorials recursively

computing factorials iteratively

## Accumulating Parameters

tracing recursive functions automatically

computing with accumulating parameters

## Recursive Problem Solving

check if a word is a palindrome

```
$ python factorial.py
give a number : 79
n! = 894618213078297528685144171539831652
069808216779571907213868063227837990693501
86053336181084101017600000000000000000000
len(n!) = 117
```

Exploiting Python long integers:

```
$ python factorial.py
give a number : 1234
...
RuntimeError: maximum recursion depth exceeded
```

An exception handler will compute  $n!$  iteratively.

# Running the Recursive factorial

long integers are no problem, but ...

## Recursive Functions

computing factorials recursively

computing factorials iteratively

## Accumulating Parameters

tracing recursive functions automatically

computing with accumulating parameters

## Recursive Problem Solving

check if a word is a palindrome

```
$ python factorial.py
give a number : 79
n! = 894618213078297528685144171539831652
069808216779571907213868063227837990693501
860533361810841010176000000000000000000
len(n!) = 117
```

Exploiting Python long integers:

```
$ python factorial.py
give a number : 1234
...
RuntimeError: maximum recursion depth exceeded
```

An exception handler will compute  $n!$  iteratively.

# Stack of Function Calls

## Recursive Functions

computing factorials recursively  
computing factorials iteratively

## Accumulating Parameters

tracing recursive functions automatically  
computing with accumulating parameters

## Recursive Problem Solving

check if a word is a palindrome

The execution of recursive functions requires a stack of function calls.

For example, for  $n = 5$ , the stack grows like

```
factorial(1) call #4: base case, return 1
factorial(2) call #3: returning 2
factorial(3) call #2: returning 6
factorial(4) call #1: returning 24
factorial(5) call #0: returning 120
```

New function calls are pushed on the stack.  
Upon return, a function call is popped off the stack.

# Stack of Function Calls

## Recursive Functions

computing factorials recursively  
computing factorials iteratively

## Accumulating Parameters

tracing recursive functions automatically  
computing with accumulating parameters

## Recursive Problem Solving

check if a word is a palindrome

The execution of recursive functions requires a stack of function calls.

For example, for  $n = 5$ , the stack grows like

```
factorial(1) call #4: base case, return 1
factorial(2) call #3: returning 2
factorial(3) call #2: returning 6
factorial(4) call #1: returning 24
factorial(5) call #0: returning 120
```

New function calls are pushed on the stack.  
Upon return, a function call is popped off the stack.

# Stack of Function Calls

## Recursive Functions

computing factorials recursively  
computing factorials iteratively

## Accumulating Parameters

tracing recursive functions automatically  
computing with accumulating parameters

## Recursive Problem Solving

check if a word is a palindrome

The execution of recursive functions requires a stack of function calls.

For example, for  $n = 5$ , the stack grows like

```
factorial(1) call #4: base case, return 1
factorial(2) call #3: returning 2
factorial(3) call #2: returning 6
factorial(4) call #1: returning 24
factorial(5) call #0: returning 120
```

New function calls are pushed on the stack.  
Upon return, a function call is popped off the stack.

# Computing Factorials Iteratively

in an exception handler

## Recursive Functions

computing factorials recursively

computing factorials iteratively

## Accumulating Parameters

tracing recursive functions automatically

computing with accumulating parameters

## Recursive Problem Solving

check if a word is a palindrome

```
def factexcept(n):
    """
    when the recursion depth is exceeded
    the factorial of n is computed iteratively
    """
    if n <= 1:
        return 1
    else:
        try:
            return n*factexcept(n-1)
        except RuntimeError:
            f = 1
            for i in range(2,n+1): f = f*i
            return f
```

# Computing Factorials Iteratively

in an exception handler

## Recursive Functions

computing factorials recursively

computing factorials iteratively

## Accumulating Parameters

tracing recursive functions automatically

computing with accumulating parameters

## Recursive Problem Solving

check if a word is a palindrome

```
def factexcept(n):
    """
    when the recursion depth is exceeded
    the factorial of n is computed iteratively
    """
    if n <= 1:
        return 1
    else:
        try:
            return n*factexcept(n-1)
        except RuntimeError:
            f = 1
            for i in range(2,n+1): f = f*i
            return f
```

# Computing Factorials Iteratively

in an exception handler

## Recursive Functions

computing factorials recursively

computing factorials iteratively

## Accumulating Parameters

tracing recursive functions automatically

computing with accumulating parameters

## Recursive Problem Solving

check if a word is a palindrome

```
def factexcept(n):
    """
    when the recursion depth is exceeded
    the factorial of n is computed iteratively
    """
    if n <= 1:
        return 1
    else:
        try:
            return n*factexcept(n-1)
        except RuntimeError:
            f = 1
            for i in range(2,n+1): f = f*i
            return f
```

