Problem Decomposition

This section of notes shows you how to break down a large problem into smaller parts that are easier to implement and manage.

Problem Solving Approaches

- Bottom up
- Top down

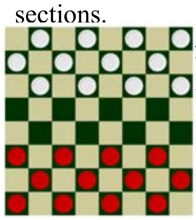
Bottom Up Approach To Design

•Start implementing all details of a solution without first developing a structure or a plan.

Here is the first of my many witty anecdotes, it took place in a "Tim Horton's" in Balzac..

•Potential problems:

–Generic problems): Redundancies and lack of coherence between sections.



ecific problem): Trying to implement all the details of at once may prove to be overwhelming.

Top Down Design

1. Start by outlining the major parts (structure)

My autobiography

Chapter 1: The humble beginnings

Chapter 2: My rise to greatness

…

Chapter 7: The end of an era

2. Then implement the solution for each part

Chapter 1: The humble beginnings

It all started ten and one score years ago with a log-shaped work station…

Procedural Programming: Breaking A Large Problem Down

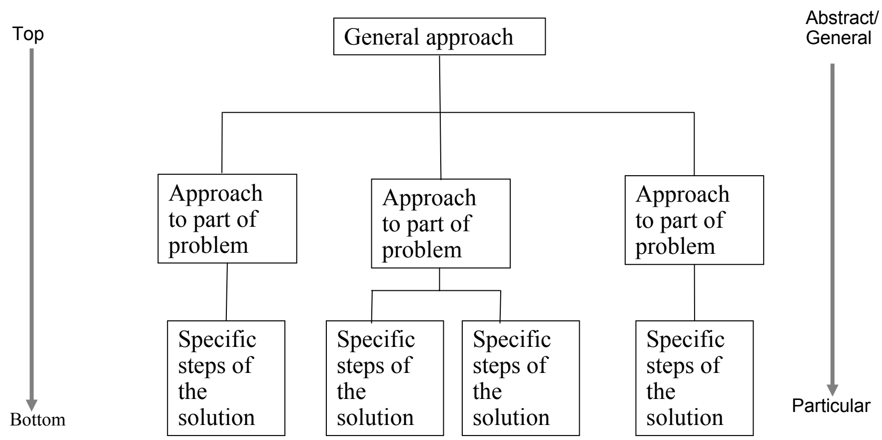
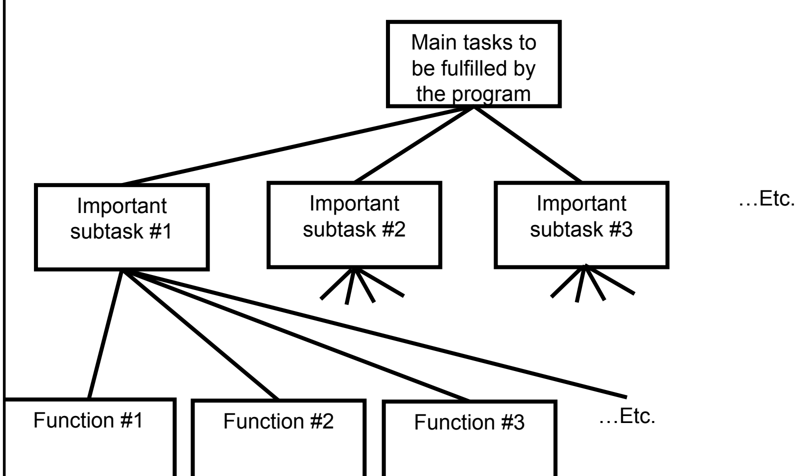


Figure extracted from Computer Science Illuminated by Dale N. and Lewis J.

