DM536
Introduction to Programming

Peter Schneider-Kamp

petersk@imada.sdu.dk

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

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 swampy.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)
```

Encapsulation

- **Idea:** wrap up a block of code in a function
  - documents use of this block of code
  - allows reuse of code by using parameters

- Example:

```
def square(t):
    for i in range(4):
        fd(t, 100)
        lt(t)
square(t)
u = Turtle(); rt(u); fd(u,10); lt(u);
square(u)
```

Generalization

- square(t) can be reused, but size of square is fixed
- **Idea:** generalize function by adding parameters
  - more flexible functionality
  - more possibilities for reuse
- Example 1:

```
def square(t, length):
    for i in range(4):
        fd(t, length)
        lt(t)
square(t, 100)
square(t, 50)
```

Generalization

- Example 2: replace square by regular polygon with n sides

```
def square(t, length):
    for i in range(4):
        fd(t, length)
        lt(t)
```

Generalization

- Example 2: replace square by regular polygon with n sides

```
def polygon(t, length):
    for i in range(4):
        fd(t, length)
        lt(t)
```

Generalization

- Example 2: replace square by regular polygon with n sides

```
def polygon(t, n, length):
    for i in range(n):
        fd(t, length)
        lt(t)
```

Generalization

- Example 2: replace square by regular polygon with n sides

```
def polygon(t, n, length):
    for i in range(n):
        fd(t, length)
        lt(t, 360/n)
```

Generalization

- Example 2: replace square by regular polygon with n sides

```
def polygon(t, n, length):
    angle = 360/n
    for i in range(n):
        fd(t, length)
        lt(t, angle)
```

Generalization

- Example 2: replace square by regular polygon with n sides

```
def polygon(t, n, length):
    angle = 360/n
    for i in range(n):
        fd(t, length)
        lt(t, angle)
polygon(t, 4, 100)
polygon(t, 6, 50)
```

Generalization

- Example 2: replace square by regular polygon with n sides

```
def polygon(t, n, length):
    angle = 360/n
    for i in range(n):
        fd(t, length)
        lt(t, angle)
polygon(t, n=4, length=100)
polygon(t, n=6, length=50)
```

Generalization

- Example 2: replace square by regular polygon with n sides

```
def polygon(t, n, length):
    angle = 360/n
    for i in range(n):
        fd(t, length)
        lt(t, angle)


square(t, 100)
```

Generalization

- Example 2: replace square by regular polygon with n sides

```
def polygon(t, n, length):
    angle = 360/n
    for i in range(n):
        fd(t, length)
        lt(t, angle)
def square(t, length):
    polygon(t, 4, length)
square(t, 100)
```

Interface Design

- **Idea:** interface = parameters + semantics + return value
- should be general (= easy to reuse)
- but as simple as possible (= easy to read and debug)

- Example:

```
def circle(t, r):
    circumference = 2*math.pi*r
    n = 10
    length = circumference / n
    polygon(t, n, length)
circle(t, 10)
circle(t, 100)
```

Interface Design

- **Idea:** interface = parameters + semantics + return value
- should be general (= easy to reuse)
- but as simple as possible (= easy to read and debug)

- Example:

```
def circle(t, r, n):
    circumference = 2*math.pi*r
#   n = 10
    length = circumference / n
    polygon(t, n, length)
circle(t, 10, 10)
circle(t, 100, 40)
```

Interface Design

- **Idea:** interface = parameters + semantics + return value
- should be general (= easy to reuse)
- but as simple as possible (= easy to read and debug)

- Example:

```
def circle(t, r):
    circumference = 2*math.pi*r
    n = int(circumference / 3) + 1
    length = circumference / n
    polygon(t, n, length)
circle(t, 10)
circle(t, 100)
```

Refactoring

- we want to be able to draw arcs
- Example:

```
def arc(t, r, angle):
    arc_length = 2*math.pi*r*angle/360
    n = int(arc_length / 3) + 1
    step_length = arc_length / n
    step_angle = float(angle) / n


    for i in range(n):
        fd(t, step_length)
        lt(t, step_angle)
```

Refactoring

- we want to be able to draw arcs
- Example:

```
def arc(t, r, angle):
    arc_length = 2*math.pi*r*angle/360
    n = int(arc_length / 3) + 1
    step_length = arc_length / n
    step_angle = float(angle) / n

def polyline(t, n, length, angle):
    for i in range(n):
        fd(t, length)
        lt(t, angle)
```

Refactoring

- we want to be able to draw arcs
- Example:

```
def arc(t, r, angle):
    arc_length = 2*math.pi*r*angle/360
    n = int(arc_length / 3) + 1
    step_length = arc_length / n
    step_angle = float(angle) / n
    polyline(t, n, step_length, step_angle)
def polyline(t, n, length, angle):
    for i in range(n):
        fd(t, length)
        lt(t, angle)
```

Refactoring

- we want to be able to draw arcs
- Example:

```
def polyline(t, n, length, angle):
    for i in range(n):
        fd(t, length)
        lt(t, angle)
```

Refactoring

- we want to be able to draw arcs
- Example:

```
def polyline(t, n, length, angle):
    for i in range(n):
        fd(t, length)
        lt(t, angle)
def polygon(t, n, length):
    angle = 360/n
    polyline(t, n, length, angle):
```

Refactoring

- we want to be able to draw arcs
- Example:

```
def arc(t, r, angle):
    arc_length = 2*math.pi*r*angle/360
    n = int(arc_length / 3) + 1
    step_length = arc_length / n
    step_angle = float(angle) / n
    polyline(t, n, step_length, step_angle)
```

Refactoring

- we want to be able to draw arcs
- Example:

```
def arc(t, r, angle):
    arc_length = 2*math.pi*r*angle/360
    n = int(arc_length / 3) + 1
    step_length = arc_length / n
    step_angle = float(angle) / n
    polyline(t, n, step_length, step_angle)
def circle(t, r):
    arc(t, r, 360)
```

Simple Iterative Development

- first structured approach to develop programs:
  1. write small program without functions
  2. encapsulate code in functions
  3. generalize functions (by adding parameters)
  4. repeat steps 1–3 until functions work
  5. refactor program (e.g. by finding similar code)

- copy & paste helpful
  - reduces amount of typing
  - no need to debug same code twice

Debugging Interfaces

- interfaces simplify testing and debugging

1. test if pre-conditions are given:
   - do the arguments have the right type?
   - are the values of the arguments ok?

2. test if the post-conditions are given:
   - does the return value have the right type?
   - is the return value computed correctly?

3. debug function, if pre- or post-conditions violated

CONDITIONAL EXECUTION

Boolean Expressions

- expressions whose value is either True or False
- logic operators for computing with Boolean values:
  - x and y — True if, and only if, x is True and y is True
  - x or y — True if (x is True or y is True)
  - not x — True if, and only if, x is False
- Python also treats numbers as Boolean expressions:
  - 0 — False
  - any other number — True
  - Please, do NOT use this feature!

Relational Operators

- relational operators are operators, whose value is Boolean

- important relational operators are:

	Example True	Example False
x < y	23 < 42	"World" < "Hej!"
x <= y	42 <= 42.0	int(math.pi) <= 2
x == y	42 == 42.0	type(2) == type(2.0)
x >= y	42 >= 42	"Hej!" >= "Hello"
x > y	"World" > "Hej!"	42 > 42

- remember to use "==" instead of "=" (assignment)!

Conditional Execution

- the if-then statement executes code only if a condition holds
- grammar rule:

  <if-then> => if <cond>:

  <instr$_1$>; …; <instr$_k$>

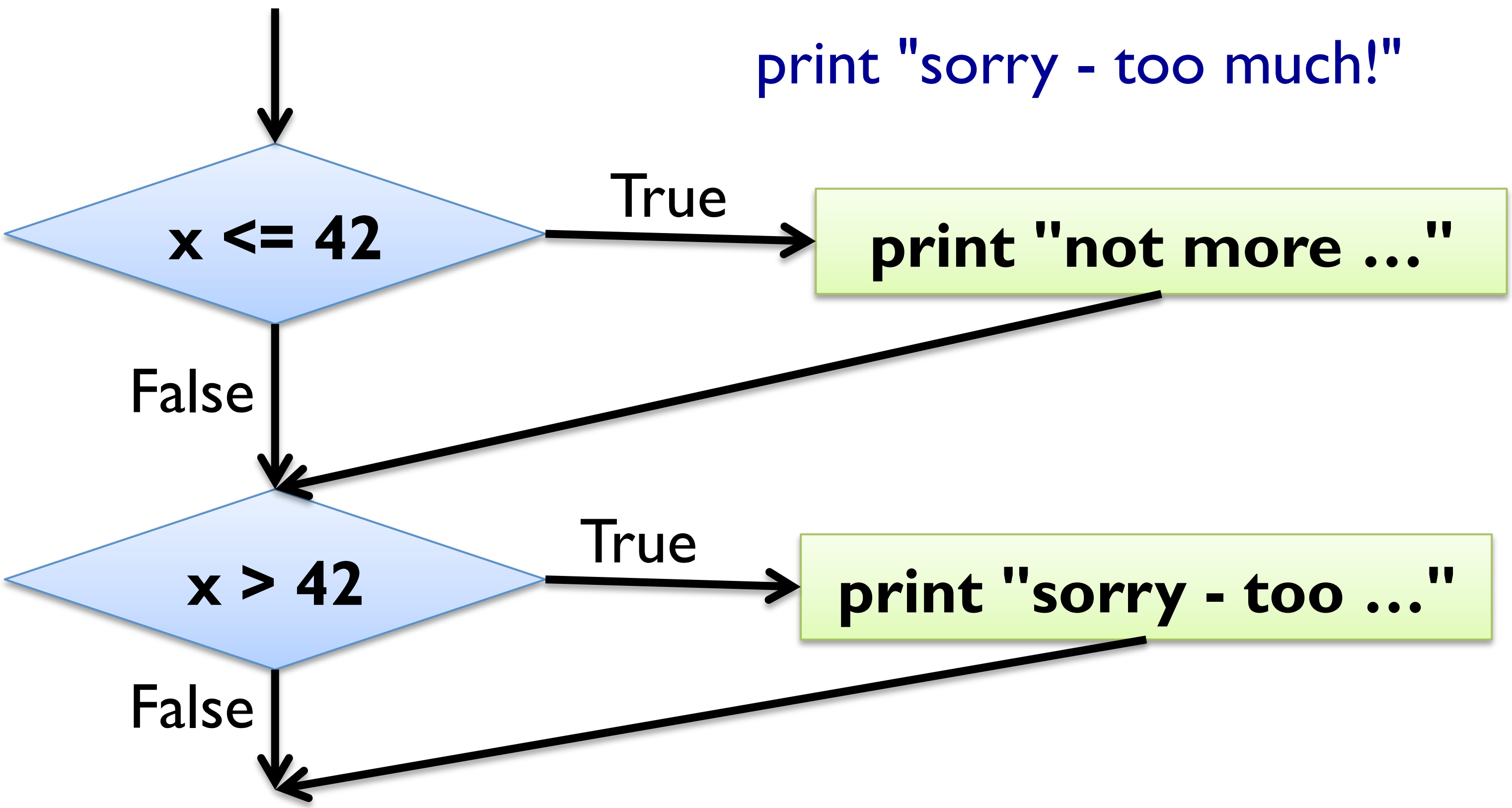
- Example:

```
if x <= 42:
    print "not more than the answer"
if x  >  42:
    print "sorry - too much!"
```

Control Flow Graph

- Example:

```
if x <= 42:
    print "not more than the answer"
if x  >  42:
    print "sorry - too much!"
```



Alternative Execution

- the if-then-else statement executes one of two code blocks
- grammar rule:

<if-then-else> => if <cond>:

<instr$_1$>; …; <instr$_k$>

else:

<instr'$_1$>; …; <instr'$_{k'}$>

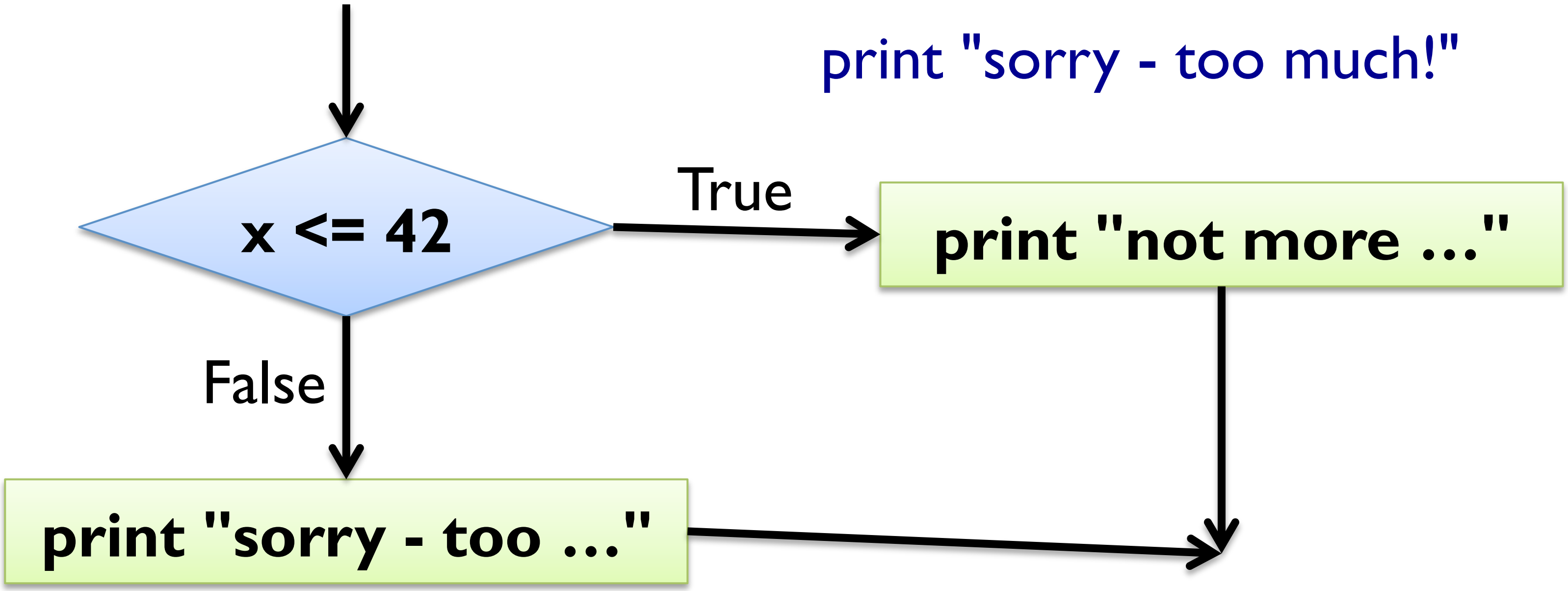
- Example:

```
if x <= 42:
    print "not more than the answer"
else:
    print "sorry - too much!"
```

Control Flow Graph

- Example:

```
if x <= 42:
    print "not more than the answer"
else:
    print "sorry - too much!"
```



Chained Conditionals

- alternative execution a special case of chained conditionals
- grammar rules:

  <if-chained> => if <cond$_1$>:

  <instr$_{1,1}$>; …; <instr$_{k1,1}$>

  elif <cond$_2$>:

  …

  else:

  <instr$_{1,m}$>; …; <instr$_{km,m}$>

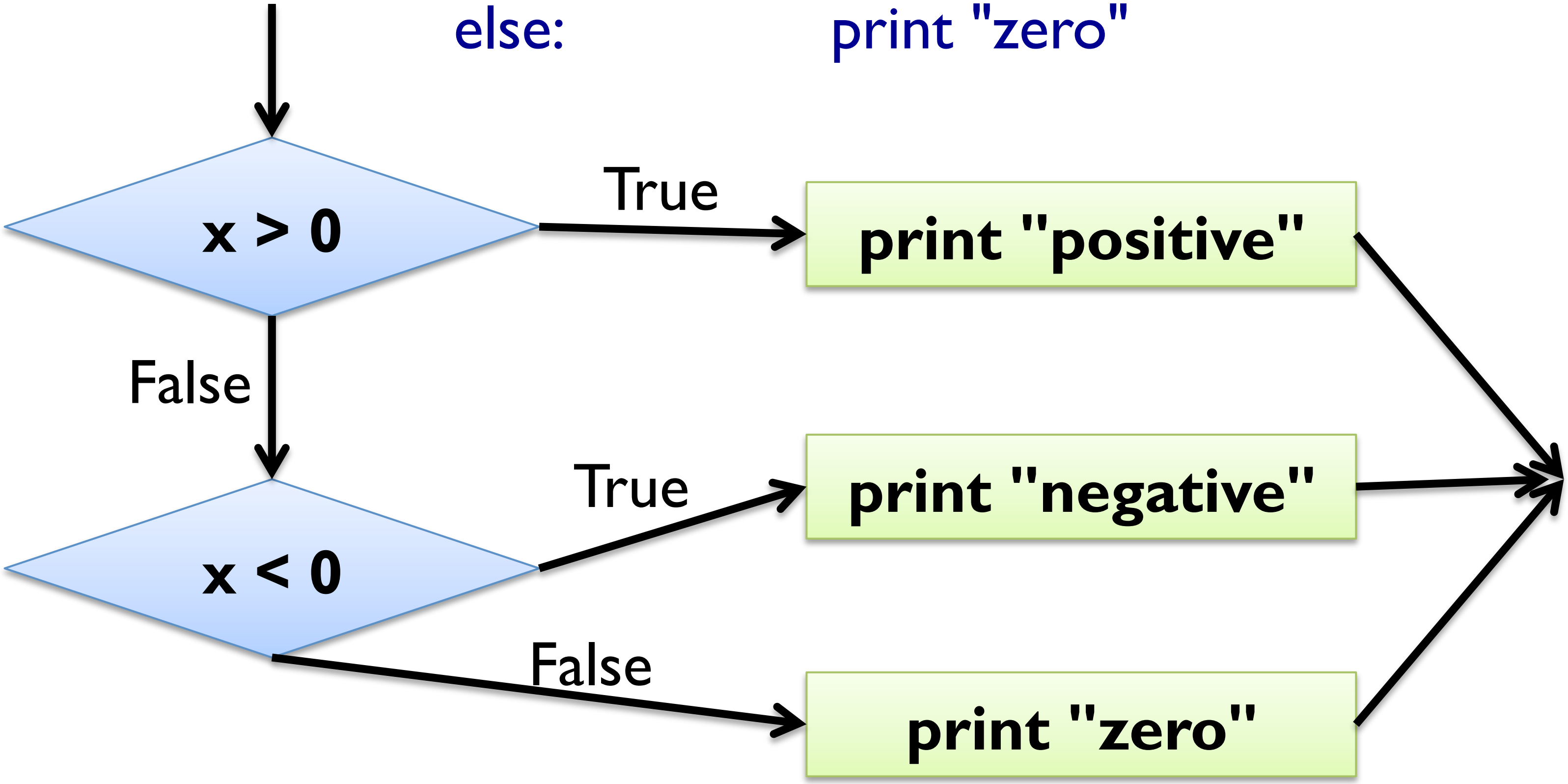
- Example:

```
if x > 0:     print "positive"
elif x < 0:   print "negative"
else:         print "zero"
```

Control Flow Diagram

- Example:

```
if x > 0:       print "positive"
elif x < 0:     print "negative"
else:           print "zero"
```



Nested Conditionals

- conditionals can be nested below conditionals:

```
x = input()
y = input()
if x > 0:
    if y > 0:      print "Quadrant 1"
    elif y < 0:    print "Quadrant 4"
    else:          print "positive x-Axis"
elif x < 0:
    if y > 0:      print "Quadrant 2"
    elif y < 0:    print "Quadrant 3"
    else:          print "negative x-Axis"
else:   print "y-Axis"
```

RECURSION

Recursion

- a function can call other functions
- a function can call **itself**
- such a function is called a *recursive* function

- Example 1:

```
def countdown(n):
    if n <= 0:
        print "Ka-Boooom!"
    else:
        print n, "seconds left!"
        countdown(n-1)
countdown(3)
```

Stack Diagrams for Recursion

Recursion

- a function can call other functions
- a function can call **itself**
- such a function is called a *recursive* function

- Example 2:

```
def polyline(t, n, length, angle):
    for i in range(n):
        fd(t, length)
        lt(t, angle)
```

Recursion

- a function can call other functions
- a function can call **itself**
- such a function is called a *recursive* function

- Example 2:

```
def polyline(t, n, length, angle):
    if n > 0:
        fd(t, length)
        lt(t, angle)
        polyline(t, n-1, length, angle)
```

Infinite Recursion

- base case = no recursive function call reached
- we say the function call *terminates*
  - Example 1: n == 0 in countdown / polyline

- infinite recursion = no base case is reached
- also called *non-termination*

- Example:

```
def infinitely_often():
    infinitely_often()
```

- Python has *recursion limit* 1000 – ask sys.getrecursionlimit()

Keyboard Input

- so far we only know input()
  - what happens when we enter Hello?
  - input() treats all input as Python expression <expr>
- for string input, use raw_input()
  - what happens when we enter 42?
  - raw_input() treats all input as string
- both functions can take one argument prompt
  - Example 1: `a = input("first side: ")`
  - Example 2: `name = raw_input("Your name:\n")`
  - "\n" denotes a new line: `print "Hello\nWorld\n!"`

Debugging using Tracebacks

- error messages in Python give important information:
  - where did the error occur?
  - what kind of error occurred?

- unfortunately often hard to localize real problem
- Example:

real problem (points to `factor = base_price / vat_price`)

error reported (points to `reverse_factor = 1 / factor`)

```
def determine_vat(base_price, vat_price):
    factor = base_price / vat_price
    reverse_factor = 1 / factor
    return reverse_factor - 1
print determine_vat(400, 500)
```

Debugging using Tracebacks

- error messages in Python give important information:
  - where did the error occur?
  - what kind of error occurred?

- unfortunately often hard to localize real problem
- Example:

```
def determine_vat(base_price, vat_price):
    factor = float(base_price) / vat_price
    reverse_factor = 1 / factor
    return reverse_factor - 1
print determine_vat(400, 500)
```

FRUITFUL FUNCTIONS

Return Values

- so far we have seen only functions with one or no return
- sometimes more than one return makes sense

- Example 1:

```
def sign(x):
    if x < 0:
        return -1
    elif x == 0:
        return 0
    else:
        return 1
```