# Recursive Algorithms

## Recursive Functions

computing factorials recursively

computing factorials iteratively

## Accumulating Parameters

tracing recursive functions automatically

computing with accumulating parameters

## Recursive Problem Solving

check if a word is a palindrome

### 1 Recursive Functions

computing factorials recursively

computing factorials iteratively

### 2 Accumulating Parameters

tracing recursive functions automatically

computing with accumulating parameters

### 3 Recursive Problem Solving

check if a word is a palindrome

# Tracing Recursive Functions

## Recursive Functions

computing factorials recursively

computing factorials iteratively

## Accumulating Parameters

tracing recursive functions automatically

computing with accumulating parameters

## Recursive Problem Solving

check if a word is a palindrome

Tracing the execution of a recursive function means: displaying for each function call:

- 1 the value for the input parameters
- 2 what is computed inside the function
- 3 the return value of the function

→ we need the number of each function call

Use an *accumulating parameter*  $k$ :

```
def factotrace(n,k):
```

increment  $k$  with each recursive call:

```
    return factotrace(n-1,k+1)
```

# Tracing Recursive Functions

## Recursive Functions

computing factorials recursively

computing factorials iteratively

## Accumulating Parameters

tracing recursive functions automatically

computing with accumulating parameters

## Recursive Problem Solving

check if a word is a palindrome

Tracing the execution of a recursive function means: displaying for each function call:

- 1 the value for the input parameters
- 2 what is computed inside the function
- 3 the return value of the function

→ we need the number of each function call

Use an *accumulating parameter*  $k$ :

```
def factotrace(n,k):
```

increment  $k$  with each recursive call:

```
    return factotrace(n-1,k+1)
```

# Tracing Recursive Functions

## Recursive Functions

computing factorials recursively

computing factorials iteratively

## Accumulating Parameters

tracing recursive functions automatically

computing with accumulating parameters

## Recursive Problem Solving

check if a word is a palindrome

Tracing the execution of a recursive function means: displaying for each function call:

- 1 the value for the input parameters
- 2 what is computed inside the function
- 3 the return value of the function

→ we need the number of each function call

Use an *accumulating parameter*  $k$ :

```
def factotrace(n,k):
```

increment  $k$  with each recursive call:

```
    return factotrace(n-1,k+1)
```

# Tracing Recursive Functions

## Recursive Functions

computing factorials recursively

computing factorials iteratively

## Accumulating Parameters

tracing recursive functions automatically

computing with accumulating parameters

## Recursive Problem Solving

check if a word is a palindrome

Tracing the execution of a recursive function means: displaying for each function call:

- 1 the value for the input parameters
- 2 what is computed inside the function
- 3 the return value of the function

→ we need the number of each function call

Use an *accumulating parameter*  $k$ :

```
def factotrace(n,k):
```

increment  $k$  with each recursive call:

```
    return factotrace(n-1,k+1)
```

# Tracing Recursive Functions

## Recursive Functions

computing factorials recursively

computing factorials iteratively

## Accumulating Parameters

tracing recursive functions automatically

computing with accumulating parameters

## Recursive Problem Solving

check if a word is a palindrome

Tracing the execution of a recursive function means: displaying for each function call:

- 1 the value for the input parameters
- 2 what is computed inside the function
- 3 the return value of the function

→ we need the number of each function call

Use an *accumulating parameter*  $k$ :

```
def factotrace(n,k):
```

increment  $k$  with each recursive call:

```
    return factotrace(n-1,k+1)
```

# Running factotrace.py

automatic trace of recursive factorial

## Recursive Functions

computing factorials recursively

computing factorials iteratively

## Accumulating Parameters

tracing recursive functions automatically

computing with accumulating parameters

## Recursive Problem Solving

check if a word is a palindrome

Call factotrace for  $n = 5$  and  $k = 0$ :

```
factotrace(5,0): call for n-1 = 4
  factotrace(4,1): call for n-1 = 3
    factotrace(3,2): call for n-1 = 2
      factotrace(2,3): call for n-1 = 1
        factotrace(1,4): base case, return 1
          factotrace(2,3): returning 2
            factotrace(3,2): returning 6
              factotrace(4,1): returning 24
                factotrace(5,0): returning 120
```

At call  $k$ , we indent with  $k$  spaces.

