

# MAT 305: Mathematical Computing

## Functions in computer programming

John Perry

University of Southern Mississippi

Spring 2017

# Outline

- 1 Functions
- 2 Functions and arguments
- 3 Indeterminates as arguments
- 4 Returning values
- 5 Pseudocode
- 6 Scripting
- 7 Summary

Functions

Functions and  
arguments

Indeterminates  
as arguments

Returning  
values

Pseudocode

Scripting

Summary

Functions

Functions and  
arguments

Indeterminates  
as arguments

Returning  
values

Pseudocode

Scripting

Summary

# Outline

- 1 Functions
- 2 Functions and arguments
- 3 Indeterminates as arguments
- 4 Returning values
- 5 Pseudocode
- 6 Scripting
- 7 Summary

# Functions?

Functions

Functions and  
arguments

Indeterminates  
as arguments

Returning  
values

Pseudocode

Scripting

Summary

- **function:** a sequence of statements organized as one command
  - may return one or more values
- names in other languages
  - C family: “functions”
  - Pascal family: “procedures” (no result) or “functions” (result)
  - object-oriented languages: “methods” or “features”

# Why functions?

- avoid retyping code
  - many patterns repeated
  - same behavior, different data
- organization, abstraction
- easier to read, maintain

# Defining a function

```
def name( argument1=default1 , argument2=default2 , ... ):  
    statement1  
    statement2  
    ...
```

where

- *name* is an identifier
- *arguments* (optional) are identifiers
- *defaults* (optional) are default values for the corresponding arguments

# Defining a function

```
def name( argument1=default1 , argument2=default2 , ... ) :  
    statement1  
    statement2  
    ...
```

where

- *name* is an identifier
- *arguments* (optional) are identifiers
- *defaults* (optional) are default values for the corresponding arguments

*not optional:*

- *:, (), def*
- *at least one statement*
- *indent all statements in function*

# Calling a function

- once  $f$  is defined, call using  $f()$
- supply data for arguments without default values



## Example

```
def greetings():  
    # greet the user  
    print 'Greetings!'
```

- name of function is greetings

*terrible choice; do not use*

- no arguments
- one statement
- one comment (# greet the user)
  - statement useful for programmers but computer ignores

# Example

Try it!

```
sage: def greetings():  
        # greet the user  
        print 'Greetings!'  
sage: greetings()  
Greetings!
```

Functions

Functions and  
arguments

Indeterminates  
as arguments

Returning  
values

Pseudocode

Scripting

Summary

# Outline

- ① Functions
- ② Functions and arguments
- ③ Indeterminates as arguments
- ④ Returning values
- ⑤ Pseudocode
- ⑥ Scripting
- ⑦ Summary

# Arguments?

**argument:** a placeholder for data

**scope:** name visible only inside function where it is defined

- data still exists outside function
- modifying argument does not modify original data, but creates new data
  - *caveat:* contents of lists and sets can be modified
- value of data forgotten immediately after function concludes

# Example

Functions

Functions and  
arguments

Indeterminates  
as arguments

Returning  
values

Pseudocode

Scripting

Summary

```
def greetings(name='Leonhard Euler'):  
    print 'Greetings,', name
```

- name of function is greetings
- one argument, name
  - default value: 'Leonhard Euler'

## Example

Try it!

```
sage: def greetings(name='Leonhard Euler'):
      print 'Greetings,', name
```

```
sage: greetings()
Greetings, Leonhard Euler
```

Functions

Functions and  
arguments

Indeterminates  
as arguments

Returning  
values

Pseudocode

Scripting

Summary

## Example

Try it!

```
sage: def greetings(name='Leonhard Euler'):  
      print 'Greetings,', name
```

```
sage: greetings()  
Greetings, Leonhard Euler
```

```
sage: greetings('Pythagoras')  
Greetings, Pythagoras
```

