DM550/DM857
Introduction to Programming

Peter Schneider-Kamp

petersk@imada.sdu.dk

http://imada.sdu.dk/~petersk/DM550/

http://imada.sdu.dk/~petersk/DM857/

Project Qualification Assessment

- first assessment on Monday, September 18, 12:15-14:00

- 3 assessments in total
- sum of points from all 3 assessments at least 50% of total

- in class assessment using your own computer
- please test BEFORE next Monday!
- Blackboard multiple choice
- Magic numbers generated using online python version at:

http://lynx.imada.sdu.dk/

Code Café

- manned Code Cafe for students
- first time Wednesday, September 6
- last time Wednesday, December 20
- closed in Week 42 (efterårsferie)

- Mondays, 15.00 – 17.00, Nicky Cordua Mattsson
- Wednesdays, 15.00 – 17.00, Troels Risum Vigsøe Frimer

- Nicky and Troels can help with any coding related issues
- issues have to be related to some IMADA course (fx this one)

GETTING YOUR HANDS DIRTY

Accessing Web Services

- any http URL can be retrieved using the `requests` module
- install using: `pip3 install requests`

- easy access to standard HTTP requests such as GET, POST, …

- Retrieve a web:

```
import requests
requests.get("http://www.sdu.dk/")
```

- Access a web service:

```
url="http://lynx.imada.sdu.dk/osrm/route/v1/driving/-73,40;-73,40.1"
print(requests.get(url).json()["routes"][0])
```

Jelling Stones to Little Mermaid

```
import requests

db = "http://dbpedia.org/"
stones = "Jelling_stones"
mermaid = "The_Little_Mermaid_(statue)"

stones = requests.get(db+"data/"+stones+".json").json()[db+"resource/"+stones]
mermaid = requests.get(db+"data/"+mermaid+".json").json()[db+"resource/"+mermaid]

stones_long = str(stones["http://www.w3.org/2003/01/geo/wgs84_pos#long"][0]["value"])
stones_lat = str(stones["http://www.w3.org/2003/01/geo/wgs84_pos#lat"][0]["value"])
mermaid_long = str(mermaid["http://www.w3.org/2003/01/geo/wgs84_pos#long"][0]["value"])
mermaid_lat = str(mermaid["http://www.w3.org/2003/01/geo/wgs84_pos#lat"][0]["value"])

url = "http://lynx.imada.sdu.dk/osrm/route/v1/driving/"
res = requests.get(url+stones_long+","+stones_lat+";"+mermaid_long+","+mermaid_lat).json()
print(res["routes"][0]["distance"])
```

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 at least one of x and 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:

