Introduction to Computer Programming

Lecture #7 - Modular Programming I: Functions

What Are Functions?

- We have seen a few examples of procedures (in Python, we call them functions; we will explain this a little later):
  - **`print`**, which we have used to display output on the screen
  - **`input`**, which we have used to get input from the keyboard as strings.
  - **`random.randint()`**, which we have used to get a random numbers
- Procedures allow us to use software routines that have already been written (frequently by other people) in our programs.

  E.g., **`magic = random.randint(1, 100)`**

Why Use Procedures?

- Procedures offer several advantages when we write programs:
  - They allow us to concentrate on a higher level abstractions, without getting bogged down in details that we are not yet ready to handle.
  - They make it easier to divide the work of writing a program among several people.
  - They are re-usable; i. e., we write it once and can use it several times in a program and we can even copy it from one program to another.

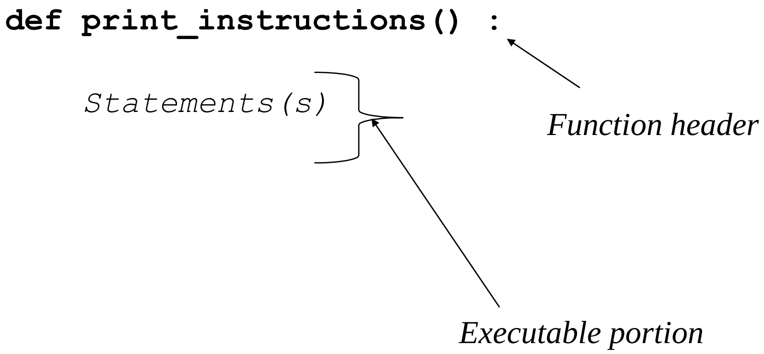
Simple Functions to print messages

- Let’s start with a simple function: Let’s a function that will print instructions for a user playing the “Magic Number” game:

```
#  printInstruction() - Print instructions for
#  the user
def print_instructions() :
    print("The object of the game is to find out")
    print("which number the computer has picked. The")
    print("computer will tell you if you guessed too")
    print("high a number or too low. "\
                 + "Try to get it with");
    print("as few guesses as possible.\n")
```

Simple Functions For Printing Messages

- The general form of the syntax is:



```
def print_instructions() :

        Statements(s)
```

Function header

Executable portion

Putting the Pieces Together

```
def print_instructions() :
    print("The object of the game is to find out")
    print("which number the computer has picked. The")
    print("computer will tell you if you guessed too")
    print("high a number or too low. "\
                           + "Try to get it with");
    print("as few guesses as possible.\n")

#  The magic number game has the user trying to
#  guess which number between 1 and 100 the
#  computer has picked

tries = 1;
print_instructions()

#  Use the random number function to pick a
#  number
magic = random.randint(1, 100)
```

```
#  Let the user make a guess
guess = int(input("Guess ?"))

while guess != magic :
  … …

#  If the user won, tell him/her
print("** Right!! ** ")
print(magic, " is the magic number\n");

#  Tell the user how many guesses it took
print("You took ", tries, " guesses\n")
```

What are parameters?

- A ***parameter*** is a value or a variable that is used to provide information to a function that is being called.
- If we are writing a function to calculate the square of a number, we can pass the value to be squared as a parameter:

```
printSquare(5);
printSquare(x)
```

actual parameter

- These are called actual parameters because these are the actual values (or variables) used by the function being called.

Formal Parameters

- Functions that use parameters must have them listed in the function header. These parameters are called ***formal parameters.***

formal parameter

```
def print_square(x)   :
   square = x*x
   print("The square of ", x, " is ",  square)
```

Parameter Passing

```
printSquare(5)

printSquare(x)

def print_square(x)   :
    square = x*x
    print("The square of ", x, " is ", square)
```

In both cases, calling the function requires copying the actual parameter's value where the function can use it. Initially, x has whatever value the actual parameter has.