Functions

Functions and  
arguments

Indeterminates  
as arguments

Returning  
values

Pseudocode

Scripting

Summary

## Example

Try it!

```
sage: def greetings(name='Leonhard Euler'):  
      print 'Greetings,', name
```

```
sage: greetings()  
Greetings, Leonhard Euler
```

```
sage: greetings('Pythagoras')  
Greetings, Pythagoras
```

```
sage: greetings(name='Pythagoras')  
Greetings, Pythagoras
```

Functions

Functions and  
arguments

Indeterminates  
as arguments

Returning  
values

Pseudocode

Scripting

Summary



## Example

Try it!

```
sage: def greetings(name='Leonhard Euler'):  
      print 'Greetings,', name
```

```
sage: greetings()  
Greetings, Leonhard Euler
```

```
sage: greetings('Pythagoras')  
Greetings, Pythagoras
```

```
sage: greetings(name='Pythagoras')  
Greetings, Pythagoras
```

```
sage: greetings(pi)  
Greetings, pi
```

# Warning 1

Functions

Functions and  
arguments

Indeterminates  
as arguments

Returning  
values

Pseudocode

Scripting

Summary

## Don't use uninitialized identifiers

```
sage: def greetings(name='Leonhard Euler'):  
        print 'Greetings,', name
```

```
sage: greetings(Pythagoras) oops: no quotes!  
...Output deleted...
```

```
NameError: name 'Pythagoras' is not defined
```

## Warning 2

Functions

Functions and  
arguments

Indeterminates  
as arguments

Returning  
values

Pseudocode

Scripting

Summary

*Scope* implies name does not exist outside hello

```
sage: def greetings(name='Leonhard Euler'):
      print 'Greetings,', name
```

```
sage: greetings('Pythagoras')
```

```
Hello, Pythagoras
```

```
sage: name
```

```
...Output deleted...
```

```
NameError: name 'name' is not defined
```

*name out of scope*

## Warning 3

Functions

Functions and  
arguments

Indeterminates  
as arguments

Returning  
values

Pseudocode

Scripting

Summary

*Scope* implies name forgotten once hello concludes

```
sage: def greetings(name='Leonhard Euler'):  
      print 'Greetings,', name
```

```
sage: greetings('Pythagoras')
```

```
Hello, Pythagoras
```

```
sage: greetings()
```

```
Hello, Leonhard Euler
```

*name has value 'Leonhard Euler' again*

## Warning 4

Can change value inside function, but value outside function remains the same

```
sage: def mischievous_hello(name='world'):  
        name = 'loser!'  
        print 'Hello,', name
```

Functions

Functions and  
arguments

Indeterminates  
as arguments

Returning  
values

Pseudocode

Scripting

Summary

## Warning 4

Functions

Functions and  
arguments

Indeterminates  
as arguments

Returning  
values

Pseudocode

Scripting

Summary

Can change value inside function, but value outside function remains the same

```
sage: def mischievous_hello(name='world'):  
      name = 'loser!'  
      print 'Hello,', name
```

```
sage: print_name = 'Dr. Perry'
```

```
sage: mischievous_hello(print_name)
```

```
Hello, loser! value of name changed in function
```

## Warning 4

Functions

Functions and  
arguments

Indeterminates  
as arguments

Returning  
values

Pseudocode

Scripting

Summary

Can change value inside function, but value outside function remains the same

```
sage: def mischievous_hello(name='world'):  
      name = 'loser!'  
      print 'Hello,', name
```

```
sage: print_name = 'Dr. Perry'
```

```
sage: mischievous_hello(print_name)
```

```
Hello, loser! value of name changed in function
```

```
sage: print_name
```

```
'Dr. Perry' value of print_name unchanged
```

## Warning 5

If defaults are not given to arguments, you must supply something

```
sage: def greetings(name):  
        print 'Greetings,', name
```

```
sage: greetings()
```

...*Output deleted*...

```
TypeError: goodbye() takes exactly 1 argument (0  
given)
```