Procedural Programming



Decomposing A Problem Into Procedures

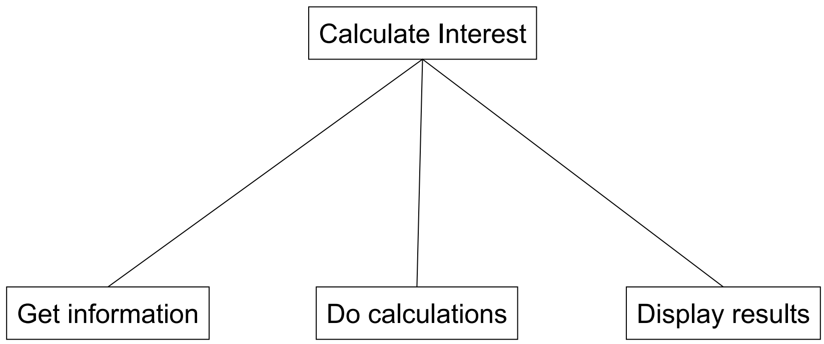
•Break down the program by what it does (described with *actions/verbs*).

•Eventually the different parts of the program will be implemented as functions.

Example Problem

- Design a program that will perform a simple interest calculation.
- The program should prompt the user for the appropriate values, perform the calculation and display the values onscreen.
- Action/verb list:
  - Prompt
  - Calculate
  - Display

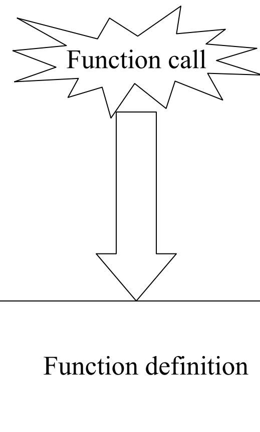
Top Down Approach: Breaking A Programming Problem Down Into Parts (Functions)



Things Needed In Order To Use Functions

- Definition
  - Indicating what the function will do when it runs
- Call
  - Getting the function to run (executing the function)

Functions (Basic Case)



Defining A Function

•Format:

def <*function name*> ():
    body

•Example:

```
def displayInstructions ():
    print "Displaying instructions"
```

Calling A Function

- **Format:**

  function name ()

- **Example**:

  displayInstructions ()

Functions: An Example That Puts Together All The Parts Of The Easiest Case

- The full version of this program can be found in UNIX under /home/231/examples/functions/firstExampleFunction.py

```
def displayInstructions ():
    print "Displaying instructions"


# main function
displayInstructions()
print "End of program"
```

Functions: An Example That Puts Together All The Parts Of The Easiest Case

•The full version of this program can be found in UNIX under /home/231/examples/functions/firstExampleFunction.py

```
def displayInstructions ():
    print "Displaying instructions"
```

Function definition

```
# main function
displayInstructions()
print "End of program"
```

Function call

Functions Should Be Defined Before They Can Be Called!

•**Correct** ☺

```
def fun ():
    print "Works"
```

Function definition

```
# main
fun ()
```

Function call

•**Incorrect** ☹

```
fun ()
```

Function call

```
def fun ():
    print "Doesn't work"
```

Function definition

Another Common Mistake

•Forgetting the brackets during the function call:

```
def fun ():
    print "In fun"

# Main function
print "In main"
fun
```


Another Common Mistake

•Forgetting the brackets during the function call:

```
def fun ():
    print "In fun"

# Main function
print "In main"
fun ()
```

The missing set of brackets does not produce a translation error

Another Problem: Creating ‘Empty’ Functions

```
def fun ():
```

 # Main

```
fun()
```

Problem: This statement appears to be a part of the body of the function but it is not indented???!!!

Another Problem: Creating ‘Empty’ Functions (2)

```
def fun ():
    print
```

A function must have at least one statement

 # Main

```
fun()
```

What You Know: Declaring Variables

- Variables are memory locations that are used for the temporary storage of information.

RAM

num = 0   num | 0 |

- Each variable uses up a portion of memory, if the program is large then many variables may have to be declared (a lot of memory may have to be allocated to store the contents of variables).

James Tam

What You Will Learn: Using Variables That Are Local To A Function

- To minimize the amount of memory that is used to store the contents of variables only declare variables when they are needed.
- When the memory for a variable is no longer needed it can be ‘freed up’ and reused.
- To set up your program so that memory for variables is only allocated (reserved in memory) as needed and de-allocated when they are not (the memory is free up) variables should be declared locally to a function.

