

CMPT 120

Functions and Decomposition

Summer 2012

Instructor: Hassan Khosravi

Defining Functions

- We have already seen how several functions work in Python
 - `raw_input`,
 - `range`,
 - `int`, and `str`
- A function must be given arguments.
 - These are the values in parentheses that come after the name of the function.
 - `int("321")`, the string "321" is the argument.
- Functions can have no arguments, or they can take several.
- Functions that return values can be used as part of an expression.
 - `x = 3*int("10") + 2`
 - ▶ , the variable `x` will contain the number 32.

Defining your own functions

- Functions are defined with a def block
 - `def linespace():`
 - `print`
 - `print "Hello"`
 - `linespace()`
 - `print "My name is Hassan"`

Example

- Read 10 numbers and return their squares using function
- `def square(num):`
- `num = num*num`
- `return num`
- `for i in range(10):`
- `input = int(raw_input("enter number: "))`
- `input_squared = square(input)`
- `print input_squared`

Perfect numbers

Find all perfect numbers between 1 to 100

- `def perfect(number):`
- `sum_divisor = 0`
- `for i in range(number-1):`
- `if number%(i+1) == 0:`
- `sum_divisor = sum_divisor + i+1`
- `if number == sum_divisor:`
- `return True`
- `else:`
- `return False`
- `for j in range (1,101):`
- `if perfect(j) == True:`
- `print j, "is perfect"`

Defining your own functions

Write a `read_integer` function

- `def read_integer(prompt):`
- `flag = True`
- `while flag == True:`
- `input = raw_input(prompt)`
- `if input.isdigit() == True:`
- `flag = False`
- `return int(input)`

- `num = read_integer("Type a number: ")`
- `print "One more is", num+1`
- `num = read_integer("Type another: ")`
- `print "One less is", num-1`

I-clicker question

- `def middle_value(a, b, c):`
- `if a <= b <= c or a >= b >= c:`
- `return b`
- `elif b <= a <= c or b >= a >= c:`
- `return a`
- `else:`
- `return c`

- `print middle_value(8,2,6) / 2`

- A:3
- B:2
- C:6
- D:5
- E:4

What happens when computer runs this code

- `half_mid = middle_value(8,2,6) / 2`
- The expression on the right of the variable assignment must be evaluated before the variable can be assigned
 - It evaluates the expression `middle_value(4,2,6) / 2`.
- The sub-expressions on either side of the division operator must be evaluated.
 - Evaluate `middle_value(4,2,6)`
 - Now, this statement is put on hold while the function does its thing
- The function `middle_value` is called.
- The arguments that are given in the calling code `(4,2,6)` are assigned to the local variables given in the argument list `(a,b,c)`.
 - `a=4` , `b=2`, `c=6`
- `c=6` is returned by the function
- The calling code gets the return value, 6. The expressions is now `6/2`.
- The integer 3 is assigned to the variable `half_mid`.

Why Use Functions?

- Functions can be used to break your program into logical sections.
 - Easier to build and debug
 - Makes the program easier to read
 - Functions are also quite useful to prevent duplication of similar code.
 - ▶ YOU SHOULD NEVER COPY PASTE CODE
 - ▶ What happens when you want to update code?
 - You need to hunt for that code everywhere to fix it
 - maintaining it is much easier.
 - Easier to distribute the work

- You are throwing a party
 - Among other things you need to
 - ▶ Greet friends coming in
 - ▶ Handle food
 - ▶ Handle Alcohol

- Instead of doing all that yourself you decide to get help from friends
 - Greeting friends → Jack
 - Food → James
 - Alcohol → Jim

Variable Scope

- Variables used inside a function are only available inside that function.
 - It is local inside that function

- `def square(num):`
- `num = num*num`
- `return num`

- `for i in range(10):`
- `input = int(raw_input("enter number: "))`
- `input_squared = square(input)`
- `print input_squared`

- This is actually a very good thing. It means that when you write a function, you can use a variable like `num` without worrying that some other part of the program is already using it.
 - Alcohol → Jim
 - ▶ Mike
 - Greeting friends → Jack
 - Food → James
 - ▶ Jim

