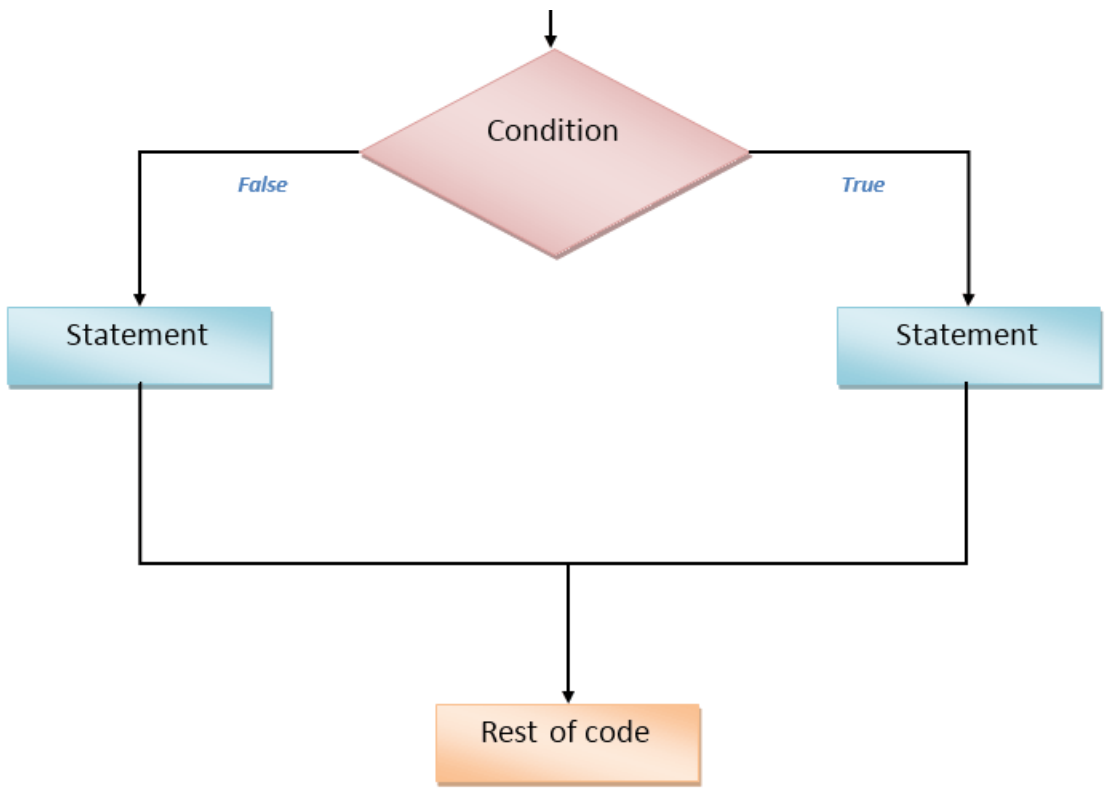
15-112
Fundamentals of Programming

Lecture 2:
Basic Building Blocks of Programming Continued



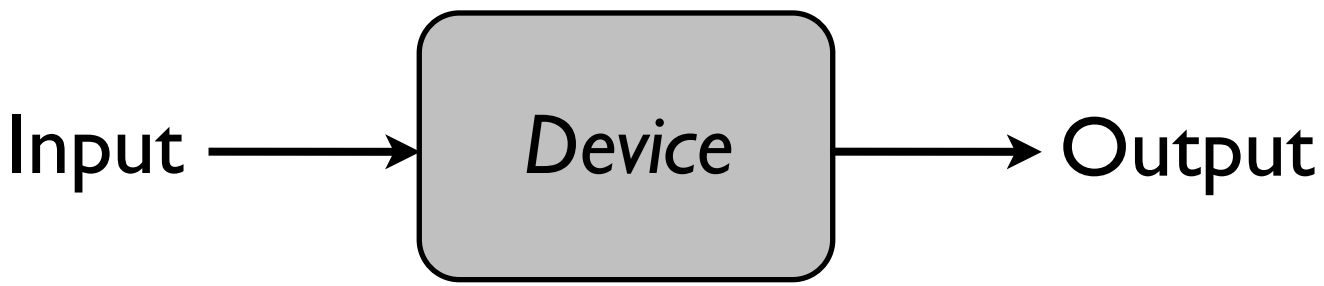
January 14, 2016

What we know so far:

What is a computer?