Functions

Functions and  
arguments

Indeterminates  
as arguments

Returning  
values

Pseudocode

Scripting

Summary



## Warning 5

Functions

Functions and  
arguments

Indeterminates  
as arguments

Returning  
values

Pseudocode

Scripting

Summary

If defaults are not given to arguments, you must supply something

```
sage: def greetings(name):  
        print 'Greetings,', name
```

```
sage: greetings()
```

...*Output deleted*...

```
TypeError: goodbye() takes exactly 1 argument (0  
given)
```

```
sage: greetings('cruel world')
```

```
Greetings, cruel world
```

Functions

Functions and  
arguments

Indeterminates  
as arguments

Returning  
values

Pseudocode

Scripting

Summary

# Outline

- 1 Functions
- 2 Functions and arguments
- 3 Indeterminates as arguments**
- 4 Returning values
- 5 Pseudocode
- 6 Scripting
- 7 Summary

# A mathematical problem

Given function  $f$  and real number  $a$   
find line tangent to  $f$  at  $x = a$

Functions

Functions and  
arguments

Indeterminates  
as arguments

Returning  
values

Pseudocode

Scripting

Summary

# A mathematical problem

Given function  $f$  and real number  $a$   
find line tangent to  $f$  at  $x = a$

Seems easy enough:

```
sage: def tangent_line(f, a):  
    # point-slope form of a line  
    b = f(a)  
    df(x) = diff(f,x)  
    m = df(a)  
    result = m*(x - a) + b  
    print 'The line tangent to', f,  
    print 'at x =', a, 'is',  
    print result
```

## Trouble is...

...it works, but complains!

```
sage: tangent_line(x**2, 1)
```

...*Output deleted*...

DeprecationWarning: Substitution using  
function-call syntax and unnamed arguments is  
deprecated and will be removed from a future release  
of Sage; you can use named arguments instead, like  
EXPR(x=..., y=...)

...*Output deleted*...

The line tangent to  $x^2$  at  $x = 1$  is  $2*x - 1$

## Trouble is...

...it works, but complains!

```
sage: tangent_line(x**2, 1)
```

...*Output deleted*...

DeprecationWarning: Substitution using  
function-call syntax and unnamed arguments is  
deprecated and will be removed from a future release  
of Sage; you can use named arguments instead, like  
EXPR(x=..., y=...)

...*Output deleted*...

The line tangent to  $x^2$  at  $x = 1$  is  $2*x - 1$

Problematic line:  $b = f(a)$

- $f$  has value  $x**2$ , an *expression*
- $b = f(a)$  wants  $f$  to be a *function*

# Solution #1

Functions

Functions and  
arguments

Indeterminates  
as arguments

Returning  
values

Pseudocode

Scripting

Summary

Define  $f$  as function, call `tangent_line()` w/ $f$

```
sage: f(x) = x**2
```

```
sage: tangent_line(f, 1)
```

The line tangent to  $x \mapsto x^2$  at  $x = 1$  is  $2*x - 1$

...but this is undesirable. *Why?*

# Solution #1

Define  $f$  as function, call `tangent_line()` w/ $f$

```
sage: f(x) = x**2
```

```
sage: tangent_line(f, 1)
```

The line tangent to  $x \mapsto x^2$  at  $x = 1$  is  $2x - 1$

...but this is undesirable. *Why?*

Inconvenient, really: `tangent_line(x**2, 1)` would be useful.



## Solution #2

Specify variable of substitution in function:

```
sage: def tangent_line(f, a):
        # point-slope form of a line
        b = f(x=a)
        df(x) = diff(f,x)
        m = df(a)
        result = m*(x - a) + b
        print 'The line tangent to', f,
        print 'at x =', a, 'is',
        print result
```

Functions

Functions and  
arguments

Indeterminates  
as arguments

Returning  
values

Pseudocode

Scripting

Summary

## Solution #2

Specify variable of substitution in function:

```
sage: def tangent_line(f, a):  
    # point-slope form of a line  
    b = f(x=a)  
    df(x) = diff(f,x)  
    m = df(a)  
    result = m*(x - a) + b  
    print 'The line tangent to', f,  
    print 'at x =', a, 'is',  
    print result
```

```
sage: tangent_line(x**2, 1)
```