Parameter Passing (continued)

```
printSquare(5)

def print_square(x)   :
    square = x*x
    print("The square of ", x, " is ",
square)
```

x *initially is set to* 5. *square is then set to the value of* x^2 *or* 5^2 *or* **25**.

Parameter Passing (continued)

```
printSquare(x)

def print_square(x)   :
    square = x*x
    print("The square of ", x, " is ",
square)
```

x *initially is set to whatever value x had in the main program. If x had the value 12, square is then set to the value of* x^2 *or* 12^2 *or* **144**.

Why parameters?

- Parameters are useful because:
  - They allow us to use the same function in different places in the program and to work with different data.
  - They allow the main program to communicate with the function and pass it whatever data it is going to use.
  - The same value can have completely different names in the main program and in the function.

The **`Squares`** Program

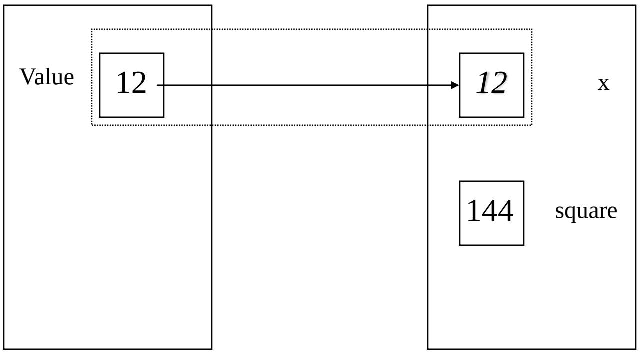
```
#  printSquare() - Prints the square of whatever
#                  value that it is given.
def print_square(x) :
    square = x*x
    print("The square of ", x, " is ",square)

#  A driver for the print_square function
#  Get a value and print its square
value_str = input("Enter a value ?")
value_float = float(value_str)
print_square(value_float)
```