A programmable device that manipulates data/information

Usually



What is a computer program ?

A set of instructions that tells the computer how to manipulate data/information.

This Lecture (and Next (and Next...))

How do these instructions look like?
(What kind of instructions are allowed?)

How can I use these instructions to write programs?
(How do I approach programming, where do I start?)

Basic Building Blocks

BASICS OF HANDLING DATA

Statements
Tells the computer to do something.

Data Types
Data is divided into different types.

Variables
Allows you to store data and access stored data.

Operators
Allows you to manipulate data.

Conditional Statements
Executes statements if a condition is satisfied.

Functions
Mini self-contained programs.

One the menu today:

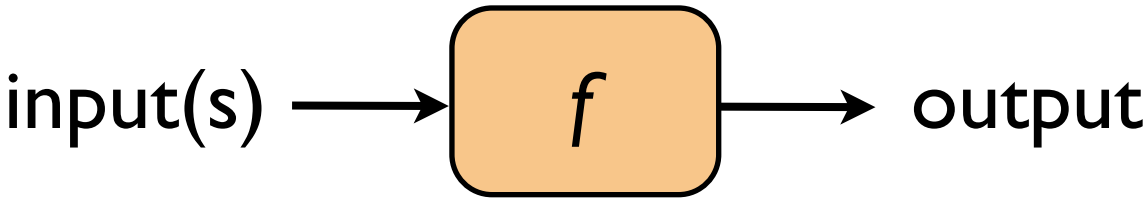
Wrapping up functions

Conditional statements

Practice problems

Functions in Python

A function in Python:



Python program =

a (main) function + other "helper" functions

Don't use print( ) or input( ):

We don't want to worry about how the input is acquired or how the output will be used.

We only use print( ) for debugging purposes.
We almost never use input( ).

Functions in Python

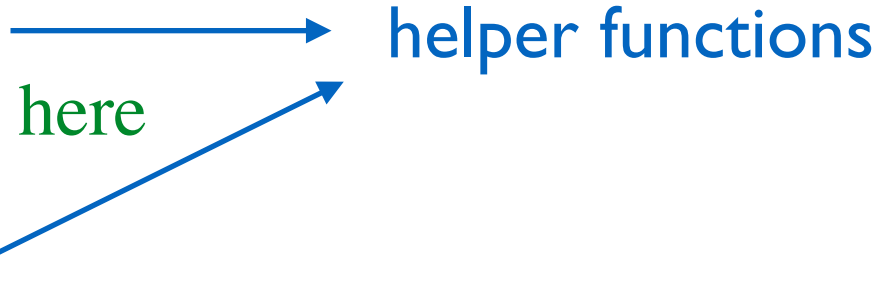
Example problem:

Write a function that takes 2 integers as input and returns the maximum of the ones digit of the numbers.

```
def max(x, y):
    # some code here

def onesDigit(x):
    # some code here

def largerOnesDigit(x, y):
    return max(onesDigit(x), onesDigit(y))
```



Testing Your Functions

Write a function that takes an integer and returns its tens digit.

tensDigit(5) should return 0
tensDigit(95) should return 9
tensDigit(4321) should return 2

Hint: If n is the input, think about the values
n // 10 and n % 10

```
def tensDigit(n):
    return (n // 10) % 10
```

Always test your function before you move on!

