

Outline

Histograms

tallying the votes
global and local variables

Arguments of Functions

of variable length
using keywords for optional arguments

Functions using Functions

the trapezoidal rule

Functional Programming

Summary + Assignments

Histograms

tallying the votes
global and local variables

Arguments of Functions

of variable length
using keywords for optional
arguments

Functions using Functions

the trapezoidal rule

Functional Programming

Summary +
Assignments

MCS 260 Lecture 14
Introduction to Computer Science
Jan Vershelde, 28 September 2007

Histograms

interpreting results of a simulation

MCS 260 L-14

28 September
2007

How do probability distributions arise in applications?

Run a simulation and tally outcomes into separate bins.

Check whether a coin is fair:

1. do a large number of coin tosses,
2. count number of heads and tails,
3. if unequal #heads and #tails, suspect unfair.

Raising the number of tosses will increase confidence.

This coin toss problem illustrates how to check whether data is uniformly distributed.

Histograms

tallying the votes
global and local variables

Arguments of Functions

of variable length
using keywords for optional arguments

Functions using Functions

the trapezoidal rule

Functional Programming

Summary + Assignments

Histograms

interpreting results of a simulation

MCS 260 L-14

28 September
2007

How do probability distributions arise in applications?

Run a simulation and tally outcomes into separate bins.

Check whether a coin is fair:

1. do a large number of coin tosses,
2. count number of heads and tails,
3. if unequal #heads and #tails, suspect unfair.

Raising the number of tosses will increase confidence.

This coin toss problem illustrates how to check whether data is uniformly distributed.

Histograms

tallying the votes
global and local variables

Arguments of Functions

of variable length
using keywords for optional arguments

Functions using Functions

the trapezoidal rule

Functional Programming

Summary +
Assignments

Histograms

interpreting results of a simulation

MCS 260 L-14

28 September
2007

Histograms

tallying the votes
global and local variables

Arguments of Functions

of variable length
using keywords for optional
arguments

Functions using Functions

the trapezoidal rule

Functional Programming

Summary + Assignments

How do probability distributions arise in applications?

Run a simulation and tally outcomes into separate bins.

Check whether a coin is fair:

1. do a large number of coin tosses,
2. count number of heads and tails,
3. if unequal #heads and #tails, suspect unfair.

Raising the number of tosses will increase confidence.

This coin toss problem illustrates how to check whether data is uniformly distributed.

Histograms

interpreting results of a simulation

MCS 260 L-14

28 September
2007

How do probability distributions arise in applications?

Run a simulation and tally outcomes into separate bins.

Check whether a coin is fair:

1. do a large number of coin tosses,
2. count number of heads and tails,
3. if unequal #heads and #tails, suspect unfair.

Raising the number of tosses will increase confidence.

This coin toss problem illustrates how to check whether data is uniformly distributed.

Histograms

tallying the votes
global and local variables

Arguments of Functions

of variable length
using keywords for optional arguments

Functions using Functions

the trapezoidal rule

Functional Programming

Summary +
Assignments

Histograms

interpreting results of a simulation

MCS 260 L-14

28 September
2007

How do probability distributions arise in applications?

Run a simulation and tally outcomes into separate bins.

Check whether a coin is fair:

1. do a large number of coin tosses,
2. count number of heads and tails,
3. if unequal #heads and #tails, suspect unfair.

Raising the number of tosses will increase confidence.

This coin toss problem illustrates how to check whether data is uniformly distributed.

Histograms

tallying the votes
global and local variables

Arguments of Functions

of variable length
using keywords for optional arguments

Functions using Functions

the trapezoidal rule

Functional Programming

Summary +
Assignments

Histograms

interpreting results of a simulation

MCS 260 L-14

28 September
2007

How do probability distributions arise in applications?

Run a simulation and tally outcomes into separate bins.

Check whether a coin is fair:

1. do a large number of coin tosses,
2. count number of heads and tails,
3. if unequal #heads and #tails, suspect unfair.

Raising the number of tosses will increase confidence.

This coin toss problem illustrates how to check whether data is uniformly distributed.

Histograms

tallying the votes
global and local variables

Arguments of Functions

of variable length
using keywords for optional arguments

Functions using Functions

the trapezoidal rule

Functional Programming

Summary +
Assignments

Histograms

interpreting results of a simulation

MCS 260 L-14

28 September
2007

Histograms

tallying the votes
global and local variables

Arguments of Functions

of variable length
using keywords for optional
arguments

Functions using Functions

the trapezoidal rule

Functional Programming

Summary +
Assignments

How do probability distributions arise in applications?

Run a simulation and tally outcomes into separate bins.

Check whether a coin is fair:

1. do a large number of coin tosses,
2. count number of heads and tails,
3. if unequal #heads and #tails, suspect unfair.

Raising the number of tosses will increase confidence.

This coin toss problem illustrates how to check whether data is uniformly distributed.

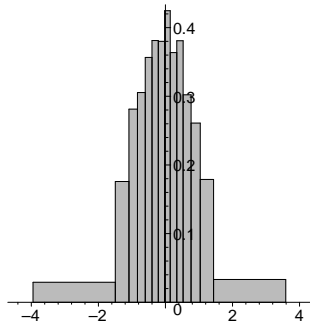
Histograms in Maple

MCS 260 L-14

28 September
2007

Given any list of data, `stats[statplots,histogram]` produces a plot:

```
[ > data := [stats[random,normald[0,1]](5000)]:  
> stats[statplots,histogram](data);
```



Histograms

tallying the votes
global and local variables

Arguments of Functions

of variable length
using keywords for optional arguments

Functions using Functions

the trapezoidal rule

Functional Programming

Summary + Assignments

Histograms

tallying the votes

global and local variables

Histograms

tallying the votes

global and local variables

Arguments of Functions

of variable length

using keywords for optional arguments

Arguments of Functions

of variable length

using keywords for optional arguments

Functions using Functions

the trapezoidal rule

Functions using Functions

the trapezoidal rule

Functional Programming

Functional Programming

Summary + Assignments

Summary + Assignments

Tallying Votes

tossing votes as coins

MCS 260 L-14

28 September
2007

Histograms

tallying the votes

global and local variables

Arguments of
Functions

of variable length

using keywords for optional
arguments

Functions using
Functions

the trapezoidal rule

Functional
Programming

Summary +
Assignments

Problem: make a machine to count votes.

Open democratic voting protocol (Yes or No):

1. machine says name of each member
2. upon hearing name, member says Yes or No
3. machine updates tally of Yes and No votes
4. at end of vote, program shows tally

Observe: this is a variant of the coin toss problem.

Tallying Votes

tossing votes as coins

MCS 260 L-14

28 September
2007

Histograms

tallying the votes

global and local variables

Arguments of
Functions

of variable length

using keywords for optional
arguments

Functions using
Functions

the trapezoidal rule

Functional
Programming

Summary +
Assignments

Problem: make a machine to count votes.

Open democratic voting protocol (Yes or No):

1. machine says name of each member
2. upon hearing name, member says Yes or No
3. machine updates tally of Yes and No votes
4. at end of vote, program shows tally

Observe: this is a variant of the coin toss problem.

Tallying Votes

tossing votes as coins

MCS 260 L-14

28 September
2007

Histograms

tallying the votes

global and local variables

Arguments of
Functions

of variable length

using keywords for optional
arguments

Functions using
Functions

the trapezoidal rule

Functional
Programming

Summary +
Assignments

Problem: make a machine to count votes.