Function call (*local variables get allocated in memory*) → The program code in the function executes (the variables are used to store information for the function) → Function ends (*local variables get de-allocated in memory*)

James Tam

Where To Create Local Variables

def <*function name*> ():

Somewhere within the body of the function (indented part)

Example:

```
def fun ():
    num1 = 1
    num2 = 2
```


Working With Local Variables: Putting It All Together

- The full version of this example can be found in UNIX under /home/231/examples/functions/secondExampleFunction.py

```
def fun ():
   num1 = 1
   num2 = 2
   print num1, " ", num2

# Main function
fun()
```

Working With Local Variables: Putting It All Together

•The full version of this example can be found in UNIX under /home/231/examples/functions/secondExampleFunction.py

```
def fun ():
    num1 = 1
    num2 = 2
    print num1, " ", num2

# Main function
fun()
```

Variables that are local to function fun

Problem: Local Variables Only Exist Inside A Function

```
def display ():
    print ""
    print "Celsius value: ", celsius
    print "Fahrenheit value :", fahrenheit
```

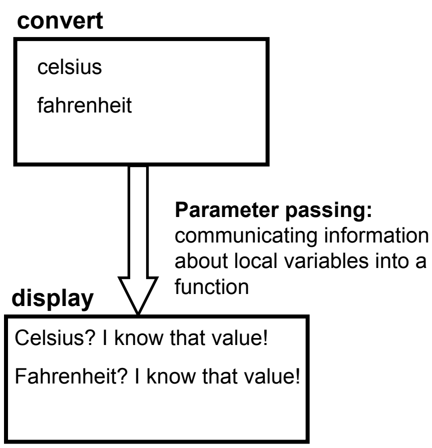
What is 'celsius'???
What is 'fahrenheit'???

```
def convert ():
    celsius = input ("Type in the celsius temperature: ")
    fahrenheit = celsius * (9 / 5) + 32
    display ()
```

Variables celsius and fahrenheit are local to function 'convert'

Solution: Parameter Passing

•Variables exist only inside the memory of a function:



Parameter Passing (Function Definition)

•**Format:**

def <*function name*> (<*parameter 1*>, <*parameter 2*>...):

•**Example:**

def display (celsius, fahrenheit):

Parameter Passing (Function Call)

- **Format:**

  <function name> (*<parameter 1>*, *<parameter 2>*...)

- **Example:**

  display (celsius, fahrenheit):

Parameter Passing: Putting It All Together

- The full version of this program can be found in UNIX under /home/231/examples/functions/temperature.py

```
def introduction ():
    print """
Celsius to Fahrenheit converter
-------------------------------
This program will convert a given Celsius temperature to an equivalent
Fahrenheit value.

    """
```

Parameter Passing: Putting It All Together (2)

```
def display (celsius, fahrenheit):
    print ""
    print "Celsius value: ", celsius
    print "Fahrenheit value:", fahrenheit


def convert ():
    celsius = input ("Type in the celsius temperature: ")
    fahrenheit = celsius * (9 / 5) + 32
    display (celsius, fahrenheit)


# Main function
introduction ()
convert ()
```


New Problem: Results That Are Derived In One Function Only Exist In That Function

Stored locally
interest = 50

```
def calculateInterest (principle, rate, time):
    interest = principle * rate * time
```

```
# main
principle = 100
rate = 0.1
time = 5
calculateInterest (principle, rate, time)
print "Interest earned $", interest
```

Problem:

Value stored in interest cannot be accessed here

Solution: Have Function Return Values Back To The Caller

```
def calculateInterest (principle, rate, time):
    interest = principle * rate * time
    return interest
```

Variable 'interest' is local to the function.

```
# main
    principle = 100
    rate = 0.1
    time = 5
    interest = calculateInterest (principle, rate, time)
    print "Interest earned $", interest
```

The value stored in the variable 'interest' local to 'calculateInterest' is passed back and stored in a variable that is local to the main function.