Use of variable i

- `def perfect(number):`
- `sum_divisor = 0`
- `for i in range(number-1):`
- `if number%(i+1) == 0:`
- `sum_divisor = sum_divisor + i+1`
- `if number == sum_divisor:`
- `return True`
- `else:`
- `return False`

- `for i in range (1,101):`
- `if perfect(i) == True:`
- `print i, "is perfect"`

If that was not the case

- it becomes very hard to write large programs.
- Imagine trying to write some code and having to check 20 different functions every time you introduce a new variable to make sure you're not using the same name over again.
- The code has very limited interaction with the rest of the program. This makes it much easier to debug programs that are separated with functions.
 - Greeting friends → Jack
 - Food → James
 - Alcohol → Jim
 - Each use ten of their friends to help them.

Assignment 1

What sort of functions may be helpful for the assignment?

Python Modules

- In most programming languages, you aren't expected to do everything from scratch.
- Some prepackaged functions come with the language
- These are usually called libraries
 - In python they are called modules
- There are many available modules in Python.
 - Module time (you should check the documentation for a module to see how to work with it)
 - ▶ <http://docs.python.org/library/time.html>
 - ▶ The time module has a function strftime that can be used to output the current date and time in a particular format.
- Modules need to be imported before being used
 - they can be used. There are so many modules that if they were all imported automatically, programs would take a long time to startup

- ▶ `import time`
- ▶ `print "Today is " + time.strftime("%B %d, %Y") + "."`

- If you import a function like `import time`
 - then you can use methods like `time.strftime("%B %d, %Y")`
- If you import a function like `From time import *`
 - `strftime("%B %d, %Y")`

Objects

- Objects are collections of properties and methods.
 - Objects are only touched on in this course and are usually covered in details in higher level courses.

- Real life objects:
 - A DVD player is an example of an object
 - ▶ Buttons correspond to various actions the player can do

- Objects in programming language
 - Are very similar to real objects

Properties and methods

- Properties works like variables. It holds some information about the object.
 - The current position in the movie might be a property. (you can change the value)
 - ▶ In python you can set properties like variables

- A method works like a function. It performs some operation on the object.
 - For the DVD player, a method might be something like “play this DVD”.
 - ▶ A method might change some of the method’s properties
 - like set the counter to 0:00:00

Class and instances

- A particular kind of object is called a class
 - there is a class called “DVD Player”.
- When you create (buy) an object in the class it's called an instance.
- An instance behaves a lot like any other variable, except
 - it contains methods and properties.
 - So, objects are really variables that contain variables and functions of their own.

Objects in Python

- Classes in Python can be created by the programmer or can come from modules.
 - We won't be creating our own classes in this course, just using classes provided by modules.
- To instantiate an object, its constructor is used. This is a function that builds the object and returns it.
 - Buying your DVD player for you and setting it up
 - `import datetime`
 - `newyr = datetime.date(2005, 01, 01) # constructor`
 - `print newyr.year # the year property`
 - `print newyr.strftime("%B %d, %Y") # the strftime method`
 - `print newyr`

- The ways you can use an object depend on how the class has been defined.
 - The things you can do with you DVD player depends on the DVD player.

- For example date class does not know how to add in the date object
 - `import datetime`
 - `first = datetime.date(1989, 12, 17)`
 - `print first`
 - `print first+7`
 - ▶ `TypeError: unsupported operand type(s) for +: 'datetime.date' and 'int'`

- So, Python doesn't know how to add the integer 7

- But, it does know how to subtract dates:
 - `import datetime`
 - `first = datetime.date(1989, 12, 17)`
 - `second = datetime.date(1990, 1, 14)`
 - `print second - first`
 - `print type(second-first)`
 - ▶ Stores the time between two events
 - `print second + first`
 - ▶ still doesn't work

Handling Errors

- `m_str = raw_input("Enter your height (in metres): ")`
 - `metres = float(m_str)`
 - `feet = 39.37 * metres / 12`
 - `print "You are " + str(feet) + " feet tall."`
- Traceback (most recent call last): File "C:/Documents and Settings/abozorgk/Desktop/sum.py", line 2, in <module> metres = float(m_str) ValueError: could not convert string to float:
- This isn't very helpful for the user as it terminates the whole program
 - Errors that happen while the program is running are called exceptions

- Python lets you catch any kind of error,
 - `m_str = raw_input("Enter your height (in metres): ")`
 - `try:`
 - `metres = float(m_str)`
 - `feet = 39.37 * metres / 12`
 - `print "You are " + str(feet) + " feet tall."`
 - `except:`
 - `print "That wasn't a number."`
- The try/except block lets the program handle exceptions when they happen.
- If any exceptions happen while the try part is running, the except code is executed. It is ignored otherwise.