Testing Your Functions

Tester function

```
def testTensDigit():
    assert(tensDigit(5) == 0)
    assert(tensDigit(95) == 9)
    assert(tensDigit(4321) == 2)
    assert(tensDigit(-1234) == 3)
    print("Passed all tests!")
```

Fail

Make sure you select your test cases carefully!

Retry:

```
def tensDigit():
    return (abs(n) // 10) % 10
```

Built-in Functions

```
print(abs(-5))
print(max(2, 3))
print(min(2, 3))
print(pow(2, 3))            # raise to the given power (pow(x,y) == x**y)
print(round(2.354, 1))      # round with the given number of digits
```

```
print(type(5), end=" ")     <class 'int'>   <class 'str'>   <class 'bool'>
print(type("hello"), end=" ")
print(type(True))
```

```
import math
print(math.factorial(10))       3628800
print(math.pi)                  3.141592653589793
```

Conditional Statements

3 Types:

if statement

if-else statement

if-elif-...-elif-else statement

if Statement

```
instruction1
instruction2

if (expression):
    instruction3
    instruction4

instruction5
```

Ideally, should evaluate to **True** or **False**.

If the expression evaluates to **True**:

```
instruction1
instruction2
instruction3
instruction4
instruction5
```

if Statement

```
instruction1
instruction2

if (expression):
    instruction3
    instruction4

instruction5
```

Ideally, should evaluate to **True** or **False**.

If the expression evaluates to **False**:

```
instruction1
instruction2
instruction5
```

if Statement

```
1.  def abs(n):
2.      if(n < 0):
3.          n = -n
4.      return n
```

```
1.  def abs(n):
2.      if(n < 0): n = -n
3.      return n
```

```
1.  def abs(n):
2.      if(n < 0):
3.          return -n
4.      return n
```

if Statement

```
def message(age)
    if (age < 16):
        print("You can't drive.")
    if (age < 18):
        print("You can't vote.")
    if (age < 21):
        print("You can't drink alcohol.")
    if (age >= 21):
        print("You can do anything that's legal.")
    print("Bye!")
```

if - else

```
instruction1
instruction2

if(expression):
    instruction3
    instruction4
else:
    instruction5
    instruction6

instruction7
```

If the expression evaluates to **True**.

```
instruction1
instruction2
instruction3
instruction4
instruction7
```

Exactly one of the two blocks will get executed!

if - else

```
instruction1
instruction2

if(expression):
    instruction3
    instruction4
else:
    instruction5
    instruction6

instruction7
```

If the expression evaluates to **False**.

```
instruction1
instruction2
instruction5
instruction6
instruction7
```

Exactly one of the two blocks will get executed!

if - else

```
def f(x, y, z):
    if((x <= y and y <= z) or (x >= y and y >= z)):
        return True
    else:
        return False
```

if - else

```
def inOrder(x, y, z):
    if((x <= y and y <= z) or (x >= y and y >= z)):
        return True
    else:
        return False
```

if - else

```
def inOrder(x, y, z):
    if((x <= y and y <= z) or (x >= y and y >= z)):
        return True
    return False
```

if - else

What if you want to check 2 or more conditions ?

```
if(expression1):
    instruction1
else:
    if(expression2):
        instruction2
    else:
        instruction3
```

Only one of `instruction1`, `instruction2`, `instruction3` will be executed.

if - elif - else

```
if(expression1):
    instruction1
else:
    if(expression2):
        instruction2
    else:
        instruction3
```

```
if(expression1):
    instruction1
elif(expression2):
    instruction2
else:
    instruction3
```

if - elif - else

```
def numberOfQuadraticRoots(a, b, c):
    # Returns number of roots (zeros) of y = a*x**2 + b*x + c
    d = b**2 - 4*a*c
    if (d > 0):
        return 2
    elif (d == 0):
        return 1
    else:
        return 0
```

if - elif - ... - elif - else

```
def getGrade(score):
    if (score >= 90):
        grade = "A"
    elif (score >= 80):
        grade = "B"
    elif (score >= 70):
        grade = "C"
    elif (score >= 60):
        grade = "D"
    else:
        grade = "F"
    return grade
```