# The Function factotrace.py

## Recursive Functions

computing factorials recursively

computing factorials iteratively

## Accumulating Parameters

tracing recursive functions automatically

computing with accumulating parameters

## Recursive Problem Solving

check if a word is a palindrome

```
def factotrace(n,k):
    """
    prints out trace information in call k
    the initial value for k should be zero
    """
    s = k*' '
    s = s + 'factotrace(%d,%d):' % (n,k)
    if n <= 1:
        print s + ' base case, return 1'
        return 1
    else:
        print s + ' call for n-1 = ' + str(n-1)
        y = n*factotrace(n-1,k+1)
        print s + ' returning %d' % y
        return y
```

# The Function factotrace.py

## Recursive Functions

computing factorials recursively  
computing factorials iteratively

## Accumulating Parameters

tracing recursive functions automatically  
computing with accumulating parameters

## Recursive Problem Solving

check if a word is a palindrome

```
def factotrace(n,k):
    """
    prints out trace information in call k
    the initial value for k should be zero
    """
    s = k*' '
    s = s + 'factotrace(%d,%d):' % (n,k)
    if n <= 1:
        print s + ' base case, return 1'
        return 1
    else:
        print s + ' call for n-1 = ' + str(n-1)
        y = n*factotrace(n-1,k+1)
        print s + ' returning %d' % y
        return y
```

# The Function factotrace.py

## Recursive Functions

computing factorials recursively  
computing factorials iteratively

## Accumulating Parameters

tracing recursive functions automatically  
computing with accumulating parameters

## Recursive Problem Solving

check if a word is a palindrome

```
def factotrace(n,k):
    """
    prints out trace information in call k
    the initial value for k should be zero
    """
    s = k*' '
    s = s + 'factotrace(%d,%d):' % (n,k)
    if n <= 1:
        print s + ' base case, return 1'
        return 1
    else:
        print s + ' call for n-1 = ' + str(n-1)
        y = n*factotrace(n-1,k+1)
        print s + ' returning %d' % y
        return y
```

# The Function factotrace.py

## Recursive Functions

computing factorials recursively  
computing factorials iteratively

## Accumulating Parameters

tracing recursive functions automatically  
computing with accumulating parameters

## Recursive Problem Solving

check if a word is a palindrome

```
def factotrace(n,k):
    """
    prints out trace information in call k
    the initial value for k should be zero
    """
    s = k*' '
    s = s + 'factotrace(%d,%d):' % (n,k)
    if n <= 1:
        print s + ' base case, return 1'
        return 1
    else:
        print s + ' call for n-1 = ' + str(n-1)
        y = n*factotrace(n-1,k+1)
        print s + ' returning %d' % y
        return y
```

# The Function factotrace.py

## Recursive Functions

computing factorials recursively  
computing factorials iteratively

## Accumulating Parameters

tracing recursive functions automatically  
computing with accumulating parameters

## Recursive Problem Solving

check if a word is a palindrome

```
def factotrace(n,k):
    """
    prints out trace information in call k
    the initial value for k should be zero
    """
    s = k*' '
    s = s + 'factotrace(%d,%d):' % (n,k)
    if n <= 1:
        print s + ' base case, return 1'
        return 1
    else:
        print s + ' call for n-1 = ' + str(n-1)
        y = n*factotrace(n-1,k+1)
        print s + ' returning %d' % y
        return y
```

# Recursive Algorithms

## Recursive Functions

computing factorials recursively  
computing factorials iteratively

## Accumulating Parameters

tracing recursive functions automatically  
computing with accumulating parameters

## Recursive Problem Solving

check if a word is a palindrome

### 1 Recursive Functions

computing factorials recursively  
computing factorials iteratively

### 2 Accumulating Parameters

tracing recursive functions automatically  
computing with accumulating parameters

### 3 Recursive Problem Solving

check if a word is a palindrome

# Factorial in Accumulator

the function `factaccu`

## Recursive Functions

computing factorials recursively

computing factorials iteratively

## Accumulating Parameters

tracing recursive functions automatically

computing with accumulating parameters

## Recursive Problem Solving

check if a word is a palindrome

Like we add for the number of the function call, we can multiply in the accumulator.

```
def factaccu(n,f):  
    """  
    accumulates the factorial in f  
    call factaccu initially with f = 1  
    """  
    if n <= 1:  
        return f  
    else:  
        return factaccu(n-1,f*n)
```

# Factorial in Accumulator

the function `factaccu`

## Recursive Functions

computing factorials recursively

computing factorials iteratively

## Accumulating Parameters

tracing recursive functions automatically

computing with accumulating parameters

## Recursive Problem Solving

check if a word is a palindrome

Like we add for the number of the function call, we can multiply in the accumulator.

```
def factaccu(n,f):  
    """  
    accumulates the factorial in f  
    call factaccu initially with f = 1  
    """  
    if n <= 1:  
        return f  
    else:  
        return factaccu(n-1,f*n)
```

# Tracing Factorial Computations

## Recursive Functions

computing factorials recursively

computing factorials iteratively

## Accumulating Parameters

tracing recursive functions automatically

computing with accumulating parameters

## Recursive Problem Solving

check if a word is a palindrome

Call `factaccu` for  $n = 5$  and  $k = 1$ :

```
factaccu(5,1) call #0: call for n-1 = 4
factaccu(4,5) call #1: call for n-1 = 3
factaccu(3,20) call #2: call for n-1 = 2
factaccu(2,60) call #3: call for n-1 = 1
factaccu(1,120) call #4: returning 120
factaccu(2,60) call #3: returning 120
factaccu(3,20) call #2: returning 120
factaccu(4,5) call #1: returning 120
factaccu(5,1) call #0: returning 120
```

Computes  $1*5$ ,  $1*5*4$ ,  $1*5*4*3$ ,  $1*5*4*3*2$ , returns 120.