The line tangent to  $x \mapsto x^2$  at  $x = 1$  is  $2x - 1$

...but this also is undesirable. *Why?*

## Solution #2

Specify variable of substitution in function:

```
sage: def tangent_line(f, a):  
      # point-slope form of a line  
      b = f(x=a)  
      df(x) = diff(f,x)  
      m = df(a)  
      result = m*(x - a) + b  
      print 'The line tangent to', f,  
      print 'at x =', a, 'is',  
      print result
```

```
sage: tangent_line(x**2, 1)
```

The line tangent to  $x \mapsto x^2$  at  $x = 1$  is  $2x - 1$   
...but this also is undesirable. *Why?*

```
sage: var('t')
```

```
sage: tangent_line(t**2, 1)
```

The line tangent to  $x \mapsto t^2$  at  $x = 1$  is  $t^2$

- Functions
- Functions and arguments
- Indeterminates as arguments
- Returning values
- Pseudocode
- Scripting
- Summary

## Solution #3

Redefine expression as function:

```
sage: def tangent_line(f, a):  
    # redefine f  
    f(x) = f  
    # point-slope form of a line  
    b = f(a)  
    df(x) = diff(f,x)  
    m = df(a)  
    result = m*(x - a) + b  
    print 'The line tangent to', f,  
    print 'at x =', a, 'is',  
    print result
```

## Solution #3

Redefine expression as function:

```
sage: def tangent_line(f, a):  
      # redefine f  
      f(x) = f  
      # point-slope form of a line  
      b = f(a)  
      df(x) = diff(f,x)  
      m = df(a)  
      result = m*(x - a) + b  
      print 'The line tangent to', f,  
      print 'at x =', a, 'is',  
      print result
```

```
sage: tangent_line(x**2, 1)
```

The line tangent to  $t \mapsto t^2$  at  $x = 1$  is  $2*x - 1$   
...better, but insufficient. *Why?*

## Solution #3

Redefine expression as function:

```
sage: def tangent_line(f, a):  
      # redefine f  
      f(x) = f  
      # point-slope form of a line  
      b = f(a)  
      df(x) = diff(f,x)  
      m = df(a)  
      result = m*(x - a) + b  
      print 'The line tangent to', f,  
      print 'at x =', a, 'is',  
      print result
```

```
sage: tangent_line(x**2, 1)
```

The line tangent to  $t \mapsto t^2$  at  $x = 1$  is  $2*x - 1$

...better, but insufficient. *Why?*

Who wants to see  $t \mapsto x$ ?

## Best (?) solution

Redefine argument as function:

```
sage: def tangent_line(f, a, x=x):  
    # redefine f  
    f(x) = f  
    # point-slope form of a line  
    b = f(a)  
    df(x) = diff(f,x)  
    m = df(a)  
    result = m*(x - a) + b  
    print 'The line tangent to', f,  
    print 'at', x, '=' , a, 'is',  
    print result
```

## Best (?) solution

Redefine argument as function:

```
sage: def tangent_line(f, a, x=x):
      # redefine f
      f(x) = f
      # point-slope form of a line
      b = f(a)
      df(x) = diff(f,x)
      m = df(a)
      result = m*(x - a) + b
      print 'The line tangent to', f,
      print 'at', x, '=', a, 'is',
      print result
sage: tangent_line(t**2, 1)
The line tangent to t |--> t^2 at t = 1 is 2*t - 1
```



Functions

Functions and  
arguments

Indeterminates  
as arguments

Returning  
values

Pseudocode

Scripting

Summary

# Outline

- ① Functions
- ② Functions and arguments
- ③ Indeterminates as arguments
- ④ Returning values**
- ⑤ Pseudocode
- ⑥ Scripting
- ⑦ Summary

## Returning values?

- Functions compute values
- Often want to work with what we've computed
  - `print` command unhelpful
  - copy and paste annoying

### Example

Compute derivative of special function, want to use it to:

- graph tangent line
- analyze concavity
- identify optimum values
- ...