Conditional Expression

exp1 **if** (*exp2*) **else** *exp3*

The whole thing is an expression (i.e. evaluates to a value).

if *exp2* is **True**, evaluates to *exp1*

if *exp2* is **False**, evaluates to *exp3*

```
x = 5
y = 3
minimum = x if (x < y) else y
```

```
if (x < y):
    minimum = x
else:
    minimum = y
```

Conditional Expression

exp1 **if** (*exp2*) **else** *exp3*

The whole thing is an expression (i.e. evaluates to a value).

if *exp2* is **True**, evaluates to *exp1*

if *exp2* is **False**, evaluates to *exp3*

print("You have", x, "dollar." **if** (x == 1) **else** "dollars.")

Conditional Expression

exp1 **if** (*exp2*) **else** *exp3*

The whole thing is an expression (i.e. evaluates to a value).

if *exp2* is **True**, evaluates to *exp1*

if *exp2* is **False**, evaluates to *exp3*

doPenguinsSuck = True **if** (numWins < 50) **else** False

this is actually the same as:

doPenguinsSuck = (numWins < 50)

Some guidelines on correct usage of conditional statements

see course notes

Exercise: round(n)

Write a function that takes a float as input and returns the integer nearest to it.

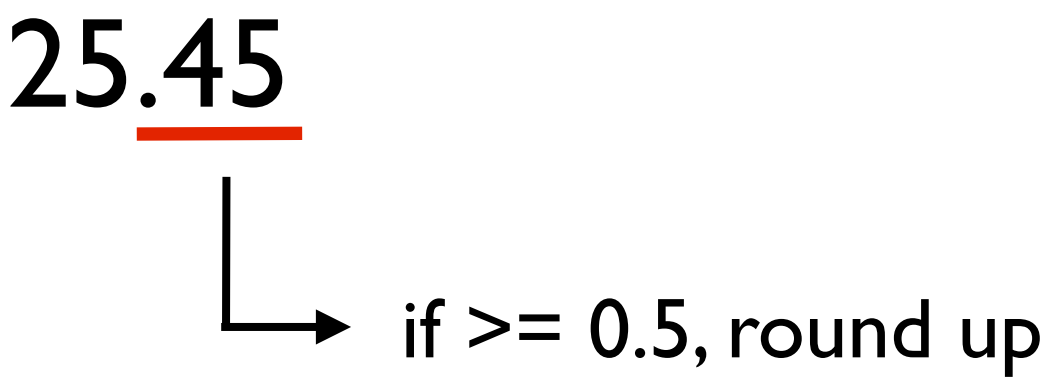
Exercise: round(n)

Steps to follow

- Find a mental picture of the solution
- Write an algorithm
- Write the code
- TEST!
- Fix the bugs (if any)