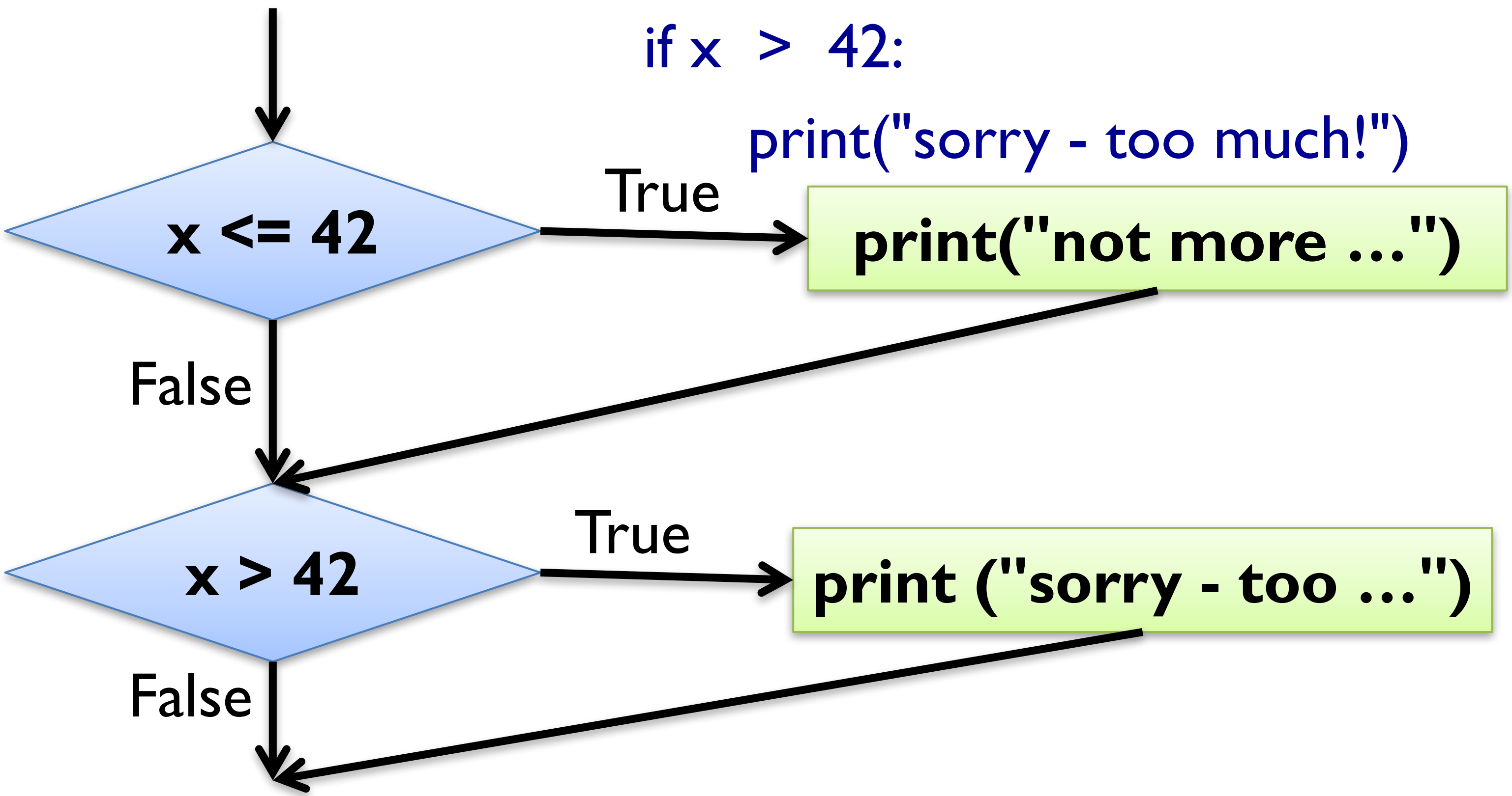
```
if x <= 42:
    print("not more than the answer")

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'$_{k'}$>

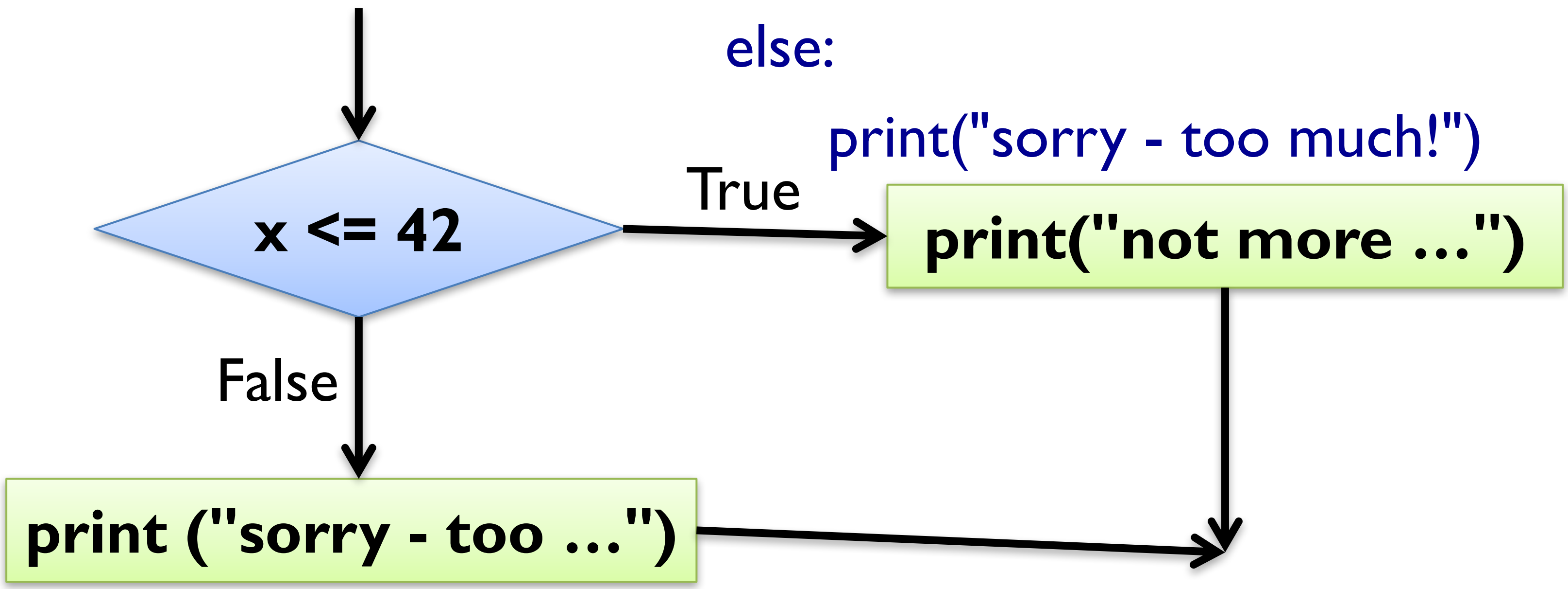
- Example:

```
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$_{km,m}$>

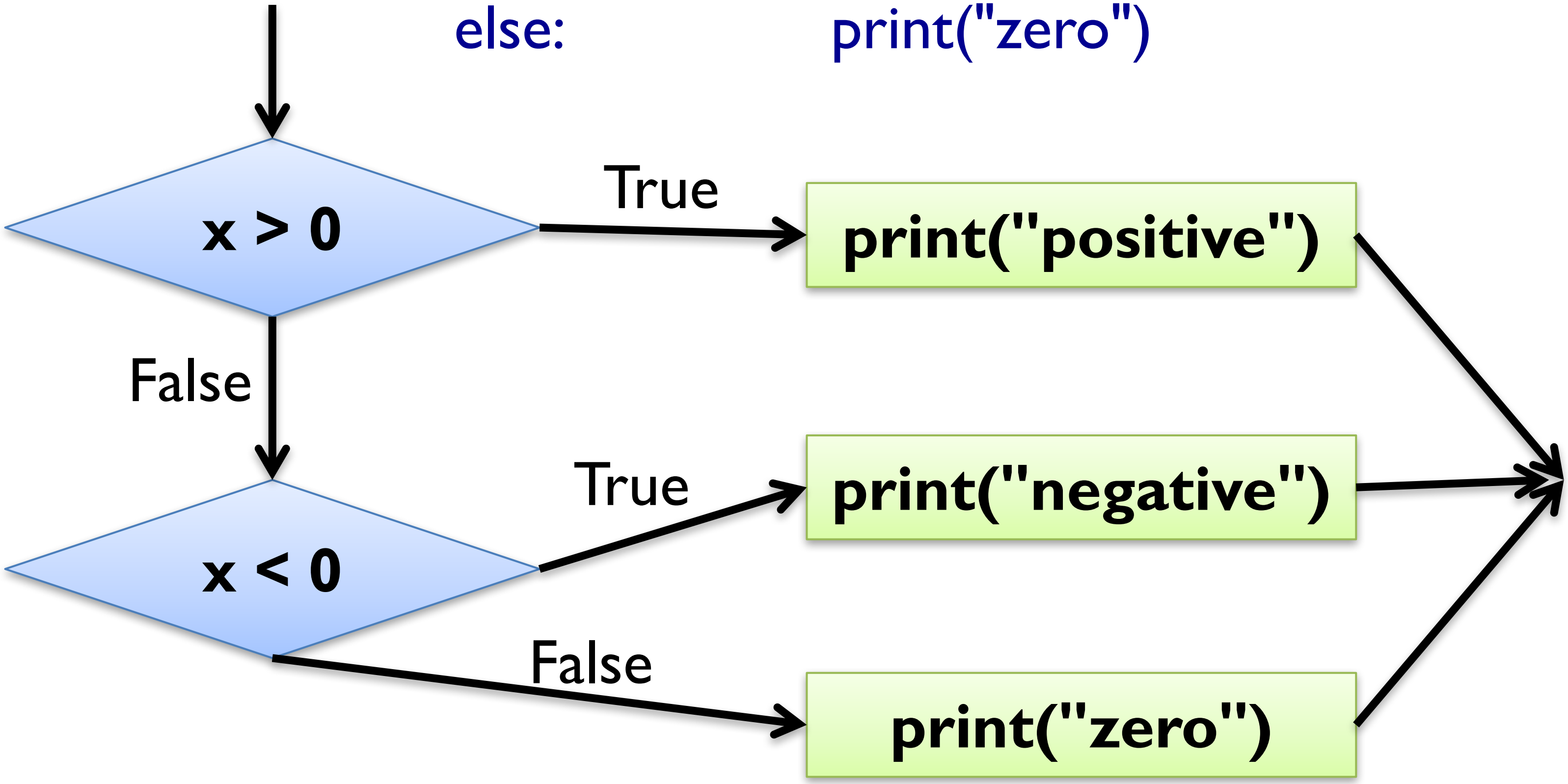
- Example:

```
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 = float(input())
y = float(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

__main__	
countdown	**n → 3**
countdown	**n → 2**
countdown	**n → 1**
countdown	**n → 0**

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):
        t.fd(length)
        t.lt(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:
        t.fd(length)
        t.lt(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?
  - what happens when we enter 42?
- the input function can take one optional argument prompt
  - Example 1: `a = float(input("first side: "))`
  - Example 2: `name = 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

error reported