the actual parameter in the function call

the formal parameter in the function header

Passing Parameters - When The User Inputs 12



Value

12

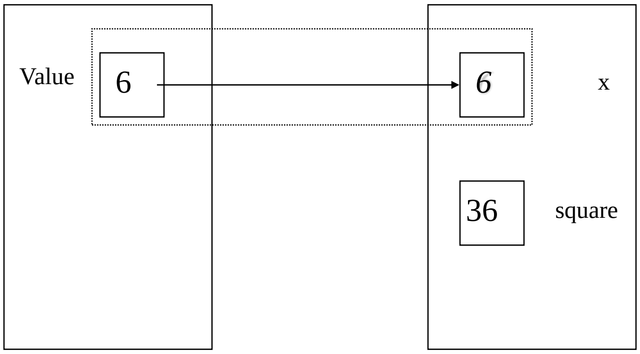
12

x

144

square

Passing Parameters - When The User Inputs 6



Value

6

6

x

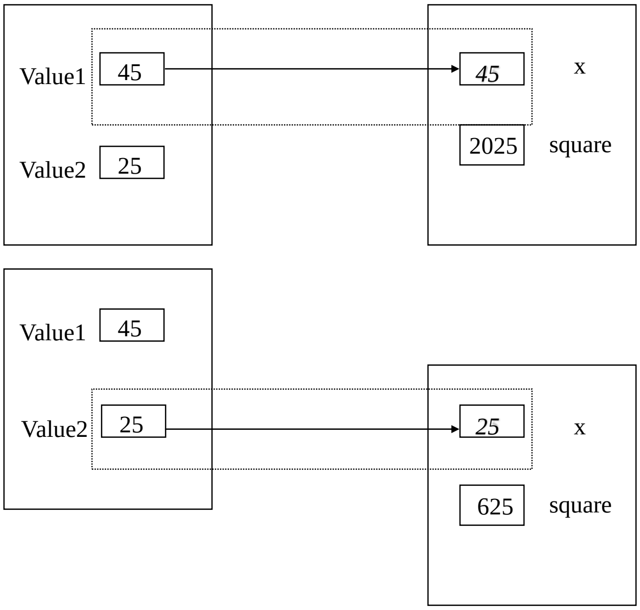
36

square

A Rewrite of **`main`**

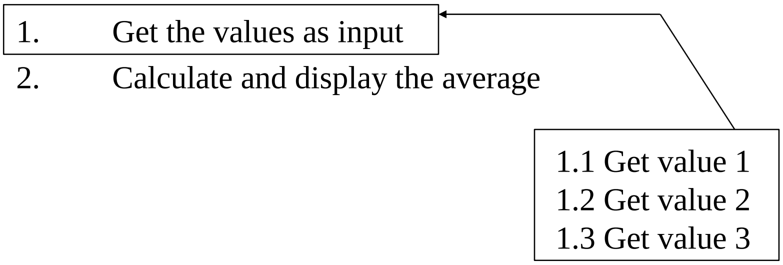
```
#  A driver for the print_square function
value1 = 45
value2 = 25
print_square(value1)
print_square(value2)
```

Passing Parameters - Using **`square`** Twice In One Program



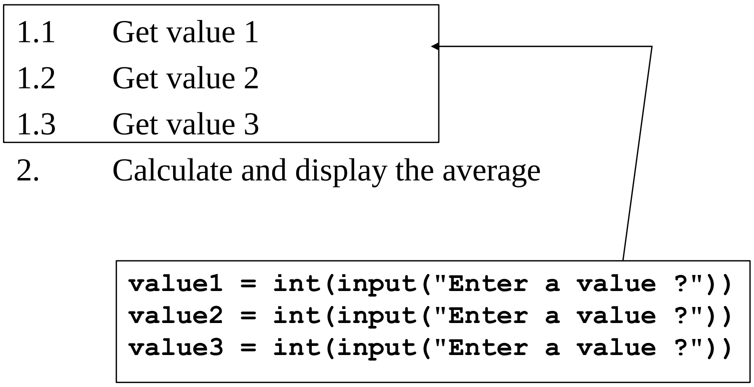
Calculating the Average of 3 Values Using a Function

- Let's re-examine how to find the average of 3 values. We have to:
    1. Get the values as input
    2. Calculate and display the average



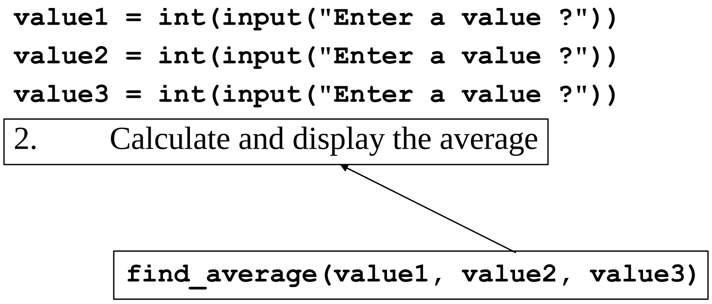
1. Get the values as input
2. Calculate and display the average

1.1 Get value 1
1.2 Get value 2
1.3 Get value 3



1.1 Get value 1
1.2 Get value 2
1.3 Get value 3
2. Calculate and display the average

```
value1 = int(input("Enter a value ?"))
value2 = int(input("Enter a value ?"))
value3 = int(input("Enter a value ?"))
```



```
value1 = int(input("Enter a value ?"))
value2 = int(input("Enter a value ?"))
value3 = int(input("Enter a value ?"))
```

2. Calculate and display the average

```
find_average(value1, value2, value3)
```

The **average3** Program

```
# Find the average of three numbers using a
function

#  find_average() -  Find the average of three
#                    numbers
def print_average(x, y, z)  :
    sum = x + y + z
    average = sum / 3
    print("The average is %3.1f\n" % average)

#  Main program - Find the average of three
#                 numbers using a function

#  Get the inputs
value1 = int(input("Enter a value ?"))
value2 = int(input("Enter a value ?"))
value3 = int(input("Enter a value ?"))
```

```
#  Call the function that calculates and
#  prints the average
print_average(value1, value2, value3)
```