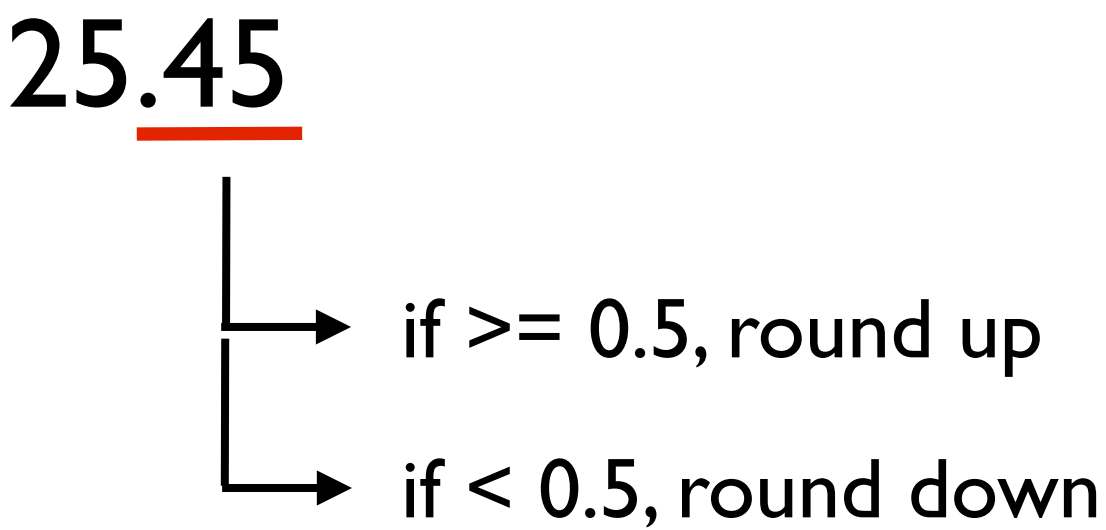
Exercise: round(n)

- Find a mental picture of the solution



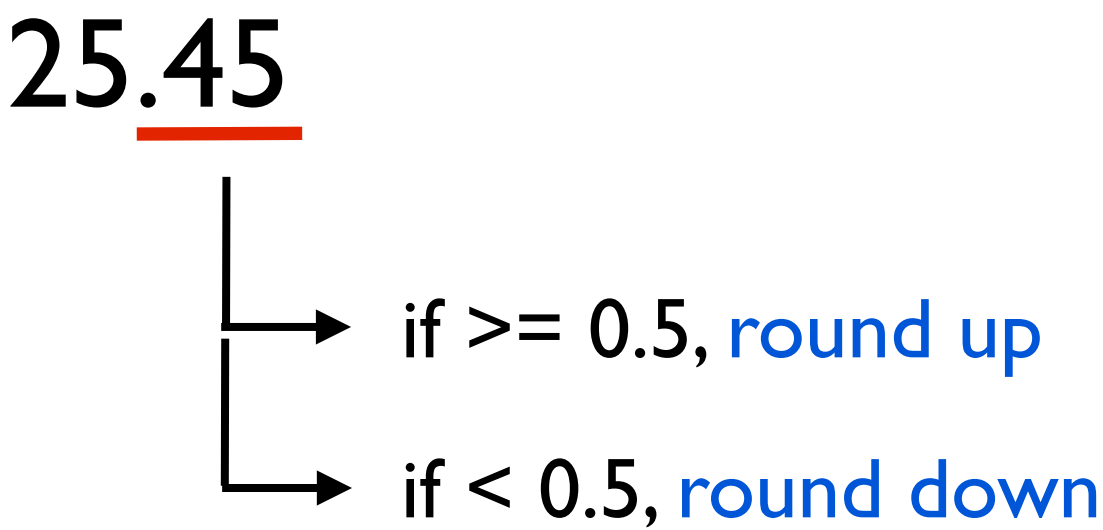
Exercise: round(n)

