DM536
Introduction to Programming

Peter Schneider-Kamp

petersk@imada.sdu.dk

http://imada.sdu.dk/~petersk/DM536/

VARIABLES, EXPRESSIONS & STATEMENTS

Values and Types

- Values = basic data objects   42   23.0   "Hello!"
- Types = classes of values   integer   float   string

- Values can be printed:
  - print <value>   print "Hello!"

- Types can be determined:
  - type(<value>)   type(23.0)

- Values and types can be compared:
  - <value> == <value>   type(3) == type(3.0)

Variables

- variable = name that refers to a value
- program state = mapping from variables to values

- values are *assigned* to variables using "=":
  - <var> = <value> b = 4

- the value referred to by a variable can be printed:
  - print <var> print b

- the type of a variable is the type of the value it refers to:
  - type(b) == type(4)

Variable Names

- start with a letter (convention: a-z)
- contain letters a-z and A-Z, digits 0-9, and underscore "_"

- can be any such name except for 31 reserved names:

and	del	from	not	while
as	elif	global	or	with
assert	else	if	pass	yield
break	except	import	print	
class	exec	in	raise	
continue	finally	is	return	
def	for	lambda	try	

Multiple Assignment

- variables can be assigned to different values at different times:
  - Example: x = 3

    x = 4
  - Instructions are executed top-to bottom => x refers to 4

- be careful, e.g., when exchanging values serially:
  - Example: x = y

    y = x
  - later x and y refer to the same value
  - Solution 1 (new variable): z = y; y = x; x = z
  - Solution 2 (parallel assign.): x, y = y, x

Operators & Operands

- Operators represent computations: + * - / **
  - Example: 23+19 day+month*30 2**6-22

- Addition "+", Multiplication "*", Subtraction "-" as usual
- Exponentiation "**": x**y means x^y

- Division "/" rounds down integers:
  - Example 1: 21/42 has value 0, NOT 0.5
  - Example 2: 21.0/42 has value 0.5
  - Example 3: 21/42.0 has value 0.5

Expressions

- Expressions can be:
  - Values: 42 23.0 "Hej med dig!"
  - Variables: x y name1234
  - built from operators: 19+23.0 x**2+y**2
- grammar rule:
  - <expr> => <value> |
    <var> |
    <expr> <operator> <expr> |
    (<expr>)
- every expression has a value:
  - replace variables by their values
  - perform operations

Operator Precedence

- expressions are evaluated left-to-right
  - Example: 64 - 24 + 2 == 42
- BUT: like in mathematics, "*" binds more strongly than "+"
  - Example: 2 + 8 * 5 == 42
- parentheses have highest precedence: 64 - (24 + 2) == 38
- PEMDAS rule:
  - Parentheses "(<expr>)"
  - Exponentiation "**"
  - Multiplication "*" and Division "/"
  - Addition "+" and Subtraction "-"

String Operations

- Addition “+” works on strings:
  - Example 1: print "Hello w" + "orld!"
  - Example 2: print "4" + "2"
- Multiplication “*” works on strings, if 2nd operands is integer:
  - Example: print "Hej!" * 10
- Subtraction “-”, Division “/”, and Exponentiation “**” do NOT work on strings

Debugging Expressions

- most beginners struggle with common Syntax Errors:
  - check that all parentheses and quotes are closed
  - check that operators have two operands
  - sequential instruction should start on the same column or be separated by a semicolon ";"

- common Runtime Error due to misspelling variable names:
  - Example:

```
a = input(); b = input()
reslut = a**b+b**a
print result
```

Statements

- instructions in Python are called *statements*

- so far we know 2 different statements:
  - print statement: `print "Ciao!"`
  - assignments “=”: `c = a**2+b**2`

- as a grammar rule:

```
<stmt>   =>    print <expr>          |
               <var> = <expr>        |
               <expr>
```

Comments

- programs are not only written, they are also read
- document program to provide intuition:
  - Example 1: c = sqrt(a**2+b**2) # use Pythagoras
  - Example 2: x, y = y, x # swap x and y
- all characters after the comment symbol "#" are ignored
  - Example: x = 23 #+19

    results in x referring to the value 23

CALLING & DEFINING FUNCTIONS

Calling Functions

- so far we have seen three different *function calls*:
  - input(): reads a value from the keyboard
  - sqrt(x): computes the square root of x
  - type(x): returns the type of the value of x

- in general, a function call is also an expression:
  - <expr> => … | <function>(<arg$_1$>, …, <arg$_n$>)
  - Example 1:
    ```
    x = input()
    print type(x)
    ```
  - Example 2:
    ```
    from math import log
    print log(4398046511104, 2)
    ```