# Tracing Factorial Computations

## Recursive Functions

computing factorials recursively

computing factorials iteratively

## Accumulating Parameters

tracing recursive functions automatically

computing with accumulating parameters

## Recursive Problem Solving

check if a word is a palindrome

Call `factaccu` for  $n = 5$  and  $k = 1$ :

```
factaccu(5,1) call #0: call for n-1 = 4
factaccu(4,5) call #1: call for n-1 = 3
factaccu(3,20) call #2: call for n-1 = 2
factaccu(2,60) call #3: call for n-1 = 1
factaccu(1,120) call #4: returning 120
factaccu(2,60) call #3: returning 120
factaccu(3,20) call #2: returning 120
factaccu(4,5) call #1: returning 120
factaccu(5,1) call #0: returning 120
```

Computes  $1*5$ ,  $1*5*4$ ,  $1*5*4*3$ ,  $1*5*4*3*2$ ,  
returns 120.

# Tracing Factorial Computations

## Recursive Functions

computing factorials recursively

computing factorials iteratively

## Accumulating Parameters

tracing recursive functions automatically

computing with accumulating parameters

## Recursive Problem Solving

check if a word is a palindrome

Call `factaccu` for  $n = 5$  and  $k = 1$ :

```
factaccu(5,1) call #0: call for n-1 = 4
factaccu(4,5) call #1: call for n-1 = 3
factaccu(3,20) call #2: call for n-1 = 2
factaccu(2,60) call #3: call for n-1 = 1
factaccu(1,120) call #4: returning 120
factaccu(2,60) call #3: returning 120
factaccu(3,20) call #2: returning 120
factaccu(4,5) call #1: returning 120
factaccu(5,1) call #0: returning 120
```

Computes  $1*5$ ,  $1*5*4$ ,  $1*5*4*3$ ,  $1*5*4*3*2$ ,  
returns 120.

# Tracing Factorial Computations

## Recursive Functions

computing factorials recursively

computing factorials iteratively

## Accumulating Parameters

tracing recursive functions automatically

computing with accumulating parameters

## Recursive Problem Solving

check if a word is a palindrome

Call `factaccu` for `n = 5` and `k = 1`:

```
factaccu(5,1) call #0: call for n-1 = 4
factaccu(4,5) call #1: call for n-1 = 3
factaccu(3,20) call #2: call for n-1 = 2
factaccu(2,60) call #3: call for n-1 = 1
factaccu(1,120) call #4: returning 120
factaccu(2,60) call #3: returning 120
factaccu(3,20) call #2: returning 120
factaccu(4,5) call #1: returning 120
factaccu(5,1) call #0: returning 120
```

Computes  $1*5$ ,  $1*5*4$ ,  $1*5*4*3$ ,  $1*5*4*3*2$ , returns 120.

# Tracing Factorial Computations

## Recursive Functions

computing factorials recursively

computing factorials iteratively

## Accumulating Parameters

tracing recursive functions automatically

computing with accumulating parameters

## Recursive Problem Solving

check if a word is a palindrome

Call `factaccu` for `n = 5` and `k = 1`:

```
factaccu(5,1) call #0: call for n-1 = 4
factaccu(4,5) call #1: call for n-1 = 3
factaccu(3,20) call #2: call for n-1 = 2
factaccu(2,60) call #3: call for n-1 = 1
factaccu(1,120) call #4: returning 120
factaccu(2,60) call #3: returning 120
factaccu(3,20) call #2: returning 120
factaccu(4,5) call #1: returning 120
factaccu(5,1) call #0: returning 120
```

Computes  $1*5$ ,  $1*5*4$ ,  $1*5*4*3$ ,  $1*5*4*3*2$ , returns 120.