Open democratic voting protocol (Yes or No):

1. machine says name of each member
2. upon hearing name, member says Yes or No
3. machine updates tally of Yes and No votes
4. at end of vote, program shows tally

Observe: this is a variant of the coin toss problem.

Tallying Votes

tossing votes as coins

MCS 260 L-14

28 September
2007

Histograms

tallying the votes

global and local variables

Arguments of
Functions

of variable length

using keywords for optional
arguments

Functions using
Functions

the trapezoidal rule

Functional
Programming

Summary +
Assignments

Problem: make a machine to count votes.

Open democratic voting protocol (Yes or No):

1. machine says name of each member
2. upon hearing name, member says Yes or No
3. machine updates tally of Yes and No votes
4. at end of vote, program shows tally

Observe: this is a variant of the coin toss problem.

Tallying Votes

tossing votes as coins

MCS 260 L-14

28 September
2007

Problem: make a machine to count votes.

Open democratic voting protocol (Yes or No):

1. machine says name of each member
2. upon hearing name, member says Yes or No
3. machine updates tally of Yes and No votes
4. at end of vote, program shows tally

Observe: this is a variant of the coin toss problem.

Histograms

tallying the votes

global and local variables

Arguments of
Functions

of variable length

using keywords for optional
arguments

Functions using
Functions

the trapezoidal rule

Functional
Programming

Summary +
Assignments

Tallying Votes

tossing votes as coins

MCS 260 L-14

28 September
2007

Histograms

tallying the votes

global and local variables

Arguments of
Functions

of variable length

using keywords for optional
arguments

Functions using
Functions

the trapezoidal rule

Functional
Programming

Summary +
Assignments

Problem: make a machine to count votes.

Open democratic voting protocol (Yes or No):

1. machine says name of each member
2. upon hearing name, member says Yes or No
3. machine updates tally of Yes and No votes
4. at end of vote, program shows tally

Observe: this is a variant of the coin toss problem.

Voting in Action

MCS 260 L-14

28 September
2007

Running the program `votes.py` at the prompt `$`:

```
$ python votes.py
Vote yes or no, 0 to stop
approve ? (y/n) y
Vote yes or no, 0 to stop
approve ? (y/n) y
Vote yes or no, 0 to stop
approve ? (y/n) n
Vote yes or no, 0 to stop
approve ? (y/n) n
Vote yes or no, 0 to stop
approve ? (y/n) y
Vote yes or no, 0 to stop
approve ? (y/n) 0
Tally of votes : [3, 2]
```

Histograms

tallying the votes

global and local variables

Arguments of
Functions

of variable length

using keywords for optional
arguments

Functions using
Functions

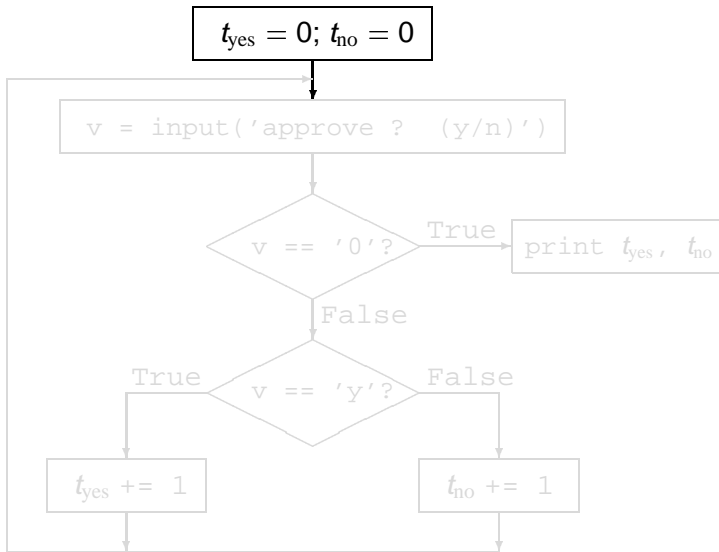
the trapezoidal rule

Functional
Programming

Summary +
Assignments

Flowchart

of the voting machine



Histograms

tallying the votes
global and local variables

Arguments of Functions

of variable length
using keywords for optional
arguments

Functions using Functions

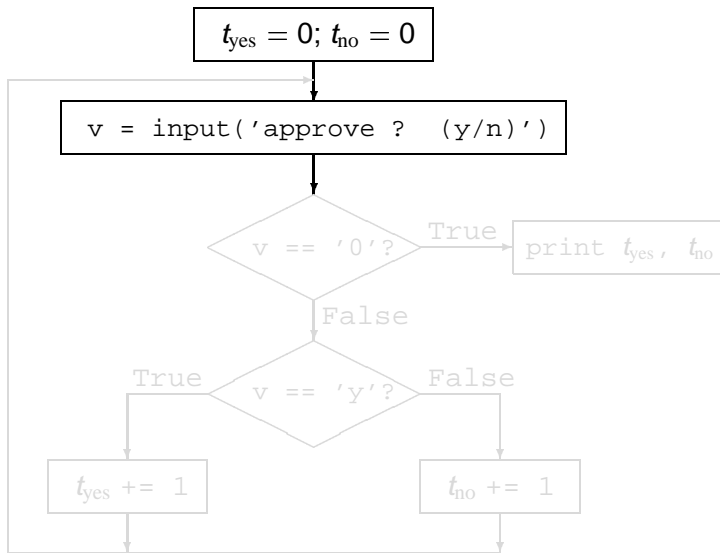
the trapezoidal rule

Functional Programming

Summary + Assignments

Flowchart

of the voting machine

[Histograms](#)

tallying the votes
global and local variables

[Arguments of Functions](#)

of variable length
using keywords for optional arguments

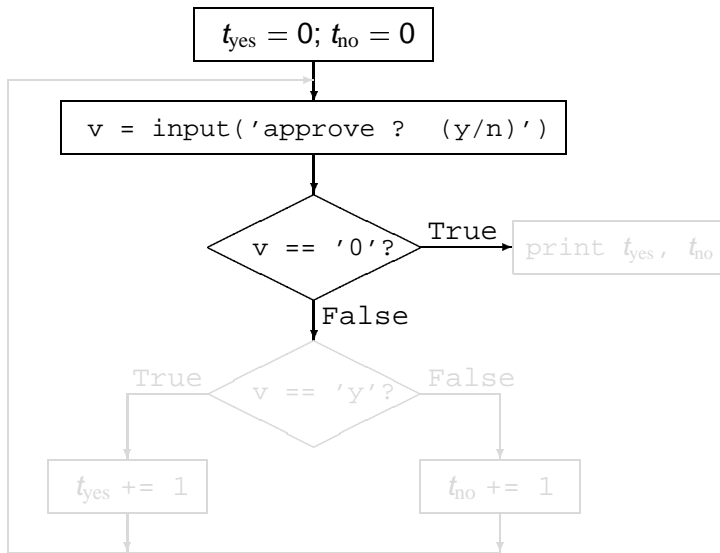
[Functions using Functions](#)

the trapezoidal rule

[Functional Programming](#)[Summary + Assignments](#)