Importing Modules

- we imported the sqrt function from the math module:

```
from math import sqrt
```

- alternatively, we can import the whole module:

```
import math
```

- using the built-in function "dir(x)" we see math's functions:

acos	cos	floor	log	sin
asin	cosh	fmod	log10	sinh
atan	degrees	frexp	modf	sqrt
atan2	exp	hypot	pow	tan
ceil	fabs	ldexp	radians	tanh

- access using "math.<function>": c = math.sqrt(a**2+b**2)

The Math Module

- contains 25 functions (trigonometric, logarithmic, …):
  - Example:
    ```
    x = input()
    print math.sin(x)**2+math.cos(x)**2
    ```
- contains 2 constants (math.e and math.pi):
  - Example: `print math.sin(math.pi / 2)`
- contains 3 meta data (__doc__, __file__, __name__):
  - print math.__doc__
  - print math.frexp.__doc__
  - print type.__doc__

Type Conversion Functions

- Python has pre-defined functions for converting values
- int(x): converts x into an integer
  - Example 1: int("1234") == int(1234.9999)
  - Example 2: int(-3.999) == -3
- float(x): converts x into a float
  - Example 1: float(42) == float("42")
  - Example 2: float("Hej!") results in Runtime Error
- str(x): converts x into a string
  - Example 1: str(23+19) == "42"
  - Example 2: str(type(42)) == "<type 'int'>"

DEFINING FUNCTIONS

Function Definitions

- functions are defined using the following grammar rule:

  <func.def> => def <function>(<arg$_1$>, …, <arg$_n$>):
  <instr$_1$>; …; <instr$_k$>

- can be used to reuse code:
  - Example:

```
def pythagoras():
    c = math.sqrt(a**2+b**2)
    print "Result:", c
a = 3; b = 4; pythagoras()
a = 7; b = 15; pythagoras()
```

- functions are values: `type(pythagoras)`

Functions Calling Functions

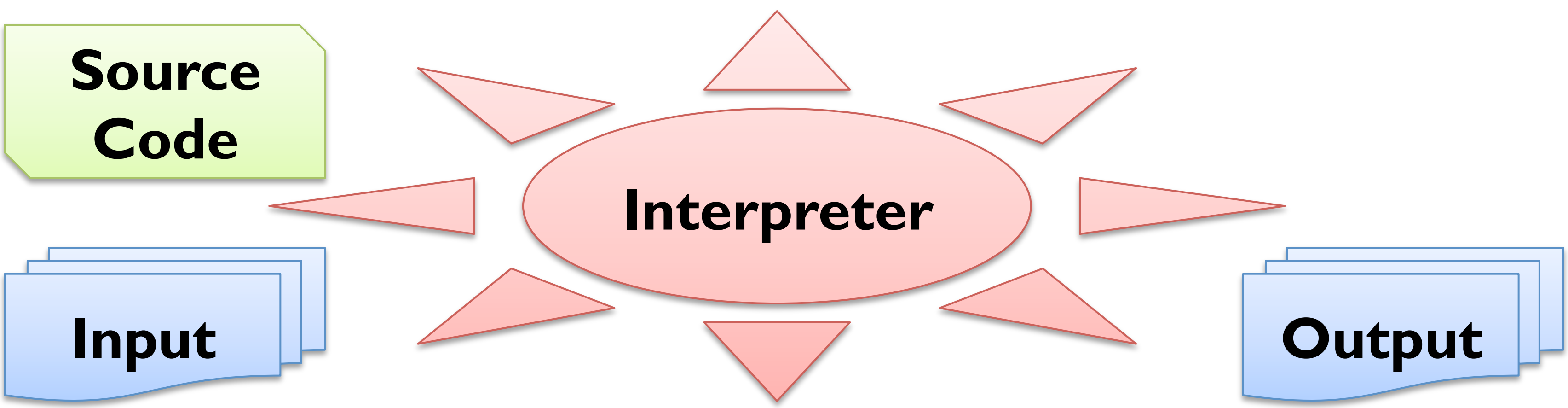
- functions can call other functions

- Example:

```
def white():
    print " #" * 8
def black():
    print "# " * 8
def all():
    white();  black();  white();  black()
    white();  black();  white();  black()
all()
```

Executing Programs (Revisited)

- Program stored in a file (*source code* file)
- Instructions in this file executed top-to-bottom
- Interpreter executes each instruction



Functions Calling Functions

- functions can call other functions

- Example:

create new function variable “white”

```
def white():
    print " #" * 8
def black():
    print "# " * 8
def all():
    white(); black(); white(); black()
    white(); black(); white(); black()
all()
```