# Tracing Factorial Computations

## Recursive Functions

computing factorials recursively

computing factorials iteratively

## Accumulating Parameters

tracing recursive functions automatically

computing with accumulating parameters

## Recursive Problem Solving

check if a word is a palindrome

Call `factaccu` for  $n = 5$  and  $k = 1$ :

```
factaccu(5,1) call #0: call for n-1 = 4
factaccu(4,5) call #1: call for n-1 = 3
factaccu(3,20) call #2: call for n-1 = 2
factaccu(2,60) call #3: call for n-1 = 1
factaccu(1,120) call #4: returning 120
factaccu(2,60) call #3: returning 120
factaccu(3,20) call #2: returning 120
factaccu(4,5) call #1: returning 120
factaccu(5,1) call #0: returning 120
```

Computes  $1*5$ ,  $1*5*4$ ,  $1*5*4*3$ ,  $1*5*4*3*2$ ,  
returns 120.

# Tracing Factorial Computations

## Recursive Functions

computing factorials recursively

computing factorials iteratively

## Accumulating Parameters

tracing recursive functions automatically

computing with accumulating parameters

## Recursive Problem Solving

check if a word is a palindrome

Call `factaccu` for  $n = 5$  and  $k = 1$ :

```
factaccu(5,1) call #0: call for n-1 = 4
factaccu(4,5) call #1: call for n-1 = 3
factaccu(3,20) call #2: call for n-1 = 2
factaccu(2,60) call #3: call for n-1 = 1
factaccu(1,120) call #4: returning 120
factaccu(2,60) call #3: returning 120
factaccu(3,20) call #2: returning 120
factaccu(4,5) call #1: returning 120
factaccu(5,1) call #0: returning 120
```

Computes  $1*5$ ,  $1*5*4$ ,  $1*5*4*3$ ,  $1*5*4*3*2$ ,  
returns 120.

# Tracing Factorial Computations

## Recursive Functions

computing factorials recursively  
computing factorials iteratively

## Accumulating Parameters

tracing recursive functions automatically  
computing with accumulating parameters

## Recursive Problem Solving

check if a word is a palindrome

Call `factaccu` for  $n = 5$  and  $k = 1$ :

```
factaccu(5,1) call #0: call for n-1 = 4
factaccu(4,5) call #1: call for n-1 = 3
factaccu(3,20) call #2: call for n-1 = 2
factaccu(2,60) call #3: call for n-1 = 1
factaccu(1,120) call #4: returning 120
factaccu(2,60) call #3: returning 120
factaccu(3,20) call #2: returning 120
factaccu(4,5) call #1: returning 120
factaccu(5,1) call #0: returning 120
```

Computes  $1*5$ ,  $1*5*4$ ,  $1*5*4*3$ ,  $1*5*4*3*2$ ,  
returns 120.

# Tracing Factorial Computations

## Recursive Functions

computing factorials recursively

computing factorials iteratively

## Accumulating Parameters

tracing recursive functions automatically

computing with accumulating parameters

## Recursive Problem Solving

check if a word is a palindrome

Call `factaccu` for  $n = 5$  and  $k = 1$ :

```
factaccu(5,1) call #0: call for n-1 = 4
factaccu(4,5) call #1: call for n-1 = 3
factaccu(3,20) call #2: call for n-1 = 2
factaccu(2,60) call #3: call for n-1 = 1
factaccu(1,120) call #4: returning 120
factaccu(2,60) call #3: returning 120
factaccu(3,20) call #2: returning 120
factaccu(4,5) call #1: returning 120
factaccu(5,1) call #0: returning 120
```

Computes  $1*5$ ,  $1*5*4$ ,  $1*5*4*3$ ,  $1*5*4*3*2$ ,  
returns 120.

# Tracing Factorial Computations

## Recursive Functions

computing factorials recursively

computing factorials iteratively

## Accumulating Parameters

tracing recursive functions automatically

computing with accumulating parameters

## Recursive Problem Solving

check if a word is a palindrome

Call `factaccu` for  $n = 5$  and  $k = 1$ :

```
factaccu(5,1) call #0: call for n-1 = 4
factaccu(4,5) call #1: call for n-1 = 3
factaccu(3,20) call #2: call for n-1 = 2
factaccu(2,60) call #3: call for n-1 = 1
factaccu(1,120) call #4: returning 120
factaccu(2,60) call #3: returning 120
factaccu(3,20) call #2: returning 120
factaccu(4,5) call #1: returning 120
factaccu(5,1) call #0: returning 120
```

Computes  $1*5$ ,  $1*5*4$ ,  $1*5*4*3$ ,  $1*5*4*3*2$ ,  
returns 120.