Flowchart

of the voting machine

[Histograms](#)

tallying the votes
global and local variables

[Arguments of Functions](#)

of variable length
using keywords for optional arguments

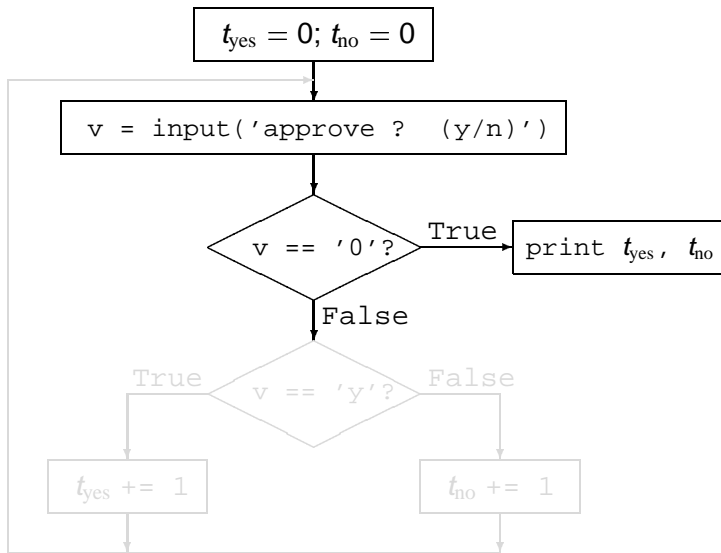
[Functions using Functions](#)

the trapezoidal rule

[Functional Programming](#)[Summary + Assignments](#)

Flowchart

of the voting machine



Histograms

tallying the votes
global and local variables

Arguments of
Functions

of variable length
using keywords for optional
arguments

Functions using
Functions

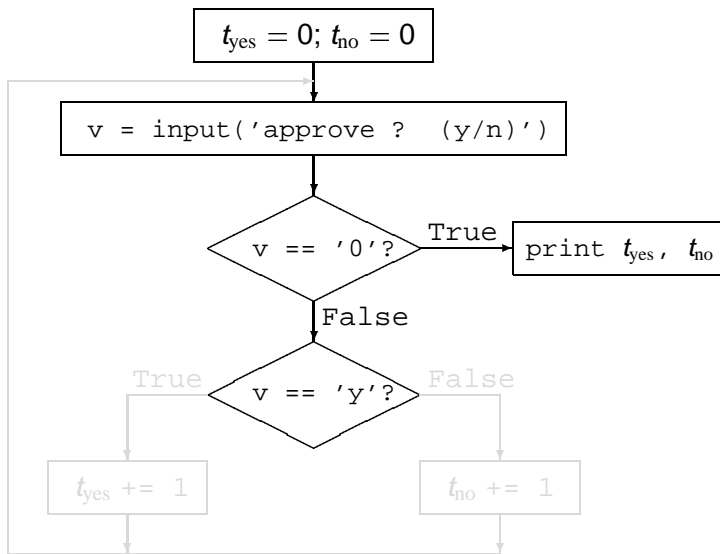
the trapezoidal rule

Functional
Programming

Summary +
Assignments

Flowchart

of the voting machine



Histograms

tallying the votes
global and local variables

Arguments of
Functions

of variable length
using keywords for optional
arguments

Functions using
Functions

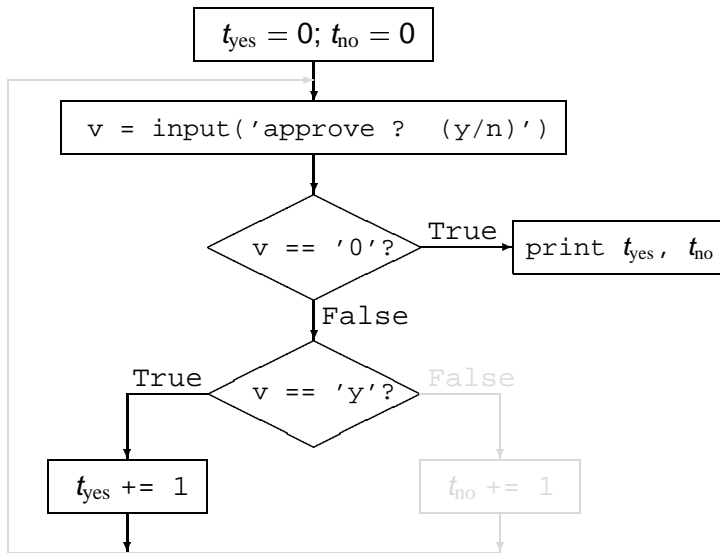
the trapezoidal rule

Functional
Programming

Summary +
Assignments

Flowchart

of the voting machine



Histograms

tallying the votes
global and local variables

Arguments of
Functions

of variable length
using keywords for optional
arguments

Functions using
Functions

the trapezoidal rule

Functional
Programming

Summary +
Assignments

Flowchart

of the voting machine

Histograms

tallying the votes
global and local variables

Arguments of Functions

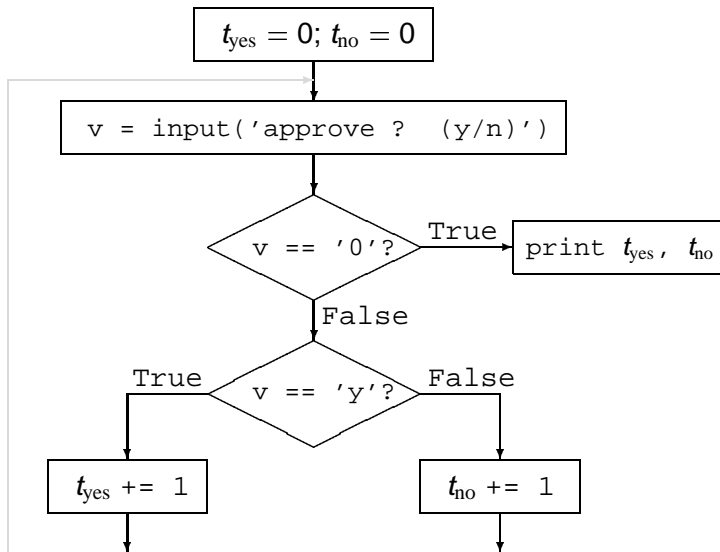
of variable length
using keywords for optional
arguments

Functions using Functions

the trapezoidal rule

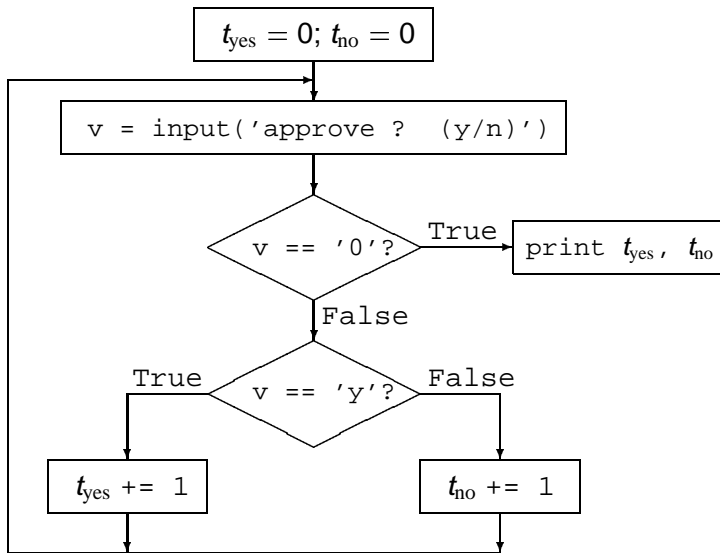
Functional Programming

Summary + Assignments



Flowchart

of the voting machine



Histograms

tallying the votes
global and local variables

Arguments of
Functions

of variable length
using keywords for optional
arguments

Functions using
Functions

the trapezoidal rule

Functional
Programming

Summary +
Assignments

Outline

MCS 260 L-14

28 September
2007

Histograms

tallying the votes

global and local variables

Histograms

tallying the votes

global and local variables

Arguments of Functions

of variable length

using keywords for optional arguments

Arguments of Functions

of variable length

using keywords for optional arguments

Functions using Functions

the trapezoidal rule

Functions using Functions

the trapezoidal rule

Functional Programming

Functional Programming

Summary + Assignments

Summary + Assignments

Global and Local Variables

hierarchy imposed on data

MCS 260 L-14

28 September
2007

In top down design we distinguish between

- ▶ functions that focus on one particular task
- ▶ the main program that calls the functions