Example – x to the nth power

- Let’s write a function to calculate x to the nth power and a driver for it (a main program whose sole purpose is to test the function.
- Our basic algorithm for the function:
  - Initialize (set) the product to 1
  - As long as n is greater than 0:
    - Multiply the product by x
    - Subtract one from n

power Program

```
#  A program to calculate 4-cubed using a
#  function called power

#  power() -  Calculates y = x to the nth power
def power(y, x, n)   :
    y = 1.0
    while n > 0 :
        y = y * x
        n = n - 1

    print("Our result is ", y)
```

```
# Main Program
#  Calculate 4-cubed using power
x = 4.0
n = 3
y = 1.0
power(y, x, n)
print("The answer is ", y)
```

The Output From **`power`**

```
Our result is 64
The answer is  1
```

Shouldn't these be the same numbers?

The problem is that communication using parameters has been one-way – the function being called listens to the main program , but the main program does not listen to the function.

Value Parameters

- The parameters that we have used all pass information from the main program to the function being called by copying the values of the parameters. We call this ***passing by value***, because the value itself is passed.
- Because we are using a copy of the value copied in another location, the original is unaffected.

Functions

- Some functions perform specific tasks and do not produce any one data item that seem to be their whole reason for existence.
- Other functions are all about producing some value or data item

Functions That Return Nothing

- Normally a function is expected to produce some ***<u>result</u>*** which is returns to the main program:

```
average = calcAverage(x, y, z);
```

- The data type of the function’s result is also called the function’s type.

Writing Functions That Return Results

- In Python, functions that return a result look a lot like functions that don’t return anything, with one key difference: there has to be a statement that indicates the value being returned:

```
def average3(a, b, c) :
    sum = a + b + c
    mean = sum / 3.0
    return mean
```

The result that we are returning is mean

Writing Functions That Return Results

- The syntax is:

  `return` *expression*

- Return statements have contain expressions, variables, constants or literals:

  ```
  return True
  return 35.4
  return sum
  return sum/3
  ```

Rewriting the `average3` Function

```
def average3(a, b, c) :
      sum = a + b + c
      return sum / 3.0
```

Maximum and Minimum

- Let’s write a pair of functions that find the minimum and maximum of two values ***a*** and ***b***.
- Initial algorithm for maximum:

  Return the larger of a and b:
- If we refine this:

```
1.1    IF a > b        return a

1.1    else    return b  //a < = b
```

•For minimum, we replace > with <

```
def maximum(x, y) :
    if x > y :
        return x
    else :
        return y
```

```
def minimum(x, y) :
    if x < y :
        return x
    else :
        return y
```

Rewriting the Payroll Program

```
#  A simple payroll program that uses a function
#  to calculate the gross pay

#  gross() -  Calculate the gross pay.
def gross(hours, rate):
    #  If hours exceed 40, pay time and a half
    if hours > 40 :
        pay = 40*rate + 1.5*rate*(hours-40)
    else :
        pay = rate * hours
    return pay
```

```
#  Main Program
#  Ask the user for payrate
rate = float(input("What is rate of pay for the
employee?"))

#  Enter the hours worked
hours = float(input("Enter the hours worked?"))

#  Get the gross pay
pay = gross(hours, rate)
print("Gross pay is $%4.2f\n" % pay)
```

return

- return serves two purposes:
  - It tells the computer the value to return as the result.
  - It tell the computer to leave the function immediately and return the main program.

```
Def gross(hours, rate) :
    #  If hours exceed 40, pay time and a half
    if hours > 40 :
      return(40*rate + 1.5*rate*(hours-40))
    return(rate*hours)
```

Rewriting pow

- We can make the pow function tell the main program about the change in **`y`** by having it return the value as the result:

```
def power(x, n) :
      … …
```

The rewritten **`pow`** program

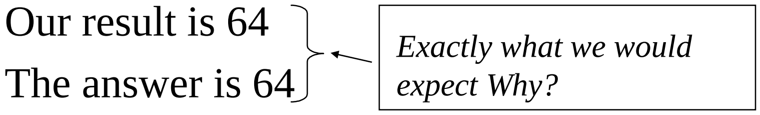
```
#  A program to calculate 4-cubed using a
#  function called power

#  power() -  Calculates y = x to the nth
#             power
def power(x, n) :
    prod = 1.0
    while n > 0 :
        prod = prod * x
        n = n - 1

    print("Our result is " , prod)
    return prod
```

```
#  Main program
x = 4.0
n = 3
y = power(x, n)
print("The answer is ", y)
```

The New Output From **`power`**



```
Our result is 64
The answer is 64
```

Exactly what we would expect Why?