# Automatic Tracing of factaccu

## Recursive Functions

computing factorials recursively  
computing factorials iteratively

## Accumulating Parameters

tracing recursive functions automatically  
computing with accumulating parameters

## Recursive Problem Solving

check if a word is a palindrome

```
def factatrace(n,f,k):
    """
    accumulates the factorial in f,
    k is used to trace the calls
    initialize f to 1 and k to 0
    """
    s = k*' '
    s = s + 'factatrace(%d,%d,%d)' % (n,f,k)
    if n <= 1:
        print s + ' returning ' + str(f)
        return f
    else:
        print s + ' call for n-1 = ' + str(n-1)
        y = factatrace(n-1,f*n,k+1)
        print s + ' returning %d' % y
        return y
```

# Automatic Tracing of factaccu

## Recursive Functions

computing factorials recursively  
computing factorials iteratively

## Accumulating Parameters

tracing recursive functions automatically  
computing with accumulating parameters

## Recursive Problem Solving

check if a word is a palindrome

```
def factatrace(n,f,k):
    """
    accumulates the factorial in f,
    k is used to trace the calls
    initialize f to 1 and k to 0
    """
    s = k*' '
    s = s + 'factatrace(%d,%d,%d)' % (n,f,k)
    if n <= 1:
        print s + ' returning ' + str(f)
        return f
    else:
        print s + ' call for n-1 = ' + str(n-1)
        y = factatrace(n-1,f*n,k+1)
        print s + ' returning %d' % y
        return y
```

# Automatic Tracing of `factaccu`

## Recursive Functions

computing factorials recursively  
computing factorials iteratively

## Accumulating Parameters

tracing recursive functions automatically  
computing with accumulating parameters

## Recursive Problem Solving

check if a word is a palindrome

```
def factatrace(n,f,k):
    """
    accumulates the factorial in f,
    k is used to trace the calls
    initialize f to 1 and k to 0
    """
    s = k*' '
    s = s + 'factatrace(%d,%d,%d)' % (n,f,k)
    if n <= 1:
        print s + ' returning ' + str(f)
        return f
    else:
        print s + ' call for n-1 = ' + str(n-1)
        y = factatrace(n-1,f*n,k+1)
        print s + ' returning %d' % y
        return y
```

# Automatic Tracing of factaccu

## Recursive Functions

computing factorials recursively  
computing factorials iteratively

## Accumulating Parameters

tracing recursive functions automatically  
computing with accumulating parameters

## Recursive Problem Solving

check if a word is a palindrome

```
def factatrace(n,f,k):
    """
    accumulates the factorial in f,
    k is used to trace the calls
    initialize f to 1 and k to 0
    """
    s = k*' '
    s = s + 'factatrace(%d,%d,%d)' % (n,f,k)
    if n <= 1:
        print s + ' returning ' + str(f)
        return f
    else:
        print s + ' call for n-1 = ' + str(n-1)
        y = factatrace(n-1,f*n,k+1)
        print s + ' returning %d' % y
        return y
```

# Automatic Tracing of factaccu

## Recursive Functions

computing factorials recursively  
computing factorials iteratively

## Accumulating Parameters

tracing recursive functions automatically  
computing with accumulating parameters

## Recursive Problem Solving

check if a word is a palindrome

```
def factatrace(n,f,k):
    """
    accumulates the factorial in f,
    k is used to trace the calls
    initialize f to 1 and k to 0
    """
    s = k*' '
    s = s + 'factatrace(%d,%d,%d)' % (n,f,k)
    if n <= 1:
        print s + ' returning ' + str(f)
        return f
    else:
        print s + ' call for n-1 = ' + str(n-1)
        y = factatrace(n-1,f*n,k+1)
        print s + ' returning %d' % y
        return y
```

# The Output of `factatrace`

## Recursive Functions

computing factorials recursively

computing factorials iteratively

## Accumulating Parameters

tracing recursive functions automatically

computing with accumulating parameters

## Recursive Problem Solving

check if a word is a palindrome

Call `factatrace` for  $n = 5$ ,  $f = 1$ ,  $k = 0$ :

```
factatrace(5,1,0) call for n-1 = 4
  factatrace(4,5,1) call for n-1 = 3
    factatrace(3,20,2) call for n-1 = 2
      factatrace(2,60,3) call for n-1 = 1
        factatrace(1,120,4) returning 120
      factatrace(2,60,3) returning 120
    factatrace(3,20,2) returning 120
  factatrace(4,5,1) returning 120
factatrace(5,1,0) returning 120
```

# Recursive Algorithms

## Recursive Functions

computing factorials recursively  
computing factorials iteratively

## Accumulating Parameters

tracing recursive functions automatically  
computing with accumulating parameters

## Recursive Problem Solving

check if a word is a palindrome

### 1 Recursive Functions

computing factorials recursively  
computing factorials iteratively

### 2 Accumulating Parameters

tracing recursive functions automatically  
computing with accumulating parameters

### 3 Recursive Problem Solving

check if a word is a palindrome

# Palindromes

If reading a word forwards and backwards is the same, then the word is a palindrome.