Also the data fits into two categories:

- ▶ variables inside a function are *local*
- ▶ data managed by the main program is *global*

Example, in the voting machine:

- ▶ the variable to store the answer will be local
- ▶ the tally of the votes is global

Histograms

tallying the votes

global and local variables

Arguments of
Functions

of variable length

using keywords for optional
arguments

Functions using
Functions

the trapezoidal rule

Functional
Programming

Summary +
Assignments

Global and Local Variables

hierarchy imposed on data

MCS 260 L-14

28 September
2007

In top down design we distinguish between

- ▶ functions that focus on one particular task
- ▶ the main program that calls the functions

Also the data fits into two categories:

- ▶ variables inside a function are *local*
- ▶ data managed by the main program is *global*

Example, in the voting machine:

- ▶ the variable to store the answer will be local
- ▶ the tally of the votes is global

Histograms

tallying the votes

global and local variables

Arguments of
Functions

of variable length

using keywords for optional
arguments

Functions using
Functions

the trapezoidal rule

Functional
Programming

Summary +
Assignments

Global and Local Variables

hierarchy imposed on data

MCS 260 L-14

28 September
2007

In top down design we distinguish between

- ▶ functions that focus on one particular task
- ▶ the main program that calls the functions

Also the data fits into two categories:

- ▶ variables inside a function are *local*
- ▶ data managed by the main program is *global*

Example, in the voting machine:

- ▶ the variable to store the answer will be local
- ▶ the tally of the votes is global

Histograms

tallying the votes

global and local variables

Arguments of
Functions

of variable length

using keywords for optional
arguments

Functions using
Functions

the trapezoidal rule

Functional
Programming

Summary +
Assignments

Global and Local Variables

hierarchy imposed on data

MCS 260 L-14

28 September
2007

In top down design we distinguish between

- ▶ functions that focus on one particular task
- ▶ the main program that calls the functions

Also the data fits into two categories:

- ▶ variables inside a function are *local*
- ▶ data managed by the main program is *global*

Example, in the voting machine:

- ▶ the variable to store the answer will be local
- ▶ the tally of the votes is global

Histograms

tallying the votes

global and local variables

Arguments of
Functions

of variable length

using keywords for optional
arguments

Functions using
Functions

the trapezoidal rule

Functional
Programming

Summary +
Assignments

Python Functions are Functions

A function f in the proper mathematical sense, called like $y = f(x)$, does not change the argument x of the function.

Updating the tally t with vote v with the function `update(t, v)`, called as `t = update(t, v)`.

The function `update` will

1. copy `t` to a new variable `nt`,
2. adjust `nt` using the value of `v`,
3. return `nt` to the caller or `update`.

The caller of `update(t, v)` will assign the returned value to `t`.

Technical terminology (in comparison with C):

- ▶ Python functions have only call by value.
- ▶ Python does not support call by reference.

Histograms

tallying the votes
global and local variables

Arguments of
Functions

of variable length
using keywords for optional
arguments

Functions using
Functions

the trapezoidal rule

Functional
Programming

Summary +
Assignments

Python Functions are Functions

A function f in the proper mathematical sense, called like $y = f(x)$, does not change the argument x of the function.

Updating the tally t with vote v with the function `update(t, v)`, called as $t = \text{update}(t, v)$.

The function `update` will

1. copy t to a new variable nt ,
2. adjust nt using the value of v ,
3. return nt to the caller or `update`.

The caller of `update(t, v)` will assign the returned value to t .

Technical terminology (in comparison with C):

- ▶ Python functions have only call by value.
- ▶ Python does not support call by reference.

Histograms

tallying the votes

global and local variables

Arguments of
Functions

of variable length

using keywords for optional
arguments

Functions using
Functions

the trapezoidal rule

Functional
Programming

Summary +
Assignments

Python Functions are Functions

A function f in the proper mathematical sense, called like $y = f(x)$, does not change the argument x of the function.

Updating the tally t with vote v with the function `update(t, v)`, called as $t = \text{update}(t, v)$.

The function `update` will

1. copy t to a new variable nt ,
2. adjust nt using the value of v ,
3. return nt to the caller or `update`.

The caller of `update(t, v)` will assign the returned value to t .

Technical terminology (in comparison with C):

- ▶ Python functions have only call by value.
- ▶ Python does not support call by reference.

Histograms

tallying the votes
global and local variables

Arguments of
Functions

of variable length
using keywords for optional
arguments

Functions using
Functions

the trapezoidal rule

Functional
Programming

Summary +
Assignments

Python Functions are Functions

A function f in the proper mathematical sense, called like $y = f(x)$, does not change the argument x of the function.

Updating the tally t with vote v with the function `update(t, v)`, called as $t = \text{update}(t, v)$.

The function `update` will

1. copy t to a new variable nt ,
2. adjust nt using the value of v ,
3. return nt to the caller or `update`.

The caller of `update(t, v)` will assign the returned value to t .

Technical terminology (in comparison with C):

- ▶ Python functions have only call by value.
- ▶ Python does not support call by reference.

Histograms

tallying the votes
global and local variables

Arguments of
Functions

of variable length
using keywords for optional
arguments

Functions using
Functions

the trapezoidal rule

Functional
Programming

Summary +
Assignments

Python Functions are Functions

A function f in the proper mathematical sense, called like $y = f(x)$, does not change the argument x of the function.

Updating the tally t with vote v with the function `update(t, v)`, called as `t = update(t, v)`.

The function `update` will

1. copy `t` to a new variable `nt`,
2. adjust `nt` using the value of `v`,
3. return `nt` to the caller or `update`.

The caller of `update(t, v)` will assign the returned value to `t`.

Technical terminology (in comparison with C):

- ▶ Python functions have only call by value.
- ▶ Python does not support call by reference.

Histograms

tallying the votes

global and local variables

Arguments of
Functions

of variable length

using keywords for optional
arguments

Functions using
Functions

the trapezoidal rule

Functional
Programming

Summary +
Assignments

Python Functions are Functions

A function f in the proper mathematical sense, called like $y = f(x)$, does not change the argument x of the function.

Updating the tally t with vote v with the function `update(t, v)`, called as $t = \text{update}(t, v)$.

The function `update` will

1. copy t to a new variable nt ,
2. adjust nt using the value of v ,
3. return nt to the caller or `update`.

The caller of `update(t, v)` will assign the returned value to t .

Technical terminology (in comparison with C):

- ▶ Python functions have only call by value.
- ▶ Python does not support call by reference.

Histograms

tallying the votes
global and local variables

Arguments of
Functions

of variable length
using keywords for optional
arguments

Functions using
Functions

the trapezoidal rule

Functional
Programming

Summary +
Assignments

Python Functions are Functions

A function f in the proper mathematical sense, called like $y = f(x)$, does not change the argument x of the function.