```
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 = 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
```

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
    if x == 0:
        return 0
    return 1
```

- important that all paths reach one return

Incremental Development

- Idea: test code while writing it
- Example: computing the distance between (x_1,y_1) and (x_2,y_2)

```
def distance(x1, y1, x2, y2):
    print("x1 y1 x2 y2:", x1, y1, x2, y2)
```

Incremental Development

- Idea: test code while writing it
- Example: computing the distance between (x_1,y_1) and (x_2,y_2)

```
def distance(x1, y1, x2, y2):
    print("x1 y1 x2 y2:", x1, y1, x2, y2)
    dx = x2 - x1          # horizontal distance
    print("dx:", dx)
```

Incremental Development

- Idea: test code while writing it
- Example: computing the distance between (x_1,y_1) and (x_2,y_2)

```
def distance(x1, y1, x2, y2):
    print("x1 y1 x2 y2:", x1, y1, x2, y2)
    dx = x2 - x1          # horizontal distance
    print("dx:", dx)
    dy = y2 - y1          # vertical distance
    print("dy:", dy)
```

Incremental Development

- Idea: test code while writing it
- Example: computing the distance between (x_1,y_1) and (x_2,y_2)

```
def distance(x1, y1, x2, y2):
    print("x1 y1 x2 y2:", x1, y1, x2, y2)
    dx = x2 - x1          # horizontal distance
    print("dx:", dx)
    dy = y2 - y1          # vertical distance
    print("dy:", dy)
    dxs = dx**2;  dys = dy**2
    print("dxs dys:", dxs, dys)
```

Incremental Development

- Idea: test code while writing it
- Example: computing the distance between (x_1,y_1) and (x_2,y_2)

```
def distance(x1, y1, x2, y2):
    print("x1 y1 x2 y2:", x1, y1, x2, y2)
    dx = x2 - x1          # horizontal distance
    dy = y2 - y1          # vertical distance
    dxs = dx**2;  dys = dy**2
    print("dxs dys:", dxs, dys)
```

Incremental Development

- Idea: test code while writing it
- Example: computing the distance between (x_1,y_1) and (x_2,y_2)

```
def distance(x1, y1, x2, y2):
    print("x1 y1 x2 y2:", x1, y1, x2, y2)
    dx = x2 - x1          # horizontal distance
    dy = y2 - y1          # vertical distance
    dxs = dx**2;  dys = dy**2
    print("dxs dys:", dxs, dys)
    ds = dxs + dys        # square of distance
    print("ds:", ds)
```

Incremental Development

- Idea: test code while writing it
- Example: computing the distance between (x_1,y_1) and (x_2,y_2)

```
def distance(x1, y1, x2, y2):
    print("x1 y1 x2 y2:", x1, y1, x2, y2)
    dx = x2 - x1          # horizontal distance
    dy = y2 - y1          # vertical distance
    dxs = dx**2;  dys = dy**2
    ds = dxs + dys        # square of distance
    print("ds:", ds)
```