Communication using the result is two-way – the function being called listens to the main program, but the main program listens to the function because data changes are explicitly passed back to the main program.

An Example – **`square2`**

- Let’s rewrite the square program so that the function calculates the square and passes its value back to the main program, which will print the result:

```
#  This illustrates how to use functions to
#  find the square of a value

#  findSquare() - Calculates the square of
#                 whatever value it is given.
def find_square(x) :
    square = x*x
    return square
```

```
#  Main Program - A driver for the findSquare
#                 method
value = float(input("Enter a value ?"))

square = find_square(value);
print("The square of ", value, " is ", square)
```

Comparing print_square and find_square

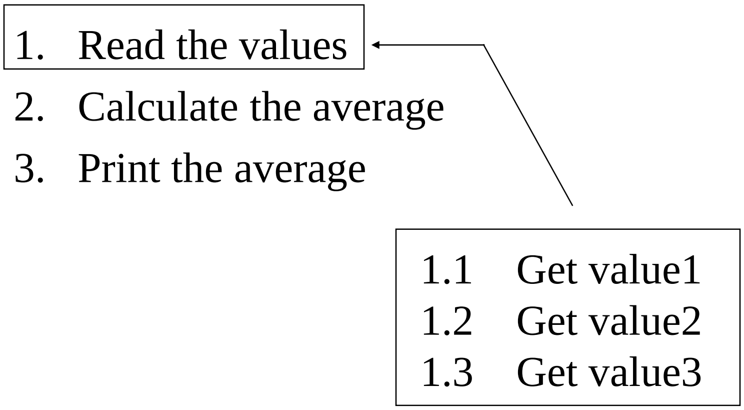
- What are the differences between **`print_square`** and **`find_square`**?
- **`print_square`**:
  - – uses value parameters
  - – prints the square; it doesn't have to pass that value to the main program
- **`find_square`**:
  - – Returns the result
  - – does not print the square; it must pass the value back to the main program

Example: Average3

- Let's write a program which will find the average of three numbers:
- Our algorithm is:

1. Read the values
2. Calculate the average
3. Print the average

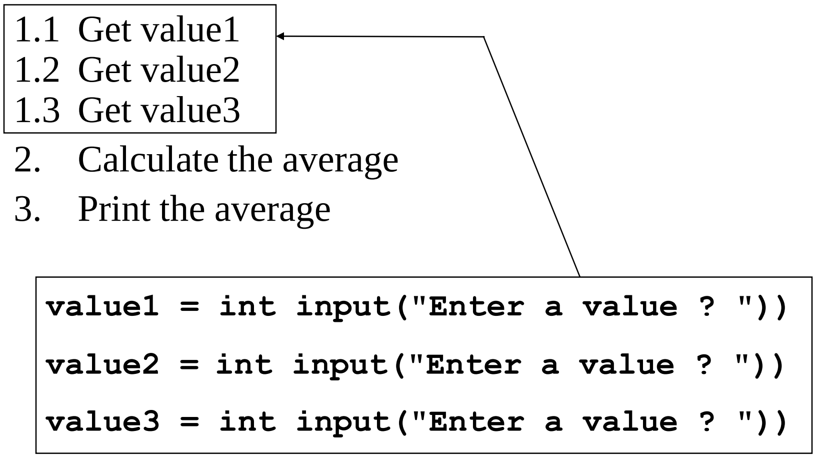
Refining **average3**'s algorithm



1. Read the values
2. Calculate the average
3. Print the average

1.1 Get value1
1.2 Get value2
1.3 Get value3

Refining **average3**'s algorithm (continued)