Updating the tally t with vote v with the function `update(t, v)`, called as $t = \text{update}(t, v)$.

The function `update` will

1. copy t to a new variable nt ,
2. adjust nt using the value of v ,
3. return nt to the caller or `update`.

The caller of `update(t, v)` will assign the returned value to t .

Technical terminology (in comparison with C):

- ▶ Python functions have only call by value.
- ▶ Python does not support call by reference.

Histograms

tallying the votes
global and local variables

Arguments of
Functions

of variable length
using keywords for optional
arguments

Functions using
Functions

the trapezoidal rule

Functional
Programming

Summary +
Assignments

Python Functions are Functions

A function f in the proper mathematical sense, called like $y = f(x)$, does not change the argument x of the function.

Updating the tally t with vote v with the function `update(t, v)`, called as $t = \text{update}(t, v)$.

The function `update` will

1. copy t to a new variable nt ,
2. adjust nt using the value of v ,
3. return nt to the caller or `update`.

The caller of `update(t, v)` will assign the returned value to t .

Technical terminology (in comparison with C):

- ▶ Python functions have only call by value.
- ▶ Python does not support call by reference.

Histograms

tallying the votes
global and local variables

Arguments of
Functions

of variable length
using keywords for optional
arguments

Functions using
Functions

the trapezoidal rule

Functional
Programming

Summary +
Assignments

The Skeleton of votes.py

top down design in words

```
# L-14 MCS 260 28 Sep 2007  histograms
#
# A voting machine tallies votes.
#
tally = [0,0] # tally[0] counts yes votes
              # tally[1] counts no votes

def poll():
    "asks whether approve or not"

def update(t,vote):
    "updates tally t with vote"

while True:      # main program
    v = poll()
    if v == '0': break
    tally = update(tally,v)
print 'Tally of votes :', tally
```

MCS 260 L-14

28 September
2007

[Histograms](#)

tallying the votes

global and local variables

[Arguments of
Functions](#)

of variable length

using keywords for optional
arguments

[Functions using
Functions](#)

the trapezoidal rule

[Functional
Programming](#)

[Summary +
Assignments](#)

The Skeleton of votes.py

top down design in words

```
# L-14 MCS 260 28 Sep 2007  histograms
#
# A voting machine tallies votes.
#
tally = [0,0] # tally[0] counts yes votes
              # tally[1] counts no votes

def poll():
    "asks whether approve or not"

def update(t,vote):
    "updates tally t with vote"

while True:      # main program
    v = poll()
    if v == '0': break
    tally = update(tally,v)
print 'Tally of votes :', tally
```

MCS 260 L-14

28 September
2007

[Histograms](#)

tallying the votes
global and local variables

[Arguments of
Functions](#)

of variable length
using keywords for optional
arguments

[Functions using
Functions](#)

the trapezoidal rule

[Functional
Programming](#)

[Summary +
Assignments](#)

The Skeleton of votes.py

top down design in words

```
# L-14 MCS 260 28 Sep 2007  histograms
#
# A voting machine tallies votes.
#
tally = [0,0] # tally[0] counts yes votes
              # tally[1] counts no votes

def poll():
    "asks whether approve or not"

def update(t,vote):
    "updates tally t with vote"

while True:      # main program
    v = poll()
    if v == '0': break
    tally = update(tally,v)
print 'Tally of votes :', tally
```

MCS 260 L-14

28 September
2007

[Histograms](#)

tallying the votes
global and local variables

[Arguments of
Functions](#)

of variable length
using keywords for optional
arguments

[Functions using
Functions](#)

the trapezoidal rule

[Functional
Programming](#)

[Summary +
Assignments](#)

The Skeleton of votes.py

top down design in words

```
# L-14 MCS 260 28 Sep 2007  histograms
#
# A voting machine tallies votes.
#
tally = [0,0] # tally[0] counts yes votes
              # tally[1] counts no votes

def poll():
    "asks whether approve or not"

def update(t,vote):
    "updates tally t with vote"

while True:      # main program
    v = poll()
    if v == '0': break
    tally = update(tally,v)
print 'Tally of votes :', tally
```

MCS 260 L-14

28 September
2007

[Histograms](#)

tallying the votes

global and local variables

[Arguments of
Functions](#)

of variable length

using keywords for optional
arguments

[Functions using
Functions](#)

the trapezoidal rule

[Functional
Programming](#)

[Summary +
Assignments](#)

The Functions

poll() and update(t,vote)

```
def poll():
    "asks whether approve or not"
    print 'Vote yes or no, 0 to stop'
    answer = raw_input('approve ? (y/n) ')
    return answer
```

answer is a local variable in poll

```
def update(t,vote):
    "updates tally t with vote"
    nt = t
    if vote == 'y':
        nt[0] += 1
    elif vote == 'n':
        nt[1] += 1
    return nt
```

assigning to t occurs via local variable nt

Histograms

tallying the votes

global and local variables

Arguments of
Functions

of variable length

using keywords for optional
arguments

Functions using
Functions

the trapezoidal rule

Functional
Programming

Summary +
Assignments

The Functions

poll() and update(t,vote)

```
def poll():
    "asks whether approve or not"
    print 'Vote yes or no, 0 to stop'
    answer = raw_input('approve ? (y/n) ')
    return answer
```

answer is a local variable in poll

```
def update(t,vote):
    "updates tally t with vote"
    nt = t
    if vote == 'y':
        nt[0] += 1
    elif vote == 'n':
        nt[1] += 1
    return nt
```

assigning to t occurs via local variable nt

Histograms

tallying the votes

global and local variables

Arguments of
Functions

of variable length

using keywords for optional
arguments

Functions using
Functions

the trapezoidal rule

Functional
Programming

Summary +
Assignments

The Functions

poll() and update(t,vote)

```
def poll():
    "asks whether approve or not"
    print 'Vote yes or no, 0 to stop'
    answer = raw_input('approve ? (y/n) ')
    return answer
```

answer is a local variable in poll

```
def update(t,vote):
    "updates tally t with vote"
    nt = t
    if vote == 'y':
        nt[0] += 1
    elif vote == 'n':
        nt[1] += 1
    return nt
```

assigning to t occurs via local variable nt

Histograms

tallying the votes

global and local variables

Arguments of
Functions

of variable length

using keywords for optional
arguments

Functions using
Functions

the trapezoidal rule

Functional
Programming

Summary +
Assignments

The Functions

poll() and update(t,vote)

```
def poll():
    "asks whether approve or not"
    print 'Vote yes or no, 0 to stop'
    answer = raw_input('approve ? (y/n) ')
    return answer
```

answer is a local variable in poll

```
def update(t,vote):
    "updates tally t with vote"
    nt = t
    if vote == 'y':
        nt[0] += 1
    elif vote == 'n':
        nt[1] += 1
    return nt
```

assigning to t occurs via local variable nt

Histograms

tallying the votes

global and local variables

Arguments of
Functions

of variable length

using keywords for optional
arguments

Functions using
Functions

the trapezoidal rule

Functional
Programming

Summary +
Assignments

Histograms

tallying the votes

global and local variables

Histograms

tallying the votes

global and local variables

Arguments of Functions

of variable length

using keywords for optional arguments

Arguments of Functions

of variable length

using keywords for optional
arguments

Functions using Functions

the trapezoidal rule

Functions using Functions

the trapezoidal rule

Functional Programming

Functional Programming

Summary + Assignments

Summary + Assignments

Arguments of Variable Length

defining more general functions

Consider the area computation of a square or rectangle.