Functions Calling Functions

- functions can call other functions
- Example:

```
def white():
    print " #" * 8
def black():
    print "# " * 8
def all():
    white();  black();  white();  black()
    white();  black();  white();  black()
all()
```

create new function variable "black"

Functions Calling Functions

- functions can call other functions

- Example:

```
def white():
    print " #" * 8
def black():
    print "# " * 8
def all():
    white();  black();  white();  black()
    white();  black();  white();  black()
all()
```

create new function variable “all”

Functions Calling Functions

- functions can call other functions

- Example:

```
def white():
    print " #" * 8
def black():
    print "# " * 8
def all():
    white();  black();  white();  black()
    white();  black();  white();  black()
all()
```

call function “all”

Functions Calling Functions

- functions can call other functions

- Example:

```
def white():
    print " #" * 8
def black():
    print "# " * 8
def all():
    white();  black();  white();  black()
    white();  black();  white();  black()
all()
```

call function “white”

Functions Calling Functions

- functions can call other functions

- Example:

```
def white():
    print " #" * 8
def black():
    print "# " * 8
def all():
    white();  black();  white();  black()
    white();  black();  white();  black()
all()
```

print
" # # # # # # # #"

Functions Calling Functions

- functions can call other functions

- Example:

```
def white():
    print " #" * 8
def black():
    print "# " * 8
def all():
    white(); black(); white(); black()
    white(); black(); white(); black()
all()
```

call function “black”

Functions Calling Functions

- functions can call other functions

- Example:

```
def white():
    print " #" * 8
def black():
    print "# " * 8
def all():
    white();  black();  white();  black()
    white();  black();  white();  black()
all()
```

print
"# # # # # # # # "

Functions Calling Functions

- functions can call other functions

- Example:

```
def white():
    print " #" * 8
def black():
    print "# " * 8
def all():
    white(); black(); white(); black()
    white(); black(); white(); black()
all()
```

call function "white"

Functions Calling Functions

- functions can call other functions

- Example:

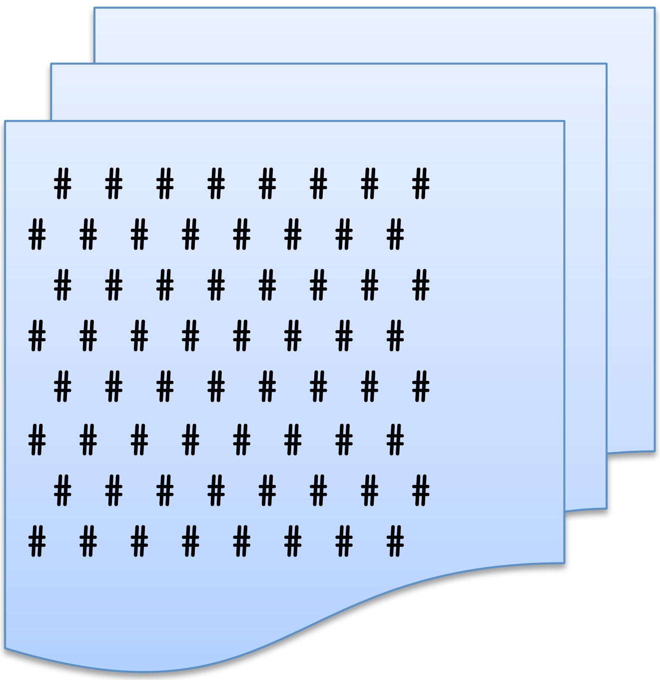
```
def white():
    print " #" * 8
def black():
    print "# " * 8
def all():
    white();  black();  white();  black()
    white();  black();  white();  black()
all()
```

print
" # # # # # # # #"

Functions Calling Functions

- functions can call other functions

- Example:




```
def white():
    print " #" * 8
def black():
    print "# " * 8
def all():
    white();  black();  white();  black()
    white();  black();  white();  black()
all()
```

Parameters and Arguments

- we have seen functions that need arguments:
  - math.sqrt(x) computes square root of x
  - math.log(x, base) computes logarithm of x w.r.t. base

- arguments are assigned to parameters of the function
  - Example:

```
def pythagoras():
    c = math.sqrt(a**2+b**2)
    print "Result:", c
a = 3; b = 4; pythagoras()
a = 7; b = 15; pythagoras()
```

Parameters and Arguments

- we have seen functions that need arguments:
  - math.sqrt(x) computes square root of x
  - math.log(x, base) computes logarithm of x w.r.t. base

- arguments are assigned to parameters of the function
  - Example:

```
def pythagoras(a, b):
    c = math.sqrt(a**2+b**2)
    print "Result:", c
a = 3; b = 4; pythagoras(a, b)
a = 7; b = 15; pythagoras(a, b)
```