Examples: mom, dad, rotor.

```
>>> s = 'motor'
>>> L = [c for c in s]
>>> L
['m', 'o', 't', 'o', 'r']
>>> L.reverse()
>>> L
['r', 'o', 't', 'o', 'm']
>>> t = ''.join(L)
>>> t
'rotom'
>>> s == t
False
```

## Recursive Functions

computing factorials recursively  
computing factorials iteratively

## Accumulating Parameters

tracing recursive functions automatically  
computing with accumulating parameters

## Recursive Problem Solving

check if a word is a palindrome

# Palindromes

If reading a word forwards and backwards is the same, then the word is a palindrome.

Examples: mom, dad, rotor.

```
>>> s = 'motor'
>>> L = [c for c in s]
>>> L
['m', 'o', 't', 'o', 'r']
>>> L.reverse()
>>> L
['r', 'o', 't', 'o', 'm']
>>> t = ''.join(L)
>>> t
'rotom'
>>> s == t
False
```

## Recursive Functions

computing factorials recursively  
computing factorials iteratively

## Accumulating Parameters

tracing recursive functions automatically  
computing with accumulating parameters

## Recursive Problem Solving

check if a word is a palindrome

# Palindromes

If reading a word forwards and backwards is the same, then the word is a palindrome.

Examples: mom, dad, rotor.

Problem: define the function

```
def is_palindrome(x):
    """
    returns true if x is a palindrome
    """
```

Three base cases:

- 1 the word is empty or only one character long
- 2 first and last character are different
- 3 the word consists of two equal characters

## Recursive Functions

computing factorials recursively  
computing factorials iteratively

## Accumulating Parameters

tracing recursive functions automatically  
computing with accumulating parameters

## Recursive Problem Solving

check if a word is a palindrome

# Palindromes

If reading a word forwards and backwards is the same, then the word is a palindrome.

Examples: mom, dad, rotor.

Problem: define the function

```
def is_palindrome(x):
    """
    returns true if x is a palindrome
    """
```

Three base cases:

- 1 the word is empty or only one character long
- 2 first and last character are different
- 3 the word consists of two equal characters

## Recursive Functions

computing factorials recursively  
computing factorials iteratively

## Accumulating Parameters

tracing recursive functions automatically  
computing with accumulating parameters

## Recursive Problem Solving

check if a word is a palindrome

# Palindromes

## Recursive Functions

computing factorials recursively  
computing factorials iteratively

## Accumulating Parameters

tracing recursive functions automatically  
computing with accumulating parameters

## Recursive Problem Solving

check if a word is a palindrome

If reading a word forwards and backwards is the same, then the word is a palindrome.

Examples: mom, dad, rotor.

Problem: define the function

```
def is_palindrome(x):  
    """  
    returns true if x is a palindrome  
    """
```

Three base cases:

- 1 the word is empty or only one character long
- 2 first and last character are different
- 3 the word consists of two equal characters

# Palindromes

## Recursive Functions

computing factorials recursively  
computing factorials iteratively

## Accumulating Parameters

tracing recursive functions automatically  
computing with accumulating parameters

## Recursive Problem Solving

check if a word is a palindrome

If reading a word forwards and backwards is the same, then the word is a palindrome.

Examples: mom, dad, rotor.

Problem: define the function

```
def is_palindrome(x):
    """
    returns true if x is a palindrome
    """
```

Three base cases:

- 1 the word is empty or only one character long
- 2 first and last character are different
- 3 the word consists of two equal characters

# Palindromes

## Recursive Functions

computing factorials recursively  
computing factorials iteratively

## Accumulating Parameters

tracing recursive functions automatically  
computing with accumulating parameters

## Recursive Problem Solving

check if a word is a palindrome

If reading a word forwards and backwards is the same, then the word is a palindrome.

Examples: mom, dad, rotor.

Problem: define the function

```
def is_palindrome(x):
    """
    returns true if x is a palindrome
    """
```

Three base cases:

- ① the word is empty or only one character long
- ② first and last character are different
- ③ the word consists of two equal characters

# The Function `is_palindrome`

## Recursive Functions

computing factorials recursively  
computing factorials iteratively

## Accumulating Parameters

tracing recursive functions automatically  
computing with accumulating parameters

## Recursive Problem Solving

check if a word is a palindrome

```
def is_palindrome(x):  
    """  
    returns true if x is a palindrome  
    """  
    if len(x) <= 1:  
        return True  
    elif x[0] != x[len(x)-1]:  
        return False  
    elif len(x) == 2:  
        return True  
    else:  
        y = x[1:len(x)-1]  
        return is_palindrome(y)
```

# The Function `is_palindrome`

## Recursive Functions

computing factorials recursively  
computing factorials iteratively

## Accumulating Parameters

tracing recursive functions automatically  
computing with accumulating parameters