The dimensions of a rectangle are length and width, but for a square we only need the length.

→ functions whose number of arguments is variable.

The arguments which may or may not appear when the function is called are collected in a tuple.

Python syntax:

```
def < name > ( < args > , * < tuple > ) :
```

The name of the `tuple` must

- ▶ appear after all other arguments `args`,
- ▶ and be preceded by `*`.

Histograms

tallying the votes
global and local variables

Arguments of
Functions

of variable length
using keywords for optional
arguments

Functions using
Functions

the trapezoidal rule

Functional
Programming

Summary +
Assignments

Arguments of Variable Length

defining more general functions

Consider the area computation of a square or rectangle.

The dimensions of a rectangle are length and width, but for a square we only need the length.

→ functions whose number of arguments is variable.

The arguments which may or may not appear when the function is called are collected in a tuple.

Python syntax:

```
def < name > ( < args > , * < tuple > ) :
```

The name of the `tuple` must

- ▶ appear after all other arguments `args`,
- ▶ and be preceded by `*`.

Histograms

tallying the votes
global and local variables

Arguments of
Functions

of variable length
using keywords for optional
arguments

Functions using
Functions

the trapezoidal rule

Functional
Programming

Summary +
Assignments

Arguments of Variable Length

defining more general functions

Consider the area computation of a square or rectangle.

The dimensions of a rectangle are length and width, but for a square we only need the length.

→ functions whose number of arguments is variable.

The arguments which may or may not appear when the function is called are collected in a tuple.

Python syntax:

```
def < name > ( < args > , * < tuple > ) :
```

The name of the `tuple` must

- ▶ appear after all other arguments `args`,
- ▶ and be preceded by `*`.

Histograms

tallying the votes
global and local variables

Arguments of
Functions

of variable length
using keywords for optional
arguments

Functions using
Functions

the trapezoidal rule

Functional
Programming

Summary +
Assignments

Arguments of Variable Length

defining more general functions

Consider the area computation of a square or rectangle.

The dimensions of a rectangle are length and width, but for a square we only need the length.

→ functions whose number of arguments is variable.

The arguments which may or may not appear when the function is called are collected in a tuple.

Python syntax:

```
def < name > ( < args > , * < tuple > ) :
```

The name of the `tuple` must

- ▶ appear after all other arguments `args`,
- ▶ and be preceded by `*`.

Histograms

tallying the votes
global and local variables

Arguments of
Functions

of variable length
using keywords for optional
arguments

Functions using
Functions

the trapezoidal rule

Functional
Programming

Summary +
Assignments

Arguments of Variable Length

defining more general functions

Consider the area computation of a square or rectangle.

The dimensions of a rectangle are length and width, but for a square we only need the length.

→ functions whose number of arguments is variable.

The arguments which may or may not appear when the function is called are collected in a tuple.

Python syntax:

```
def < name > ( < args > , * < tuple > ) :
```

The name of the `tuple` must

- ▶ appear after all other arguments `args`,
- ▶ and be preceded by `*`.

Histograms

tallying the votes
global and local variables

Arguments of
Functions

of variable length
using keywords for optional
arguments

Functions using
Functions

the trapezoidal rule

Functional
Programming

Summary +
Assignments

Arguments of Variable Length

defining more general functions

Consider the area computation of a square or rectangle.

The dimensions of a rectangle are length and width, but for a square we only need the length.

→ functions whose number of arguments is variable.

The arguments which may or may not appear when the function is called are collected in a tuple.

Python syntax:

```
def < name > ( < args > , * < tuple > ) :
```

The name of the `tuple` must

- ▶ appear after all other arguments `args`,
- ▶ and be preceded by `*`.

Histograms

tallying the votes
global and local variables

Arguments of
Functions

of variable length
using keywords for optional
arguments

Functions using
Functions

the trapezoidal rule

Functional
Programming

Summary +
Assignments

Arguments of Variable Length

defining more general functions

Consider the area computation of a square or rectangle.

The dimensions of a rectangle are length and width, but for a square we only need the length.

→ functions whose number of arguments is variable.

The arguments which may or may not appear when the function is called are collected in a tuple.

Python syntax:

```
def < name > ( < args > , * < tuple > ) :
```

The name of the `tuple` must

- ▶ appear after all other arguments `args`,
- ▶ and be preceded by `*`.

Histograms

tallying the votes
global and local variables

Arguments of
Functions

of variable length
using keywords for optional
arguments

Functions using
Functions

the trapezoidal rule

Functional
Programming

Summary +
Assignments

Area of Square or Rectangle

```
def area ( length , *width ) :
    "returns area of rectangle"
    if len(width) == 0 :                # square
        return length**2
    else :                              # rectangle
        return length*width[0]
```

Observe the different meanings of * !

```
print 'area of square or rectangle'
L = input('give length : ')
W = input('give width : ')
if W == 0 :
    a = area(L)
else :
    a = area(L,W)
print 'the area is', a
```

Histograms

tallying the votes
global and local variables

Arguments of Functions

of variable length
using keywords for optional
arguments

Functions using Functions

the trapezoidal rule

Functional Programming

Summary + Assignments

Area of Square or Rectangle

```
def area ( length , *width ) :
    "returns area of rectangle"
    if len(width) == 0 :                # square
        return length**2
    else :                              # rectangle
        return length*width[0]
```

Observe the different meanings of * !

```
print 'area of square or rectangle'
L = input('give length : ')
W = input('give width : ')
if W == 0 :
    a = area(L)
else :
    a = area(L,W)
print 'the area is', a
```

Histograms

tallying the votes
global and local variables

Arguments of Functions

of variable length
using keywords for optional
arguments

Functions using Functions

the trapezoidal rule

Functional Programming

Summary + Assignments

Outline

Histograms

tallying the votes

global and local variables

Arguments of Functions

of variable length

using keywords for optional arguments

Functions using Functions

the trapezoidal rule

Functional Programming

Summary + Assignments

MCS 260 L-14

28 September
2007

Histograms

tallying the votes

global and local variables

Arguments of Functions

of variable length

using keywords for optional
arguments

Functions using Functions

the trapezoidal rule

Functional Programming

Summary + Assignments

Using Keywords

for optional arguments

If arguments are optional, then we may identify the extra arguments of a function with keywords.

Instead of `a = area(L,W)`
we require `a = area(L,width=W)`.

Python syntax:

```
def < f > ( < a > , * < t > , ** < dict > ) :
```

The name of the dictionary `dict` must

- ▶ appear at the very end of the arguments,
- ▶ and be preceded by `**`.

Histograms

tallying the votes
global and local variables

Arguments of Functions

of variable length
using keywords for optional
arguments

Functions using Functions

the trapezoidal rule

Functional Programming

Summary + Assignments

Using Keywords

for optional arguments

If arguments are optional, then we may identify the extra arguments of a function with keywords.

Instead of `a = area(L,W)`
we require `a = area(L,width=W)`.

Python syntax:

```
def < f > ( < a > , * < t > , ** < dict > ) :
```

The name of the dictionary `dict` must

- ▶ appear at the very end of the arguments,
- ▶ and be preceded by `**`.

Histograms

tallying the votes
global and local variables

Arguments of Functions

of variable length
using keywords for optional arguments

Functions using Functions

the trapezoidal rule

Functional Programming

Summary + Assignments

Using Keywords

for optional arguments

If arguments are optional, then we may identify the extra arguments of a function with keywords.

Instead of `a = area(L,W)`
we require `a = area(L,width=W)`.