- `got_height = False`
- `while not got_height:`
- `m_str = raw_input("Enter your height (in metres): ")`
- `try:`
- `metres = float(m_str)`
- `got_height = True # if we're here, it was converted.`
- `except:`
- `print "Please enter a number."`
- `feet = 39.37 * metres / 12`
- `print "You are " + str(feet) + " feet tall."`

Catching Different Types of Errors

- `got_height = False`
- `while not got_height:`
- `m_str = raw_input("Enter your height (in metres): ")`
- `try:`
- `b= 10/0`
- `metres = float(m_str)`
- `got_height = True # if we're here, it was converted.`
- `except:`
- `print "Please enter a number."`
- `feet = 39.37 * metres / 12`
- `print "You are " + str(feet) + " feet tall."`

- Type of errors

- 10/0

- ▶ ZeroDivisionError

- Float("asd")

- ▶ ValueError

- got_height = False

- while not got_height:

- m_str = raw_input("Enter your height (in metres): ")

- try:

- metres = float(m_str)

- got_height = True # if we're here, it was converted.

- except ValueError:

- print "Please enter a number."

- feet = 39.37 * metres / 12

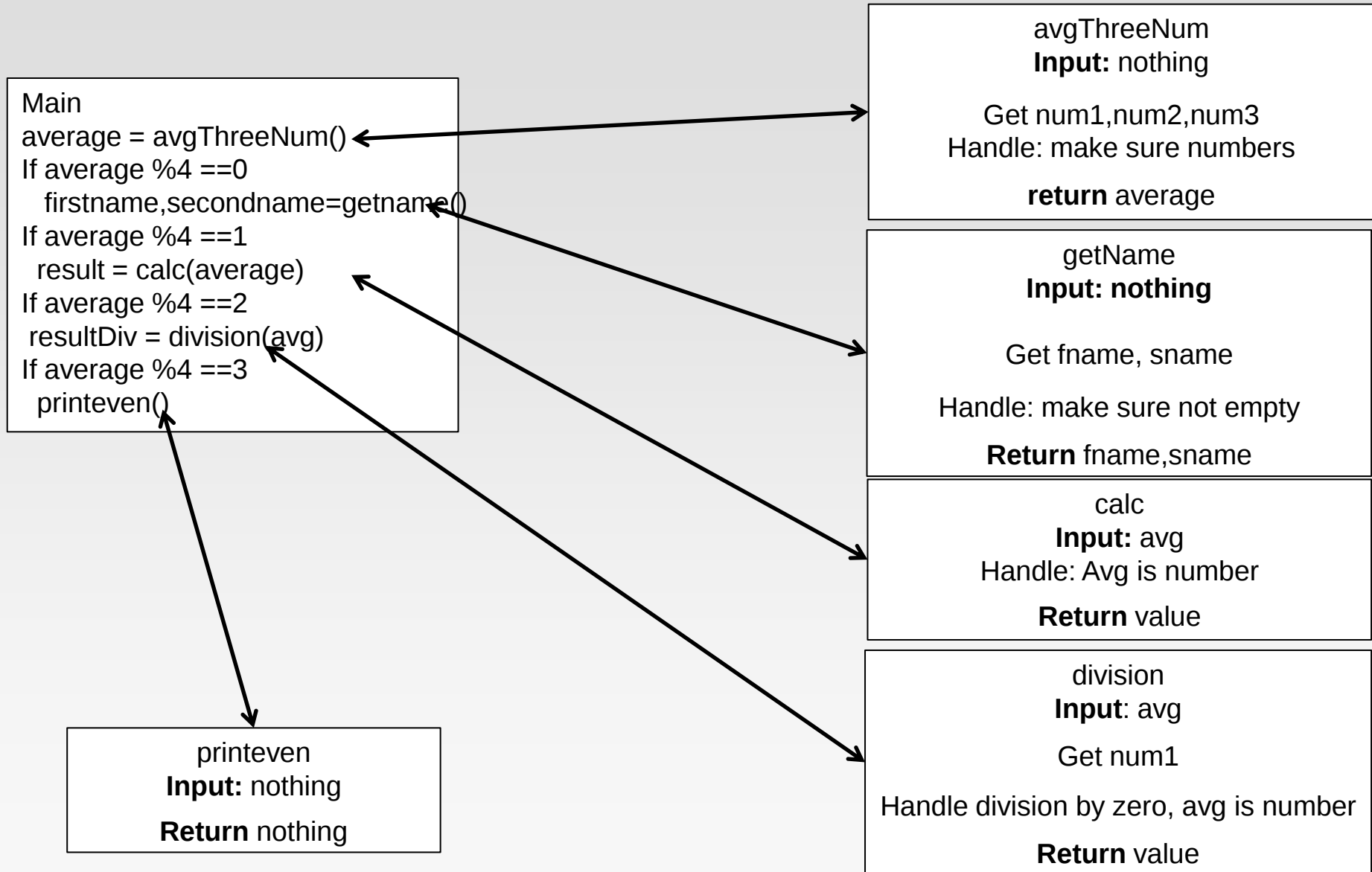
- print "You are " + str(feet) + " feet tall."