## Recursive Problem Solving

check if a word is a palindrome

```
def is_palindrome(x):  
    """  
    returns true if x is a palindrome  
    """  
    if len(x) <= 1:  
        return True  
    elif x[0] != x[len(x)-1]:  
        return False  
    elif len(x) == 2:  
        return True  
    else:  
        y = x[1:len(x)-1]  
        return is_palindrome(y)
```

# The Function `is_palindrome`

## Recursive Functions

computing factorials  
recursively  
computing factorials  
iteratively

## Accumulating Parameters

tracing recursive  
functions  
automatically  
computing with  
accumulating  
parameters

## Recursive Problem Solving

check if a word is a  
palindrome

```
def is_palindrome(x):  
    """  
    returns true if x is a palindrome  
    """  
    if len(x) <= 1:  
        return True  
    elif x[0] != x[len(x)-1]:  
        return False  
    elif len(x) == 2:  
        return True  
    else:  
        y = x[1:len(x)-1]  
        return is_palindrome(y)
```

# The Function `is_palindrome`

## Recursive Functions

computing factorials recursively  
computing factorials iteratively

## Accumulating Parameters

tracing recursive functions automatically  
computing with accumulating parameters

## Recursive Problem Solving

check if a word is a palindrome

```
def is_palindrome(x):  
    """  
    returns true if x is a palindrome  
    """  
    if len(x) <= 1:  
        return True  
    elif x[0] != x[len(x)-1]:  
        return False  
    elif len(x) == 2:  
        return True  
    else:  
        y = x[1:len(x)-1]  
        return is_palindrome(y)
```

# Running the Script

## palindromes.py

The file `palindromes.py` contains the definition of the function `is_palindrome` and

```
def main():
    """
    prompts the user for a word and checks
    if it is a palindrome
    """
    w = raw_input('give a word : ')
    s = 'the word \'' + w + '\' is '
    if not is_palindrome(w):
        s = s + 'not '
    s = s + 'a palindrome'
    print s

if __name__ == "__main__": main()
```

### Recursive Functions

computing factorials recursively

computing factorials iteratively

### Accumulating Parameters

tracing recursive functions automatically

computing with accumulating parameters

### Recursive Problem Solving

check if a word is a palindrome

Recursive  
Functions

computing factorials  
recursively  
computing factorials  
iteratively

Accumulating  
Parameters

tracing recursive  
functions  
automatically  
computing with  
accumulating  
parameters

Recursive  
Problem  
Solving

check if a word is a  
palindrome

# Running the Script

## palindromes.py

Giving on input a string of characters:

```
$ python palindromes.py  
give a word : palindromes  
the word "palindromes" is not a palindrome
```

Because of the `raw_input`:

```
$ python palindromes.py  
give a word : 1234321  
the word "1234321" is a palindrome
```

The palindrome tester works just as well on numbers.

Recursive  
Functions

computing factorials  
recursively  
computing factorials  
iteratively

Accumulating  
Parameters

tracing recursive  
functions  
automatically  
computing with  
accumulating  
parameters

Recursive  
Problem  
Solving

check if a word is a  
palindrome

# Running the Script

## palindromes.py

Giving on input a string of characters:

```
$ python palindromes.py  
give a word : palindromes  
the word "palindromes" is not a palindrome
```

Because of the `raw_input`:

```
$ python palindromes.py  
give a word : 1234321  
the word "1234321" is a palindrome
```

The palindrome tester works just as well on numbers.

## Exercises

Recursive  
Functionscomputing factorials  
recursivelycomputing factorials  
iterativelyAccumulating  
Parameterstracing recursive  
functions  
automaticallycomputing with  
accumulating  
parametersRecursive  
Problem  
Solvingcheck if a word is a  
palindrome

- 1 The  $n$ th Fibonacci number  $F_n$  is defined for  $n \geq 2$  as  $F_n = F_{n-1} + F_{n-2}$  and  $F_0 = 0$ ,  $F_1 = 1$ .  
Give a Python function `Fibonacci` to compute  $F_n$ .
- 2 Use an accumulating parameter `k` to `Fibonacci` to count the function calls. When tracing the execution, print with `k` spaces as indentations.
- 3 Write an equivalent C function to compute factorials recursively. Use a main interactive program to test it.
- 4 Extend `is_palindrome` with an extra accumulating parameter `k` to keep track of the function calls.  
Trace the execution with this extended function, using `k` spaces as indentations.
- 5 Write a recursive function to sum a list of numbers.  
There are two base cases: an empty list and a list with one element. The recursive sum add the first element to the sum of the rest of the list.

# Summary and Assignments

## Recursive Functions

computing factorials recursively  
computing factorials iteratively

## Accumulating Parameters

tracing recursive functions automatically  
computing with accumulating parameters

## Recursive Problem Solving

check if a word is a palindrome

Background material for this lecture:

- §5.5 in *Computer Science: an overview*,
- start of chapter 9 of *Python Programming*.