# The return command

Functions

Functions and  
arguments

Indeterminates  
as arguments

Returning  
values

Pseudocode

Scripting

Summary

`return` *value1*, *value2*, ...

- reports the data values *value1*, *value2*, etc.
- only works inside functions
- only reports to caller of current function

# Modify implementation

Functions

Functions and  
arguments

Indeterminates  
as arguments

Returning  
values

Pseudocode

Scripting

Summary

```
sage: def tangent_line(f, a, x=x):  
      # redefine f  
      f(x) = x  
      # point-slope form of a line  
      b = f(x=a)  
      df(x) = diff(f,x)  
      m = df(a)  
      result = m*(x - a) + b  
      return result
```

# Modify implementation

Functions

Functions and  
arguments

Indeterminates  
as arguments

Returning  
values

Pseudocode

Scripting

Summary

```
sage: def tangent_line(f, a, x=x):
        # redefine f
        f(x) = x
        # point-slope form of a line
        b = f(x=a)
        df(x) = diff(f,x)
        m = df(a)
        result = m*(x - a) + b
        return result
sage: tangent_line(t**2, 1, t)
2*t - 1
```

Functions

Functions and  
arguments

Indeterminates  
as arguments

Returning  
values

Pseudocode

Scripting

Summary

# Outline

- 1 Functions
- 2 Functions and arguments
- 3 Indeterminates as arguments
- 4 Returning values
- 5 Pseudocode**
- 6 Scripting
- 7 Summary

# Pseudocode?

## description of algorithm

- many formats
- format independent of computer language
- prefer mathematics to programming
  - “ $i$ th element of  $L$ ” or “ $L_i$ ”, not  $L[i-1]$

## Our pseudocode format

**algorithm** *name*

**inputs**

*input1*  $\in$  *domain1*

*input2*, *description of type*

...

**outputs**

*output1*, *relationship to inputs*

*output2*, *relationship to inputs*

...

**do**

*English or mathematical statement 1*

*English or mathematical statement 2*

...



## Our pseudocode format

**algorithm** *name*

**inputs**

*input1*  $\in$  *domain1*

*input2*, *description of type*

...

**outputs**

*output1*, *relationship to inputs*

*output2*, *relationship to inputs*

...

**do**

*English or mathematical statement 1*

*English or mathematical statement 2*

...

*Try it now on the given problem*

## Example problem

Write a program to compute the line tangent to  $f(x)$  at  $x = x_0$ .

- Write pseudocode answering:
  - ① What inputs will we need?
    - domain of each input (what set/type of object)
  - ② What outputs do we expect?
    - inputs' purpose & relationship to output
  - ③ How do we use the inputs to generate the output?
    - think step-by-step
    - do a sample problem:  $f(x) = x^2$ ,  $x_0 = 3$
    - think about possible errors errors
- Implement pseudocode