Using Return Values

- **Format (Single value returned):**

  return <*value returned*> # Function definition

  <*variable name*> = <*function name*> () # Function call

- **Example (Single value returned):**

  return interest # Function definition

  interest = calculateInterest (principle, rate, time) # Function call

Using Return Values

- **Format (Multiple values returned):**

  return <*value1*>, <*value 2*>... # Function definition
  <*variable 1*>, <variable 2>... = <*function name*> () # Function call

- **Example (Multiple values returned):**

```
return principle, rate, time                                  # Function definition
principle, rate, time = getInputs (principle, rate, time)  # Function call
```


Using Return Values: Putting It All Together

- The full version of this program can be found in UNIX under /home/231/examples/functions/interest.py

```
def introduction ():
    print """
Simple interest calculator
-------------------------------
With given values for the principle, rate and time period this program
will calculate the interest accrued as well as the new amount (principle
plus interest).
    """
```

Using Return Values: Putting It All Together (2)

```
def getInputs (principle, rate, time):
    principle = input("Enter the original principle: ")
    rate = input("Enter the yearly interest rate %")
    rate = rate / 100.0
    time = input("Enter the number of years that money will be invested: ")
    return principle, rate, time

def calculate (principle, rate, time, interest, amount):
    interest = principle * rate * time
    amount = principle + interest
    return interest, amount
```


Using Return Values: Putting It All Together (3)

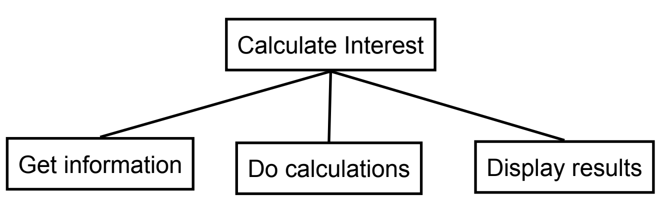
```
def display (principle, rate, time, interest, amount):
    temp = rate * 100
    print ""
    print "With an investment of $", principle, " at a rate of", temp, "%",
    print " over", time, " years..."
    print "Interest accrued $", interest
    print "Amount in your account $", amount
```

Using Return Values: Putting It All Together (4)

```
# Main function
principle = 0
rate = 0
time = 0
interest = 0
amount = 0

introduction ()
principle, rate, time = getInputs (principle, rate, time)
interest, amount = calculate (principle, rate, time, interest, amount)
display (principle, rate, time, interest, amount)
```

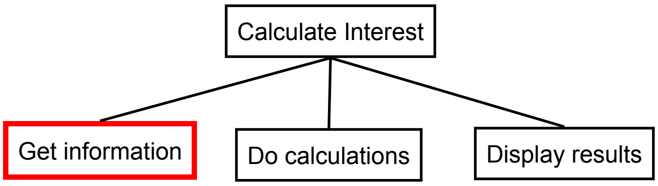

Testing Functions

- This is an integral part of the top down approach to designing programs.
- Recall with the top down approach:
  1. Outline the structure of different parts of the program without implementing the details of each part (i.e., specify what functions that the program must consist of but don't write the code for the functions yet).



Testing Functions

2. Implement the body of each function, one-at-a-time.




```
# Get information
def getInput (principle, rate, time):
   principle = input (“Enter the principle: “)
   rate = input("Enter the yearly interest rate %")
   rate = rate / 100.0
   time = input("Enter the number of years the
                    money will be invested: ")
   return principle, rate, time
```


Testing Functions

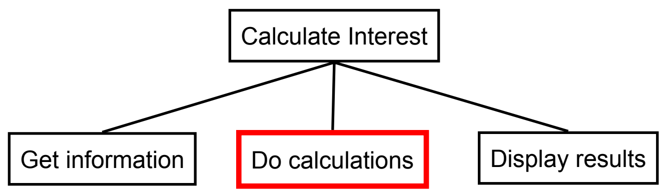
2. As each function has been written test each one to check for errors.

```
# main
   principle, rate, time = getInput (principle, rate, time)
   print “principle”, principle
   print “rate”, rate
   print “time”, time
```

```
# Get information
def getInput (principle, rate, time):
              :              :
    return principle, rate, time
```

Testing Functions

2. As each function has been written test each one to check for errors.




```
# Do calculations
def calculate (principle, rate, time, interest,
                    amount):
    interest = principle * rate * time
    amount = principle + interest
    return interest, amount
```