- Find a mental picture of the solution



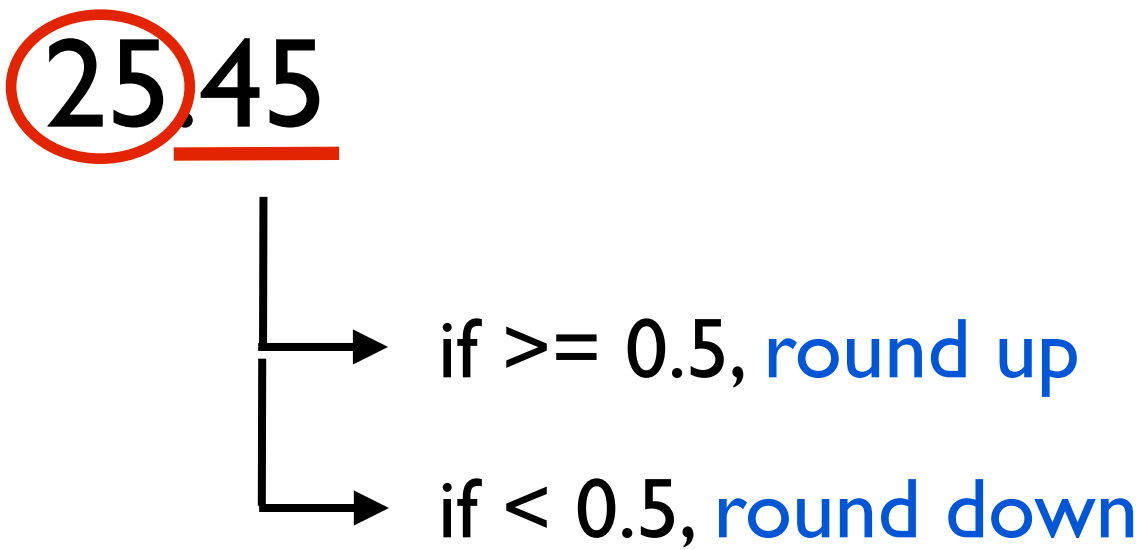
Exercise: round(n)

- Find a mental picture of the solution



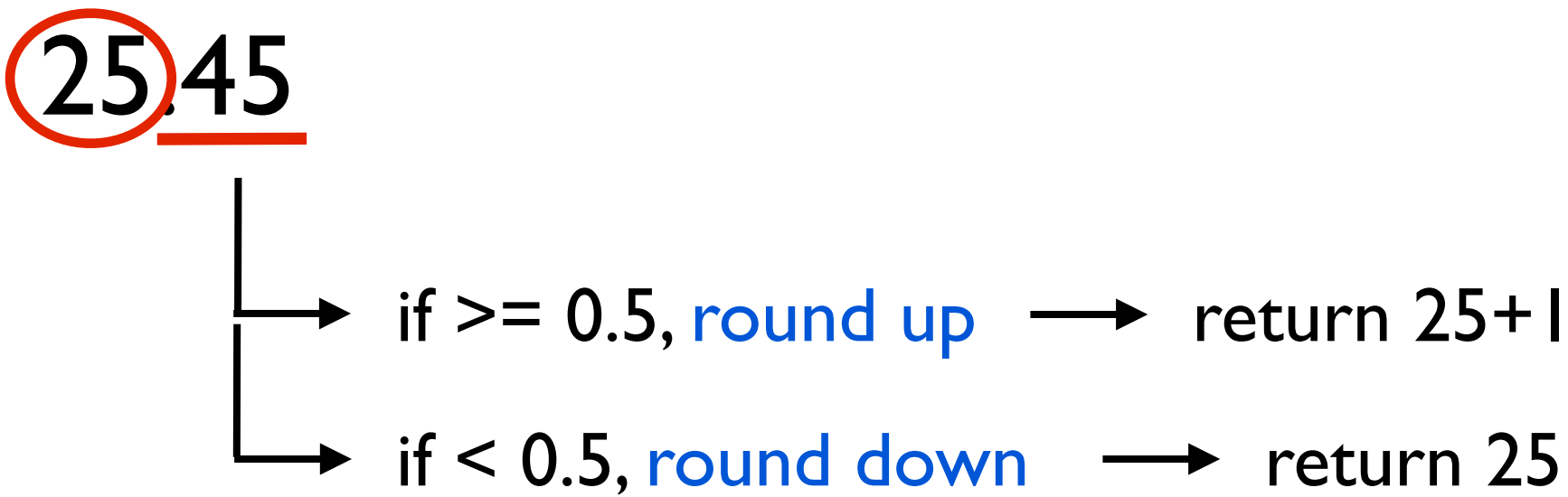
Exercise: round(n)

- Find a mental picture of the solution



Exercise: round(n)

- Find a mental picture of the solution



Exercise: round(n)