Python syntax:

```
def < f > ( < a > , * < t > , ** < dict > ) :
```

The name of the dictionary `dict` must

- ▶ appear at the very end of the arguments,
- ▶ and be preceded by `**`.

Histograms

tallying the votes
global and local variables

Arguments of
Functions

of variable length
using keywords for optional
arguments

Functions using
Functions

the trapezoidal rule

Functional
Programming

Summary +
Assignments

Optional Arguments as Keywords

MCS 260 L-14

28 September
2007

```
def area ( length , **width ):  
    "returns area of rectangle"  
    if len(width) == 0:                # square  
        return length**2  
    else:                               # rectangle  
        a = length  
        for each in width:  
            a *= width[each]  
        return a
```

observe the access to the dictionary ...

```
# input of L and W omitted  
if W == 0:  
    a = area(L)  
else:  
    a = area(L,width=W)  
print 'the area is', a
```

Calling area(L,W) no longer possible.

Histograms

tallying the votes
global and local variables

Arguments of
Functions

of variable length
using keywords for optional
arguments

Functions using
Functions

the trapezoidal rule

Functional
Programming

Summary +
Assignments

Optional Arguments as Keywords

MCS 260 L-14

28 September
2007

```
def area ( length , **width ):  
    "returns area of rectangle"  
    if len(width) == 0:                # square  
        return length**2  
    else:                              # rectangle  
        a = length  
        for each in width:  
            a *= width[each]  
        return a
```

observe the access to the dictionary ...

```
# input of L and W omitted
```

```
if W == 0:
```

```
    a = area(L)
```

```
else:
```

```
    a = area(L,width=W)
```

```
print 'the area is', a
```

Calling `area(L,W)` no longer possible.

Histograms

tallying the votes
global and local variables

Arguments of
Functions

of variable length
using keywords for optional
arguments

Functions using
Functions

the trapezoidal rule

Functional
Programming

Summary +
Assignments

Outline

Histograms

tallying the votes

global and local variables

Arguments of Functions

of variable length

using keywords for optional arguments

Functions using Functions

the trapezoidal rule

Functional Programming

Summary + Assignments

MCS 260 L-14

28 September
2007

Histograms

tallying the votes

global and local variables

Arguments of Functions

of variable length

using keywords for optional
arguments

Functions using Functions

the trapezoidal rule

Functional Programming

Summary +
Assignments

Functions using Functions

the trapezoidal rule

MCS 260 L-14

28 September
2007

To approximate the integral of a function $f(x)$ over $[a, b]$, the trapezoidal rule is

$$\int_a^b f(x) dx \approx \frac{1}{2}(f(a) + f(b))(b - a).$$

Geometrically, we approximate the area under $f(x)$ for $x \in [a, b]$ by the area of a trapezium, with base $[a, b]$ and heights $f(a)$ and $f(b)$.

A function as argument of a Python function, template:

```
def < rule > ( < f > , < a > , < b > ) :  
    "integrate function f over [a,b]"
```

Histograms

tallying the votes
global and local variables

Arguments of
Functions

of variable length
using keywords for optional
arguments

Functions using
Functions

the trapezoidal rule

Functional
Programming

Summary +
Assignments

Functions using Functions

the trapezoidal rule

MCS 260 L-14

28 September
2007

To approximate the integral of a function $f(x)$ over $[a, b]$, the trapezoidal rule is

$$\int_a^b f(x) dx \approx \frac{1}{2}(f(a) + f(b))(b - a).$$

Geometrically, we approximate the area under $f(x)$ for $x \in [a, b]$ by the area of a trapezium, with base $[a, b]$ and heights $f(a)$ and $f(b)$.

A function as argument of a Python function, template:

```
def < rule > ( < f > , < a > , < b > ) :  
    "integrate function f over [a,b]"
```

Histograms

tallying the votes
global and local variables

Arguments of
Functions

of variable length
using keywords for optional
arguments

Functions using
Functions

the trapezoidal rule

Functional
Programming

Summary +
Assignments

Functions using Functions

the trapezoidal rule

MCS 260 L-14

28 September
2007

To approximate the integral of a function $f(x)$ over $[a, b]$, the trapezoidal rule is

$$\int_a^b f(x) dx \approx \frac{1}{2}(f(a) + f(b))(b - a).$$

Geometrically, we approximate the area under $f(x)$ for $x \in [a, b]$ by the area of a trapezium, with base $[a, b]$ and heights $f(a)$ and $f(b)$.

A function as argument of a Python function, template:

```
def < rule > ( < f > , < a > , < b > ) :  
    "integrate function f over [a,b]"
```

Histograms

tallying the votes
global and local variables

Arguments of
Functions

of variable length
using keywords for optional
arguments

Functions using
Functions

the trapezoidal rule

Functional
Programming

Summary +
Assignments

The Trapezoidal Rule in Python

MCS 260 L-14

28 September
2007

```
def traprule(f,a,b):  
    "trapezoidal rule for f(x) over [a,b]"  
    return (b-a)*(f(a) + f(b))/2
```

```
import math  
s = 'integrating exp() over '  
print s + '[a,b]'  
a = input('give a : '  
b = input('give b : '  
y = traprule(math.exp,a,b)  
print s + '[%.1E,%.1E] : ' % (a,b)  
print 'the approximation : %.15E' % y  
e = math.exp(b) - math.exp(a)  
print ' the exact value : %.15E' % e
```

Histograms

tallying the votes
global and local variables

Arguments of
Functions

of variable length
using keywords for optional
arguments

Functions using
Functions

the trapezoidal rule

Functional
Programming

Summary +
Assignments

The Trapezoidal Rule in Python

MCS 260 L-14

28 September
2007

```
def traprule(f,a,b):
    "trapezoidal rule for f(x) over [a,b]"
    return (b-a)*(f(a) + f(b))/2

import math
s = 'integrating exp() over '
print s + '[a,b]'
a = input('give a : ')
b = input('give b : ')
y = traprule(math.exp,a,b)
print s + '[%.1E,%.1E] : ' % (a,b)
print 'the approximation : %.15E' % y
e = math.exp(b) - math.exp(a)
print ' the exact value : %.15E' % e
```

Histograms

tallying the votes
global and local variables

Arguments of
Functions

of variable length
using keywords for optional
arguments

Functions using
Functions

the trapezoidal rule

Functional
Programming

Summary +
Assignments

The Trapezoidal Rule in Python

MCS 260 L-14

28 September
2007

```
def traprule(f,a,b):
    "trapezoidal rule for f(x) over [a,b]"
    return (b-a)*(f(a) + f(b))/2

import math
s = 'integrating exp() over '
print s + '[a,b]'
a = input('give a : ')
b = input('give b : ')
y = traprule(math.exp,a,b)
print s + '[%.1E,%.1E] : ' % (a,b)
print 'the approximation : %.15E' % y
e = math.exp(b) - math.exp(a)
print ' the exact value : %.15E' % e
```

Histograms

tallying the votes
global and local variables

Arguments of
Functions

of variable length
using keywords for optional
arguments

Functions using
Functions

the trapezoidal rule

Functional
Programming

Summary +
Assignments

Running `traprule.py` at the prompt `$`

Running `traprule.py` at the prompt `$`

```
$ python traprule.py
integrating exp() over [a,b]
give a : 0
give b : 1
integrating exp() over [0.0E+00,1.0E+00] :
the approximation : 1.859140914229523E+00
  the exact value : 1.718281828459045E+00
```

Using functions as parameters to other functions allows to write more *generic* functions.