# Example pseudocode

**algorithm** *tangent\_line*

**inputs**

$f$ , a function of a variable  $x$

$x_0 \in \mathbb{R}$

## Example pseudocode

**algorithm** *tangent\_line*

**inputs**

$f$ , a function of a variable  $x$

$x_0 \in \mathbb{R}$

**outputs**

the line tangent to  $f(x)$  at  $x = x_0$

Functions

Functions and  
arguments

Indeterminates  
as arguments

Returning  
values

Pseudocode

Scripting

Summary

## Example pseudocode

**algorithm** *tangent\_line*

**inputs**

$f$ , a function of a variable  $x$

$x_0 \in \mathbb{R}$

**outputs**

the line tangent to  $f(x)$  at  $x = x_0$

**do**

— *We need two things for a line: a point  $(x_0, y_0)$  and the slope  $m$*

## Example pseudocode

**algorithm** *tangent\_line*

**inputs**

*f*, a function of a variable  $x$

$x_0 \in \mathbb{R}$

**outputs**

the line tangent to  $f(x)$  at  $x = x_0$

**do**

– *We need two things for a line: a point  $(x_0, y_0)$  and the slope  $m$*

Let  $y_0 = f(x_0)$

– *Use Calculus to find  $m$*

## Example pseudocode

**algorithm** *tangent\_line*

**inputs**

$f$ , a function of a variable  $x$

$x_0 \in \mathbb{R}$

**outputs**

the line tangent to  $f(x)$  at  $x = x_0$

**do**

– *We need two things for a line: a point  $(x_0, y_0)$  and the slope  $m$*

Let  $y_0 = f(x_0)$

– *Use Calculus to find  $m$*

Let  $fderiv = f'(x)$

Let  $m = fderiv(x_0)$

## Example pseudocode

**algorithm** *tangent\_line*

**inputs**

*f*, a function of a variable *x*

$x_0 \in \mathbb{R}$

**outputs**

the line tangent to  $f(x)$  at  $x = x_0$

**do**

– *We need two things for a line: a point  $(x_0, y_0)$  and the slope  $m$*

Let  $y_0 = f(x_0)$

– *Use Calculus to find  $m$*

Let  $fderiv = f'(x)$

Let  $m = fderiv(x_0)$

– *Point-slope form:  $y - y_0 = m(x - x_0)$*

Let  $line = m(x - x_0) + y_0$



## Example pseudocode

**algorithm** *tangent\_line*

**inputs**

*f*, a function of a variable *x*

$x_0 \in \mathbb{R}$

**outputs**

the line tangent to  $f(x)$  at  $x = x_0$

**do**

– *We need two things for a line: a point  $(x_0, y_0)$  and the slope  $m$*

Let  $y_0 = f(x_0)$

– *Use Calculus to find  $m$*

Let  $fderiv = f'(x)$

Let  $m = fderiv(x_0)$

– *Point-slope form:  $y - y_0 = m(x - x_0)$*

Let  $line = m(x - x_0) + y_0$

**return** *line*

## Example implementation

Compare to implementation:

```
def tangent_line(f, a, x=x):  
    # redefine f  
    f(x) = x  
    # point-slope form of a line  
    b = f(x=a)  
    df(x) = diff(f,x)  
    m = df(a)  
    result = m*(x - a) + b  
    return result
```

Functions

Functions and  
arguments

Indeterminates  
as arguments

Returning  
values

Pseudocode

Scripting

Summary

## Example implementation

Compare to implementation:

```
def tangent_line(f, a, x=x):  
    # redefine f  
    f(x) = x  
    # point-slope form of a line  
    b = f(x=a)  
    df(x) = diff(f,x)  
    m = df(a)  
    result = m*(x - a) + b  
    return result
```

*Specify x as input to pseudocode?*

- not inherent to *problem*, but to *programming*, so
- do not include in pseudocode, but
- this is arguable

# Example run

- Functions
- Functions and arguments
- Indeterminates as arguments
- Returning values
- Pseudocode
- Scripting
- Summary

*Behold: the power of symbolic computation!*

```
sage: var('a b c')
```

```
sage: tangent_line(a*x**2 + b*x + c, 1, x)  
(2*a + b)*(x - 1) + a + b + c
```

## Combine with plots

We show the plots of  $e^x$  and its tangent line at  $x = 0$

```
sage: f = e**t
sage: tanline = tangent_line(f, 0, t)
sage: fplot = plot(f, -2, 2, color='black',
                  thickness=2)
sage: lineplot = plot(tanline, -2, 2,
                     linestyle='dashed')
sage: fplot + lineplot
```