- `got_height = False`
- `while not got_height:`
- `m_str = raw_input("Enter your height (in metres): ")`
- `try:`
- `metres = float(m_str)`
- `10/ metres`
- `got_height = True # if we're here, it was converted.`
- `except ZeroDivisionError:`
- `print "division by zero"`
- `except ValueError:`
- `print "please enter integer"`
- `feet = 39.37 * metres / 12`
- `print "You are " + str(feet) + " feet tall."`

Example

- Write a program that finds the average of three numbers.
 - If the remainder of average divided by four is 0 then ask for the first name name and surname of
 - If the remainder of average divided by four is 1 then calculate and print $(\text{average})^3 - (\text{average})^2$
 - If the remainder of average divided by four is 2 then ask for a new number n and calculate and print $\text{average}/n$
 - If the remainder of average divided by four is 3 then print all positive even numbers smaller than 15

Example



printeven
Input: nothing
Return nothing

- def printeven():
- for i in range(0,15,2):
- print i

division
Input: avg
Get num1
Handle division by zero, avg is number
Return value

- def division(avg):
- num1 = read_integer("please enter a number")
- try:
- value = avg/num1
- except ZeroDivisionError:
- print "you had division by zero"
- return 0
- except TypeError:
- print "avg is not a number"
- return 0
- return value

- def read_integer(prompt):
- flag = True
- while flag == True:
- input = raw_input(prompt)
- if input.isdigit() == True:
- flag = False
- return int(input)

calc
Input: avg
Handle: Avg is number
Return value

- def calc(avg):
- try:
- avg = float(avg)
- except ValueError:
- print "avg in calc is not a number"
- return 0
- value = avg*avg*avg - avg**2
- return value

getName
Input: nothing

Get fname, sname

Handle: make sure not empty

Return fname,sname

- def getName():
- fname = raw_input("What is your first name? ")
- while fname=="":
- fname = raw_input("Please enter your name: ")
- sname = raw_input("What is your surname name? ")
- while sname=="":
- sname = raw_input("Please enter your surname: ")
- return fname, sname

avgThreeNum

Input: nothing

Get num1,num2,num3

Handle: make sure numbers

return average

- def avgThreeNum():
- num1 = read_integer("please enter first number")
- num2 = read_integer("please enter second number")
- num3 = read_integer("please enter third number")
- avg = (num1 + num2 + num3)/3
- return avg

```
Main
average = avgThreeNum()
If average %4 ==0
    firstname,secondname=getname()
If average %4 ==1
    result = calc(average)
If average %4 ==2
    resultDiv = division(avg)
If average %4 ==3
    printeven()
```

- avg = avgThreeNum()
- print avg
- if avg % 4 == 0:
 - firstname, secondname = getName()
 - print firstname, secondname
- elif avg % 4 == 1:
 - results = calc(avg)
 - print results
- elif avg % 4 == 2:
 - results = division(avg)
 - print results
- else:
 - printeven()