Testing Functions

2. As each function has been written test each one to check for errors.

```
# main
# Test case 1: Interest = 0, Amount = 0
interest, amount = calculate (0, 0, 0, interest, amount)
print "interest", interest, ' ', "amount", amount

# Test case 2: Interest = 50, Amount = 150
interest, amount = calculate (100, 0.1, 5, interest, amount)
print "interest", interest, ' ', "amount", amount
```

```
# Do calculations
def calculate (principle, rate, time, interest,
                    amount):
    interest = principle * rate * time
    amount = principle + interest
    return interest, amount # 0, 0
```

Testing Functions

2. As each function has been written test each one to check for errors.

```
# main
# Test case 1: Interest = 0, Amount = 0
interest, amount = calculate (0, 0, 0, interest, amount)
print "interest", interest, ' ', "amount", amount

# Test case 2: Interest = 50, Amount = 150
interest, amount = calculate (100, 0.1, 5, interest, amount)
print "interest", interest, ' ', "amount", amount
```

```
# Do calculations
def calculate (principle, rate, time, interest,
               amount):
    interest = principle * rate * time
    amount = principle + interest
    return interest, amount # 50, 150
```


The Type And Number Of Parameters Must Match!

•Correct ☺:

```
def fun1 (num1, num2):
    print num1, num2

def fun2 (num1, str1):
    print num1, str1

# main
num1 = 1
num2 = 2
str1 = "hello"

fun1 (num1, num2)
fun2 (num1, str1)
```

Two parameters (a number and a string) are passed into the call for 'fun2' which matches the type for the two parameters listed in the definition for function 'fun2'

Two numeric parameters are passed into the call for 'fun1' which matches the two parameters listed in the definition for function 'fun1'

Another Common Mistake: The Parameters Don't Match

•Incorrect ☹:

```
def fun1 (num1):
    print num1, num2

def fun2 (num1, num2):
    num1 = num2 + 1
    print num1, num2


# main
num1 = 1
num2 = 2
str1 = "hello"

fun1 (num1, num2)
fun2 (num1, str1)
```

Two parameters (a number and a string) are passed into the call for 'fun2' but in the definition of the function it's expected that both parameters are numeric.

Two numeric parameters are passed into the call for 'fun1' but only one parameter is listed in the definition for function 'fun1'

Program Design: Finding The Candidate Functions

- The process of going from a problem description (words that describe what a program is supposed to do) to writing a program.
- The first step is to look at verbs either directly in the problem description (indicates what actions should the program be capable of) or those which can be inferred from the problem description.
- Each action may be implemented as a function but complex actions may have to be decomposed further into several functions.

Rules Of Thumb For Defining Functions

1. Each function should have one well defined task. If it doesn't then it may have to be decomposed into multiple sub-functions.
   a) Clear function: A function that converts lower case input to capitals.
   b) Ambiguous function: A function that prompts for a string and then converts that string to upper case.
2. Try to avoid writing functions that are longer than one screen in size (again this is just a rule of thumb or guideline!)

Program Design: An Example Problem

•(Paraphrased from the book "Pascal: An introduction to the Art and Science of Programming" by Walter J. Savitch.

Problem statement:

Design a program to make change. Given an amount of money, the program will indicate how many quarters, dimes and pennies are needed. The cashier is able to determine the change needed for values of a dollar or less.

Actions that may be needed:

•Action 1: Prompting for the amount of money

•Action 2: Computing the combination of coins needed to equal this amount

•Action 3: Output: Display the number of coins needed

Program Design: An Example Problem

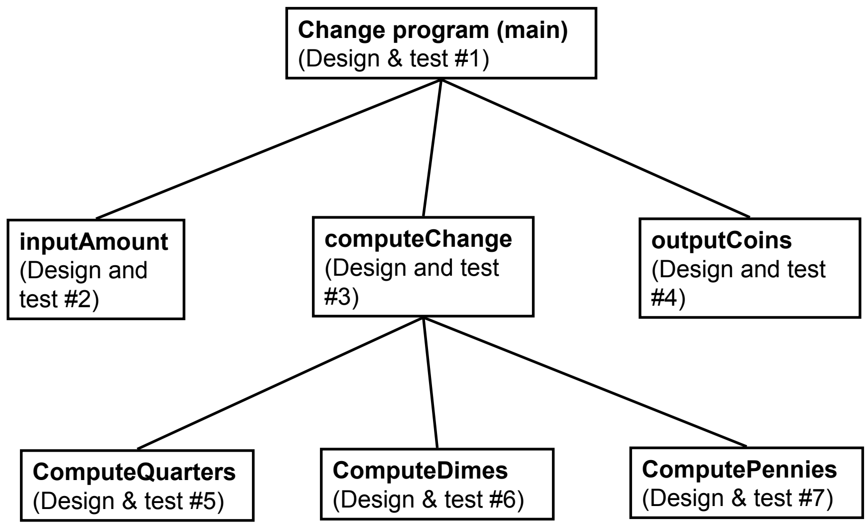
- However Action 2 (computing change) is still complex and may require further decomposition into sub-actions.
- One sensible decomposition is:
  - Sub-action 2A: Compute the number of quarters to be given out.
  - Sub-action 2B: Compute the number of dimes to be given out.
  - Sub-action 2C: Compute the number of pennies to be given out.