Functions

Functions and  
arguments

Indeterminates  
as arguments

Returning  
values

Pseudocode

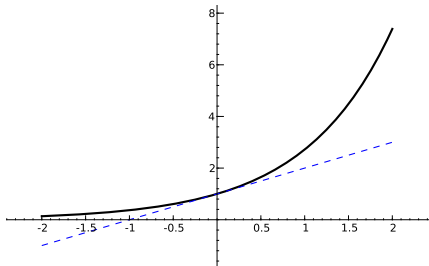
Scripting

Summary

## Combine with plots

We show the plots of  $e^x$  and its tangent line at  $x = 0$

```
sage: f = e**t
sage: tanline = tangent_line(f, 0, t)
sage: fplot = plot(f, -2, 2, color='black',
                  thickness=2)
sage: lineplot = plot(tanline, -2, 2,
                     linestyle='dashed')
sage: fplot + lineplot
```



# Combining functions

Functions

Functions and  
arguments

Indeterminates  
as arguments

Returning  
values

Pseudocode

Scripting

Summary

It would be nice to have a function that graphs an arbitrary  $f(x)$  and its tangent line at  $x = x_0$ . Options include:

# Combining functions

It would be nice to have a function that graphs an arbitrary  $f(x)$  and its tangent line at  $x = x_0$ . Options include:

- Repeat previous commands for each  $f$  and each  $x_0$

*a lot of work!*



# Combining functions

It would be nice to have a function that graphs an arbitrary  $f(x)$  and its tangent line at  $x = x_0$ . Options include:

- Repeat previous commands for each  $f$  and each  $x_0$   
*a lot of work!*
- Encapsulate commands in another function

# Pseudocode

**algorithm** *plot\_function\_and\_tangent*

**inputs**

$f$ , a function of a variable  $x$

$x_0 \in \mathbb{R}$

**outputs**

the plot of  $f(x)$  and the line tangent to  $f$  at  $x = x_0$

**do**

Let  $P_1$  be the plot of  $f(x)$  in a neighborhood of  $x_0$

Let  $g(x)$  be the line tangent to  $f$  at  $x_0$  *Already solved!*

Let  $P_2$  be the plot of  $g(x)$  in the same neighborhood of  $x_0$

**return**  $P_1$  and  $P_2$  combined

# Implementation

```
def plot_function_and_tangent(f, x0, x=x,  
                             xmin=-2, xmax=2):  
  
    # plots f(x) and line tangent to f at x0  
    # over [ xmin, xmax ];  
    # returns combination of these plots  
  
    P1 = plot(f, xmin, xmax, rgbcolor='black',  
              thickness=2)  
  
    # next line reuses previous code  
    g = tangent_line(f, x0, x)  
  
    P2 = plot(g, xmin, xmax, linestyle='dashed')  
    return P1 + P2
```

*Whitespace*

*distinguishes*

*different*

*tasks*

# Examples

```
sage: def plot_function_and_tangent...
```

Functions

Functions and  
arguments

Indeterminates  
as arguments

Returning  
values

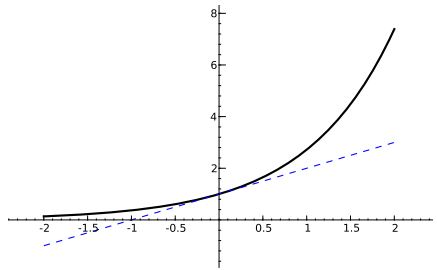
Pseudocode

Scripting

Summary

# Examples

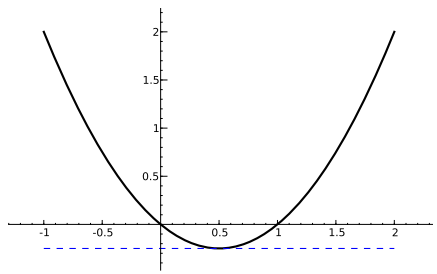
```
sage: def plot_function_and_tangent...  
sage: plot_function_and_tangent(e**x,x)
```



- Functions
- Functions and arguments
- Indeterminates as arguments
- Returning values
- Pseudocode
- Scripting
- Summary