Steps to follow

- Find a mental picture of the solution

- Write an algorithm

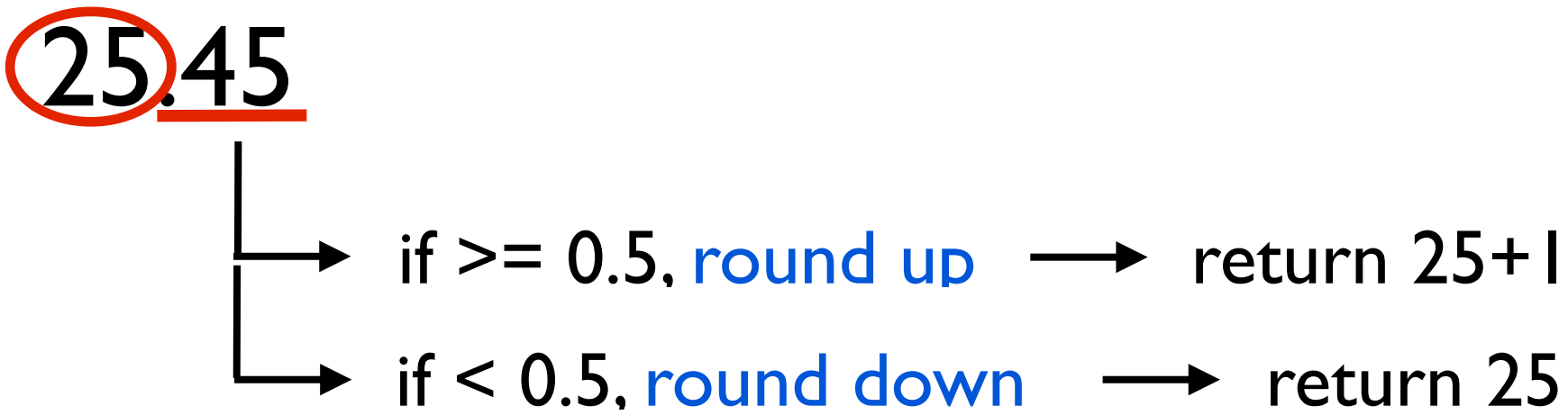
- Write the code

- TEST!

- Fix the bugs (if any)

Exercise: round(n)

- Write an algorithm



- Let n be the input number.
- Let intPart be the integer part of n.
  Let decPart be the decimal part of n.
- if decPart >= 0.5, return intPart + 1
- if decPart < 0.5, return intPart

Exercise: round(n)

Steps to follow

- Find a mental picture of the solution
- Write an algorithm
- Write the code
- TEST!
- Fix the bugs (if any)

Exercise: round(n)

- Write the code

algorithm:

- Let n be the input number.
- Let intPart be the integer part of n.
  Let decPart be the decimal part of n.
- if decPart >= 0.5, return intPart + 1
- if decPart < 0.5, return intPart

```
def round(n):
    intPart = int(n)
    decPart = n % 1
    if(decPart >= 0.5): return intPart + 1
    else: return intPart
```

Exercise: round(n)

- Find a mental picture of the solution

- Write an algorithm

- Write the code

- TEST!

- Fix the bugs (if any)

Exercise: round(n)

- TEST!

```
def testRound():
    assert(round(0) == 0)
    assert(round(0.5) == 1)
    assert(round(0.49999) == 0)
    assert(round(1238123.00001) == 1238123)
    assert(round(-0.5) == 0)    Error
    assert(round(-0.49999) == 0)
    assert(round(-0.51) == -1)
    assert(round(-1238123.00001) == -1238123)
    print("Passed all tests!")
```

Exercise: round(n)

Steps to follow

- Find a mental picture of the solution
- Write an algorithm
- Write the code
- TEST!
- Fix the bugs (if any)

Exercise: round(n)

- Fix the bugs (if any)

Exercise for you.