- Rules of thumb for designing functions:
  1. Each function should have one well defined task. If it doesn't then it may have to be decomposed into multiple sub-functions.
     a) Clear function: A function that prompts the user to enter the amount of money.
     b) Ambiguous function: A function that prompts for the amount of money and computes the number of quarters to be given as change.
  2. Try to avoid writing functions that are longer than one screen in size (again this is just a rule of thumb or guideline!)

Determining What Information Needs To Be Tracked

1. Amount of change to be returned
2. Number of quarters to be given as change
3. Number of dimes to be given as change
4. Number pennies to be given as change
5. The remaining amount of change still left (the value updates or changes as quarters, dimes and pennies are given out)

Outline Of The Program



First Implement Functions As Skeletons/Stubs

- After laying out a design for your program write functions as skeletons/stubs.
- (Don't type them all in at once).
- Skeleton function:
  - It's a outline of a function with a bare minimum amount that is needed to translate to machine (keywords required, function name, a statement to define the body – return values and parameters may or may not be included in the skeleton).

Code Skeleton: Change Maker Program

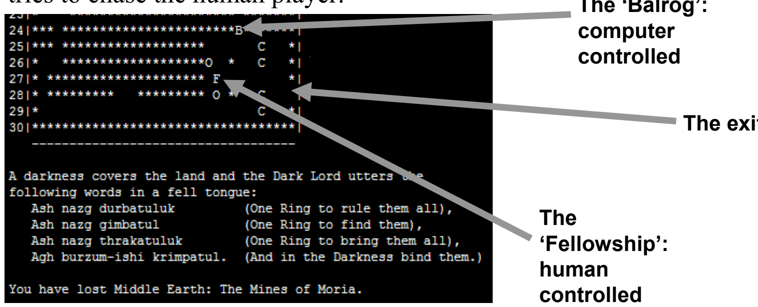
```
def inputAmount (amount):
    return amount

def computeQuarters (amount, amountLeft, quarters):
    return amountLeft, quarters

def computeDimes (amountLeft, dimes):
    return amountLeft, dimes

def computePennies (amountLeft, pennies):
    return pennies

def computeChange (amount, quarters, dimes, pennies):
    amountLeft = 0
    return quarters, dimes, pennies

def outputCoins (amount, quarters, dimes, pennies):
    print ""
```


Code Skeleton: Change Maker Program (2)

```
# MAIN FUNCTION
amount = 0
quarters = 0
dimes = 0
pennies = 0
```