# Examples

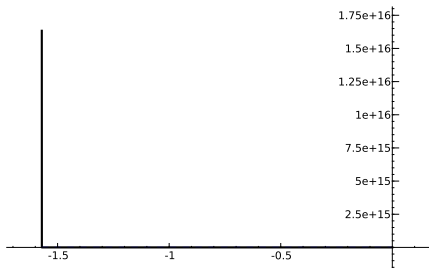
```
sage: def plot_function_and_tangent...  
sage: plot_function_and_tangent(x**2-x,0.5,xmin=-1,  
xmax=2)
```



- Functions
- Functions and arguments
- Indeterminates as arguments
- Returning values
- Pseudocode
- Scripting
- Summary

## Examples

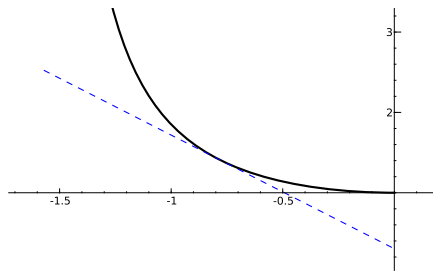
```
sage: def plot_function_and_tangent...  
sage: plot_function_and_tangent(sec(x), -pi/2,  
      xmin=-pi/2, xmax=0)
```



*ouch. need to adjust ymax*

# Examples

```
sage: def plot_function_and_tangent...  
sage: good_sec = plot_function_and_tangent(  
        sec(x),  
        xmin=-pi/2,x0=-pi/4,xmax=0)  
sage: show(good_sec,ymax=3)
```



*Note: sec(x) does not work in older versions, apparently because its derivative is not computed*



Functions

Functions and  
arguments

Indeterminates  
as arguments

Returning  
values

Pseudocode

Scripting

Summary

# Outline

- ① Functions
- ② Functions and arguments
- ③ Indeterminates as arguments
- ④ Returning values
- ⑤ Pseudocode
- ⑥ Scripting**
- ⑦ Summary

# Scripting?

`script` a sequence of Sage statements saved to a file

- write, save
- load at later date
- no need to type, re-type statements

Functions

Functions and  
arguments

Indeterminates  
as arguments

Returning  
values

Pseudocode

Scripting

Summary

# Scripting?

`script` a sequence of Sage statements saved to a file

- write, save
- load at later date
- no need to type, re-type statements

Let's script our `tangent_line()` function.

## Creating a script

- Back to 'Files'
- Type `calc_utils.sage` in "Filename" box
- Choose "Sage Code (.sage)" in "Create" drop-down
- put `tangent_line()` code in new file
- Save, close
  - `calc_utils.sage` appears in list of files!

## Using a script

- Create new Sage worksheet
  - or open old one
- “Attach” the script:  
`sage: attach calc_utils.sage`
- Try it:  
`sage: tangent_line(x**2, 1)`  
`2*x - 1`

## Using a script

- Create new Sage worksheet
  - or open old one
- “Attach” the script:  
`sage: attach calc_utils.sage`
- Try it:  
`sage: tangent_line(x**2, 1)`  
`2*x - 1`
- You have now “extended” Sage!
- Can use `tangent_line()` in any worksheet, or even in other scripts, using `%attach` directive

Functions

Functions and  
arguments

Indeterminates  
as arguments

Returning  
values

Pseudocode

Scripting

Summary

# Outline

- 1 Functions
- 2 Functions and arguments
- 3 Indeterminates as arguments
- 4 Returning values
- 5 Pseudocode
- 6 Scripting
- 7 Summary

## Summary

- Functions collect several Sage statements into one
  - organizes solutions to problems
  - abstraction makes problem-solving easier
- define using `def (...):`
- Functions receive *arguments* as data
  - can specify default values
  - function does not change arguments, *but...*
  - elements of collections can be changed
- Return value(s) using `return`



- Scripts save functions for later usage

Functions

Functions and  
arguments

Indeterminates  
as arguments

Returning  
values

Pseudocode

Scripting

Summary