Incremental Development

- Idea: test code while writing it
- Example: computing the distance between (x_1,y_1) and (x_2,y_2)

```
def distance(x1, y1, x2, y2):
    print("x1 y1 x2 y2:", x1, y1, x2, y2)
    dx = x2 - x1          # horizontal distance
    dy = y2 - y1          # vertical distance
    dxs = dx**2;  dys = dy**2
    ds = dxs + dys        # square of distance
    print("ds:", ds)
    d = math.sqrt(ds)     # distance
    print(d)
```

Incremental Development

- Idea: test code while writing it
- Example: computing the distance between (x_1,y_1) and (x_2,y_2)

```
def distance(x1, y1, x2, y2):
    print("x1 y1 x2 y2:", x1, y1, x2, y2)
    dx = x2 - x1          # horizontal distance
    dy = y2 - y1          # vertical distance
    dxs = dx**2;  dys = dy**2
    ds = dxs + dys        # square of distance
    d = math.sqrt(ds)     # distance
    print(d)
```

Incremental Development

- Idea: test code while writing it
- Example: computing the distance between (x_1,y_1) and (x_2,y_2)

```
def distance(x1, y1, x2, y2):
    print("x1 y1 x2 y2:", x1, y1, x2, y2)
    dx = x2 - x1         # horizontal distance
    dy = y2 - y1         # vertical distance
    dxs = dx**2;  dys = dy**2
    ds = dxs + dys       # square of distance
    d = math.sqrt(ds)    # distance
    print(d)
    return d
```

Incremental Development

- Idea: test code while writing it
- Example: computing the distance between (x_1,y_1) and (x_2,y_2)

```
def distance(x1, y1, x2, y2):
    dx = x2 - x1          # horizontal distance
    dy = y2 - y1          # vertical distance
    dxs = dx**2;  dys = dy**2
    ds = dxs + dys        # square of distance
    d = math.sqrt(ds)     # distance
    return d
```

Incremental Development

- Idea: test code while writing it
- Example: computing the distance between (x_1,y_1) and (x_2,y_2)

```
def distance(x1, y1, x2, y2):
    dx = x2 - x1          # horizontal distance
    dy = y2 - y1          # vertical distance
    return math.sqrt(dx**2 + dy**2) # use Pythagoras
```

Incremental Development

- Idea: test code while writing it

1. start with minimal function
2. add functionality piece by piece
3. use variables for intermediate values
4. print those variables to follow your progress
5. remove unnecessary output when function is finished

Composition

- function calls can be arguments to functions
- direct consequence of arguments being expressions

- Example: area of a circle from center and peripheral point

```
def area(radius):
    return math.pi * radius**2

def area_from_points(xc, yc, xp, yp):
    return area(distance(xc, yc, xp, yp))
```

Boolean Functions

- boolean functions = functions that return True or False
- useful e.g. as <cond> in a conditional execution
- Example:

```
def divides(x, y):
    if y // x * x == y:    # remainder of integer division is 0
        return True
    return False
```

Boolean Functions

- boolean functions = functions that return True or False
- useful e.g. as <cond> in a conditional execution
- Example:

```
def divides(x, y):
    if y % x == 0:        # remainder of integer division is 0
        return True
    return False
```

Boolean Functions

- boolean functions = functions that return True or False
- useful e.g. as <cond> in a conditional execution
- Example:

```
def divides(x, y):
    return y % x == 0
```

Boolean Functions

- boolean functions = functions that return True or False
- useful e.g. as <cond> in a conditional execution
- Example:

```
def divides(x, y):
    return y % x == 0

def even(x):
    return divides(2, x)
```

Boolean Functions

- boolean functions = functions that return True or False
- useful e.g. as <cond> in a conditional execution
- Example:

```
def divides(x, y):
    return y % x == 0

def even(x):
    return divides(2, x)

def odd(x):
    return not divides(2, x)
```

Boolean Functions

- boolean functions = functions that return True or False
- useful e.g. as <cond> in a conditional execution
- Example:

```
def divides(x, y):
    return y % x == 0

def even(x):
    return divides(2, x)

def odd(x):
    return not even(x)
```