Parameters and Arguments

- we have seen functions that need arguments:
  - math.sqrt(x) computes square root of x
  - math.log(x, base) computes logarithm of x w.r.t. base
- arguments are assigned to parameters of the function
  - Example:

```
def pythagoras(a, b):
    c = math.sqrt(a**2+b**2)
    print "Result:", c
pythagoras(3, 4)
pythagoras(7, 15)
```

Parameters and Arguments

- we have seen functions that need arguments:
  - math.sqrt(x) computes square root of x
  - math.log(x, base) computes logarithm of x w.r.t. base

- arguments are assigned to parameters of the function
  - Example:

```
def pythagoras(a, b):
    c = math.sqrt(a**2+b**2)
    print "Result:", c
pythagoras(3, 4)
pythagoras(2**3-1, 2**4-1)
```

Parameters and Arguments

- we have seen functions that need arguments:
  - math.sqrt(x) computes square root of x
  - math.log(x, base) computes logarithm of x w.r.t. base

- arguments are assigned to parameters of the function
  - Example:

```
def pythagoras(a, b):
    c = math.sqrt(a**2+b**2)
    print "Result:", c
pythagoras(3, 4)
x = 2**3-1; y = 2**4-1
pythagoras(x, y)
```

Variables are Local

- parameters and variables are local
- local = only available in the function defining them
- Example: in module math:

```
def sqrt(x):
    …
```

x local to math.sqrt

in our program:

```
def pythagoras(a, b):
    c = math.sqrt(a**2+b**2)
    print "Result:", c
x = 3;  y =4;  pythagoras(x, y)
```

a local to pythagoras

b local to pythagoras

c local to pythagoras

x,y local to __main__

Stack Diagrams

__main__	x	→	3
	y	→	4
pythagoras	a	→	3
	b	→	4
math.sqrt	x	→	25

Tracebacks

- stack structure printed on runtime error
- Example:

```
def broken(x):
    print x / 0
def caller(a, b):
    broken(a**b)
caller(2,5)
```

```
Traceback (most recent call last):
  File "test.py", line 5, in <module>
    caller(2,5)
  File "test.py", line 4, in caller
    broken(a**b)
  File "test.py", line 2, in broken
    print x/0
ZeroDivisionError: integer division or modulo by zero
```

Return Values

- we have seen functions that return values:
  - math.sqrt(x) returns the square root of x
  - math.log(x, base) returns the logarithm of x w.r.t. base

- What is the return value of our function pythagoras(a, b)?
- special value None returned, if no return value given (*void*)

- declare return value using return statement: return <expr>
- Example:

```
def pythagoras(a, b):
    c = math.sqrt(a**2+b**2)
    return c
print pythagoras(3, 4)
```

Motivation for Functions

- functions give names to blocks of code
  - easier to read
  - easier to debug
- avoid repetition
  - easier to make changes
- functions can be debugged separately
  - easier to test
  - easier to find errors
- functions can be reused (for other programs)
  - easier to write new programs

Debugging Function Definitions

- make sure you are using latest files (save, then run python -i)

- biggest problem for beginners is *indentation*
  - all lines on the same level must have the same indentation
  - mixing spaces and tabs is very dangerous
  - try to use only spaces – a good editor helps!

- do not forget to use “:” at end of first line
- indent body of function definition by e.g. 4 spaces

TURTLE WORLD & INTERFACE DESIGN

Turtle World

- available from
  - http://www.greenteapress.com/thinkpython/swampy/install.html
- basic elements of the library
  - can be imported using from TurtleWorld import *
  - w = TurtleWorld() creates new world w
  - t = Turtle() creates new turtle t
  - wait_for_user() can be used at the end of the program

Simple Repetition

- two basic commands to the turtle
  - fd(t, 100) advances turtle t by 100
  - lt(t) turns turtle t 90 degrees to the left
- drawing a square requires 4x drawing a line and turning left
  - fd(t,100); lt(t); fd(t,100); lt(t); fd(t,100); lt(t); fd(t,100); lt(t)
- simple repetition using for-loop for <var> in range(<expr>):
  <instr$_1$>; <instr$_2$>
- Example:

```
for i in range(4):
    print i
```

Simple Repetition

- two basic commands to the turtle
  - fd(t, 100) advances turtle t by 100
  - lt(t) turns turtle t 90 degrees to the left
- drawing a square requires 4x drawing a line and turning left
  - fd(t,100); lt(t); fd(t,100); lt(t); fd(t,100); lt(t); fd(t,100); lt(t)
- simple repetition using for-loop

```
for <var> in range(<expr>):
    <instr_1>;  <instr_2>
```

- Example:

```
for i in range(4):
    fd(t, 100)
    lt(t)
```