Example: finding the minimum or maximum in a list, use comparison function (`>` or `<`) as argument.

Histograms

tallying the votes
global and local variables

Arguments of
Functions

of variable length
using keywords for optional
arguments

Functions using
Functions

the trapezoidal rule

Functional
Programming

Summary +
Assignments

Running `traprule.py`

Running `traprule.py` at the prompt `$`

```
$ python traprule.py
integrating exp() over [a,b]
give a : 0
give b : 1
integrating exp() over [0.0E+00,1.0E+00] :
the approximation : 1.859140914229523E+00
  the exact value : 1.718281828459045E+00
```

Using functions as parameters to other functions allows to write more *generic* functions.

Example: finding the minimum or maximum in a list, use comparison function (`>` or `<`) as argument.

Histograms

tallying the votes
global and local variables

Arguments of
Functions

of variable length
using keywords for optional
arguments

Functions using
Functions

the trapezoidal rule

Functional
Programming

Summary +
Assignments

Running `traprule.py`

Running `traprule.py` at the prompt `$`

```
$ python traprule.py
integrating exp() over [a,b]
give a : 0
give b : 1
integrating exp() over [0.0E+00,1.0E+00] :
the approximation : 1.859140914229523E+00
  the exact value : 1.718281828459045E+00
```

Using functions as parameters to other functions allows to write more *generic* functions.

Example: finding the minimum or maximum in a list, use comparison function (`>` or `<`) as argument.

Histograms

tallying the votes
global and local variables

Arguments of
Functions

of variable length
using keywords for optional
arguments

Functions using
Functions

the trapezoidal rule

Functional
Programming

Summary +
Assignments

Functional Programming

Lisp

Early high level programming languages like C are heavily influenced by *the Von Neumann architecture*.

What this means is that the programmer is aware of the internal workings of the computer. For example, a skilled C programmer knows the distinction between the contents of a memory cell and its address.

Advantages and criticisms:

- + the programmer has great power and flexibility
- the description of algorithms is independent of computers

Except for recursion, we know already enough of Python to apply functional programming.

Histograms

tallying the votes
global and local variables

Arguments of Functions

of variable length
using keywords for optional
arguments

Functions using Functions

the trapezoidal rule

Functional Programming

Summary +
Assignments

Functional Programming

Lisp

Early high level programming languages like C are heavily influenced by *the Von Neumann architecture*.

What this means is that the programmer is aware of the internal workings of the computer. For example, a skilled C programmer knows the distinction between the contents of a memory cell and its address.

Advantages and criticisms:

- + the programmer has great power and flexibility
- the description of algorithms is independent of computers

Except for recursion, we know already enough of Python to apply functional programming.

Histograms

tallying the votes
global and local variables

Arguments of Functions

of variable length
using keywords for optional
arguments

Functions using Functions

the trapezoidal rule

Functional Programming

Summary +
Assignments

Functional Programming

Lisp

Early high level programming languages like C are heavily influenced by *the Von Neumann architecture*.

What this means is that the programmer is aware of the internal workings of the computer. For example, a skilled C programmer knows the distinction between the contents of a memory cell and its address.

Advantages and criticisms:

- + the programmer has great power and flexibility
- the description of algorithms is independent of computers

Except for recursion, we know already enough of Python to apply functional programming.

Histograms

tallying the votes
global and local variables

Arguments of Functions

of variable length
using keywords for optional
arguments

Functions using Functions

the trapezoidal rule

Functional Programming

Summary +
Assignments

Functional Programming

Lisp

Early high level programming languages like C are heavily influenced by *the Von Neumann architecture*.

What this means is that the programmer is aware of the internal workings of the computer. For example, a skilled C programmer knows the distinction between the contents of a memory cell and its address.

Advantages and criticisms:

- + the programmer has great power and flexibility
- the description of algorithms is independent of computers

Except for recursion, we know already enough of Python to apply functional programming.

Histograms

tallying the votes
global and local variables

Arguments of Functions

of variable length
using keywords for optional
arguments

Functions using Functions

the trapezoidal rule

Functional Programming

Summary +
Assignments

Functional Programming

Lisp

MCS 260 L-14

28 September
2007

Early high level programming languages like C are heavily influenced by *the Von Neumann architecture*.

What this means is that the programmer is aware of the internal workings of the computer. For example, a skilled C programmer knows the distinction between the contents of a memory cell and its address.

Advantages and criticisms:

- + the programmer has great power and flexibility
- the description of algorithms is independent of computers

Except for recursion, we know already enough of Python to apply functional programming.

Histograms

tallying the votes
global and local variables

Arguments of
Functions

of variable length
using keywords for optional
arguments

Functions using
Functions

the trapezoidal rule

Functional
Programming

Summary +
Assignments

Functional Programming

Lisp

MCS 260 L-14

28 September
2007

Early high level programming languages like C are heavily influenced by *the Von Neumann architecture*.

What this means is that the programmer is aware of the internal workings of the computer. For example, a skilled C programmer knows the distinction between the contents of a memory cell and its address.

Advantages and criticisms:

- + the programmer has great power and flexibility
- the description of algorithms is independent of computers

Except for recursion, we know already enough of Python to apply functional programming.

Histograms

tallying the votes
global and local variables

Arguments of
Functions

of variable length
using keywords for optional
arguments

Functions using
Functions

the trapezoidal rule

Functional
Programming

Summary +
Assignments

Functional Programming

Lisp

MCS 260 L-14

28 September
2007

Early high level programming languages like C are heavily influenced by *the Von Neumann architecture*.

What this means is that the programmer is aware of the internal workings of the computer. For example, a skilled C programmer knows the distinction between the contents of a memory cell and its address.

Advantages and criticisms:

- + the programmer has great power and flexibility
- the description of algorithms is independent of computers

Except for recursion, we know already enough of Python to apply functional programming.

Histograms

tallying the votes
global and local variables

Arguments of
Functions

of variable length
using keywords for optional
arguments

Functions using
Functions

the trapezoidal rule

Functional
Programming

Summary +
Assignments

Functional Programming

Lisp

MCS 260 L-14

28 September
2007

Early high level programming languages like C are heavily influenced by *the Von Neumann architecture*.

What this means is that the programmer is aware of the internal workings of the computer. For example, a skilled C programmer knows the distinction between the contents of a memory cell and its address.

Advantages and criticisms:

- + the programmer has great power and flexibility
- the description of algorithms is independent of computers

Except for recursion, we know already enough of Python to apply functional programming.

Histograms

tallying the votes
global and local variables

Arguments of
Functions

of variable length
using keywords for optional
arguments

Functions using
Functions

the trapezoidal rule

Functional
Programming

Summary +
Assignments

Summary + Assignments

MCS 260 L-14

28 September
2007

We covered in the lecture:

- ▶ more of chapter 5 in *Making Use of Python*;
- ▶ pages 606-607 in *The Art & Craft of Computing*.

Assignments:

1. Simulate a coin toss in a Python program, applying `random.randint(0,1)` at least a thousand times. Count the number of 0s and 1s. Is Python's coin fair?
2. Modify the vote tally to include abstain votes.
3. Extend the area function into the volume computation of a cube, or general parallelepiped. For a cube, only one parameter will be given, otherwise, the user must specify length, width, and height.

Homework will be collected on Monday 8 October.

Histograms

tallying the votes
global and local variables

Arguments of
Functions

of variable length
using keywords for optional
arguments

Functions using
Functions

the trapezoidal rule

Functional
Programming

Summary +
Assignments