How To Come With An Algorithm/Solution

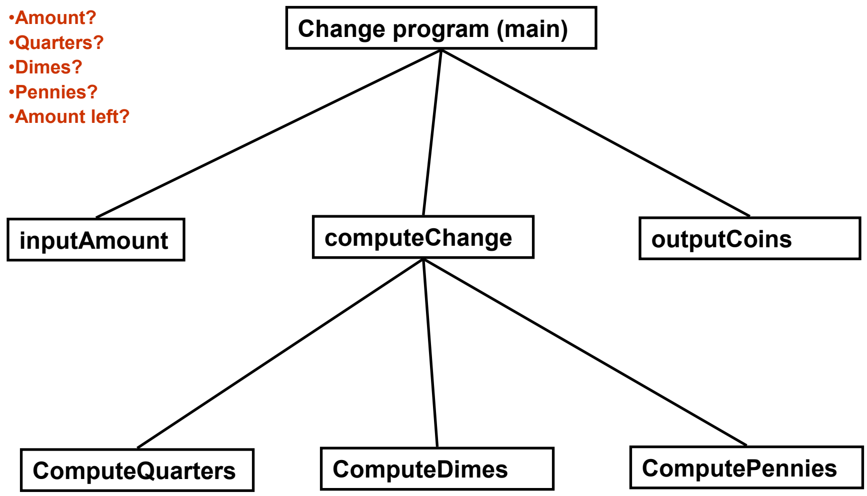
- An algorithm is the series of steps (not necessarily linear!) that provide the solution to your problem.
- If there is a physical analogy to the problem then try visualizing the problem using real world objects or scenarios.
  - E.g., Write a program that implements some rudimentary artificial intelligence. The program is a game that takes place on a 2D grid. The human player tries to find the exit and the computer controlled opponent tries to chase the human player.



How To Come With An Algorithm/Solution (2)

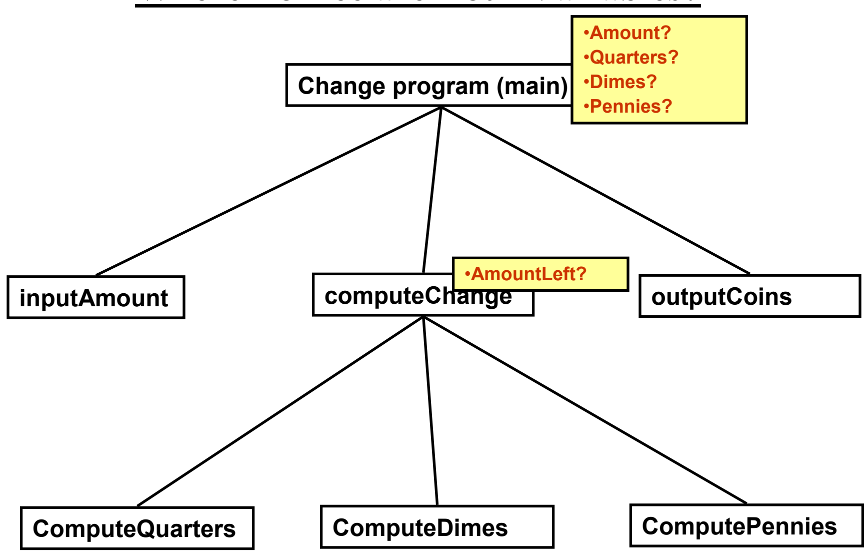
- If the problem is more abstract and you may be unable to come with the general solution for the program.
- Try working out a solution for a particular example and see if that solution can be extended from that specific case to a more generalized formula.

Where To Declare Your Variables?



t7

Where To Declare Your Variables?



t7 Try to help more with design:
For each module try to figure out what do we need to know going in and what do we need to know coming out

Remove the desing and test here (move it to just before slide no. 56 (design and test but greyed out).

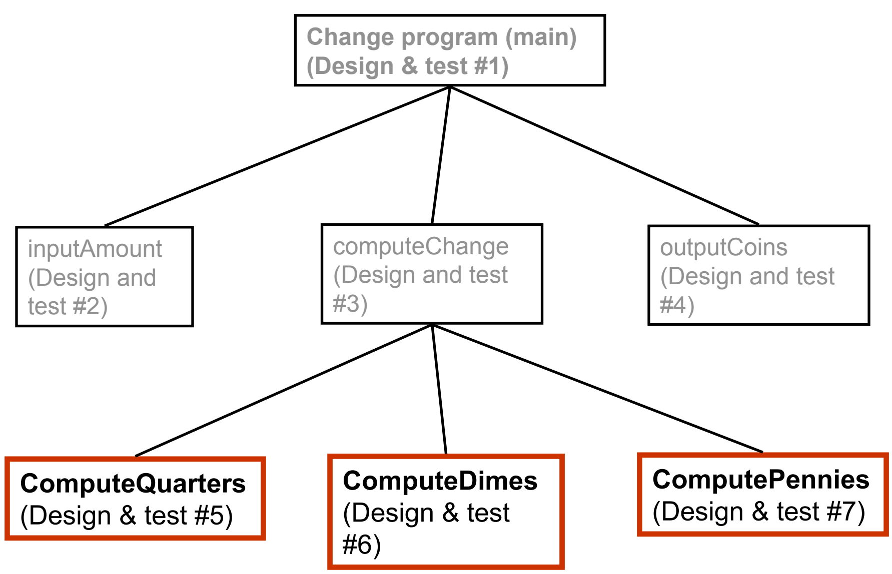
Also show design and test with alternative order that's also valid.
tamj, 3/28/2008

Implementing And Testing Input Functions

```
# Function definition
def inputAmount (amount):
    amount = input ("Enter an amount of change from 1 to 99 cents: ")
    return amount

# Testing the function definition
amount = inputAmount (amount)
print "amount:", amount
```

Test that your inputs were read in correctly DON'T ASSUME that they were!

Implementing And Testing The Compute Functions



Implementing And Testing ComputeQuarters

```
# Function definition
def computeQuarters (amount, amountLeft, quarters):
    quarters = amount / 25
    amountLeft = amount % 25
    return amountLeft, quarters


# Function test
amount = 0;
amountLeft = 0
quarters = 0
amount = input ("Enter amount: ")
amountLeft, quarters = computeQuarters (amount, amountLeft, quarters)
print "Amount:", amount
print "Amount left:", amountLeft
print "Quarters:", quarters
```

Check the program calculations against some hand calculations.

Functions And Scope

- The scope of an identifier (variable, constant) is where it may be accessed and used.
- **Example**:

```
def fun1 ():
    num = 10
        :
    print num

def fun2 ():
        :

# main
    :
```

'num' comes into scope (is visible and can be used)

(End of function): num goes out of scope, no longer accessible

Functions And Scope (2)

- The scope of an identifier (variable, constant) is where it may be accessed and used.
- **Example**:

```
def fun1 ():
    num = 10  ⎫
        :     ⎬ Scope of 'num'
    print num ⎭

def fun2 ():
        :

# main
    :
```

Scope of 'num'

Functions And Scope (3)

- The scope of an identifier (variable, constant) is where it may be accessed and used.
- **Example**:

```
def fun1 ():
    num = 10  ⎫
        :     ⎬ Scope of 'num'
    print num ⎭

def fun2 ():
    print num   <- 'num' is not defined

# main
    :
```

Scope of 'num'

'num' is not defined

Getting Around Scope Without Parameters?

- The scope of an identifier (variable, constant) is where it may be accessed and used.
- **Example**:

```
def fun1 ():
    num = 10
        :
    print num
```

Scope of 'num'

```
def fun2 ():
    num = 0
    print num
```

OK let's define 'num'

Oh no!...it's the wrong 'num'!!!

```
# main
   :
```


Yet Another Common Mistake: Not Saving Return Values

- Just because a function returns a value does not automatically mean the value will be usable by the caller of that function.

```
def fun ():
   return 1
```

This value has to be stored or used in some expression by the caller

- That is because return values have to be explicitly saved by the caller of the function.
- Example

```
def fun ():
    length = 4
    width = 3
    area = length * width
```

```
# MAIN
area = 0
fun ()
print area
```

```
# Fixed MAIN
area = 0
area = fun ()
print area
```

Why Employ Problem Decomposition And Modular Design

- Drawback
  - Complexity – understanding and setting up inter-function communication may appear daunting at first
  - Tracing the program may appear harder as execution appears to "jump" around between functions.
- Benefit
  - Solution is easier to visualize (only one part of a time)
  - Easier to test the program (testing all at once increases complexity)
  - Easier to maintain (if functions are independent changes in one function can have a minimal impact on other functions, if the code for a function is used multiple times then updates only have to made once)
  - Less redundancy, smaller program size (especially if the function is used many times throughout the program).

After This Section You Should Now Know

- How to write the definition for a function
  - How to write a function call
- How to pass information to and from functions via parameters and return values
- How and why to declare variables locally
- How to test functions and procedures
- How to design a program from a problem statement
  - How to determine what are the candidate functions
  - How to determine what variables are needed and where they need to be declared
  - Some approaches for developing simple algorithms (problem solving techniques)