1.1 Get value1
1.2 Get value2
1.3 Get value3
2. Calculate the average
3. Print the average

```
value1 = int input("Enter a value ? "))

value2 = int input("Enter a value ? "))

value3 = int input("Enter a value ? "))
```

Refining **average3**'s algorithm (continued)

```
value1 = int input("Enter a value ? "))
value2 = int input("Enter a value ? "))
value3 = int input("Enter a value ? "))
```

2. Calculate the average
3. Print the average

```
average = find_average(value1, value2, value3)
```

Refining **average3**'s algorithm (continued)

```
value1 = int input("Enter a value ? "))
value2 = int input("Enter a value ? "))
value3 = int input("Enter a value ? "))
average = find_average(value1, value2, value3)
```

3. Print the average

```
print("The average is ", average)
```

Average3c.py

```
#  Find the average of three numbers using a
#  function

#  find_average() -  Find the average of three
#                    numbers
def find_average(x, y, z) :
    sum = x + y + z
    average = sum / 3.0
    return average
```

```
#  Main program
#  Get the inputs
value1 = int(input("Enter a value ? "))
value2 = int(input("Enter a value ? "))
value3 = int(input("Enter a value ? "))

#  Call the function that calculates the
#  average
average = find_average(value1, value2, value3)
print("The average is ", average)
```

A program to calculate Grade Point Average

Example - Ivy College uses a grading system, where the passing grades are A, B, C, and D and where F (or any other grade) is a failing grade. Assuming that all courses have equal weight and that the letter grades have the following numerical value:

Letter grade	Numerical value
A	4
B	3
C	2
D	1
F	0

write a program that will calculate a student's grade point average.

Let's Add– Dean's List

- Let's include within the program a method that will print a congratulatory message if the student makes the Dean's List.
- We will write a function **`deansList`** that will print the congratulatory message and another method **`printInstructions`**.

A program to calculate Grade Point Average

Input - The student's grades

Output - Grade point average and a congratulatory message (if appropriate)

Other information

"A" is equivalent to 4 and so on

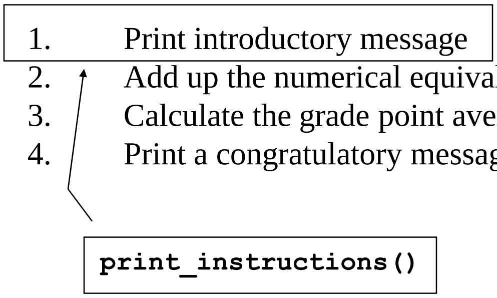
GPA = Sum of the numerical equivalents/ Number of grades

Our first step is to write out our initial algorithm:

1. Print introductory message
2. Add up the numerical equivalents of all the grades
3. Calculate the grade point average and print it out
4. Print a congratulatory message (if appropriate)

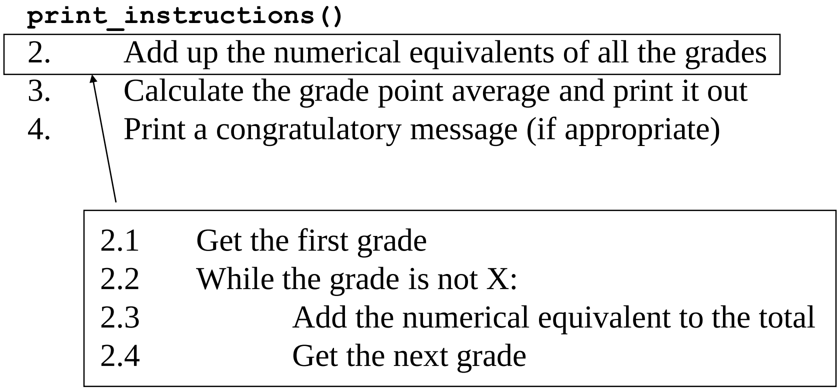
Refining the GPA Algorithm

1. Print introductory message
2. Add up the numerical equivalents of all the grades
3. Calculate the grade point average and print it out
4. Print a congratulatory message (if appropriate)



`print_instructions()`

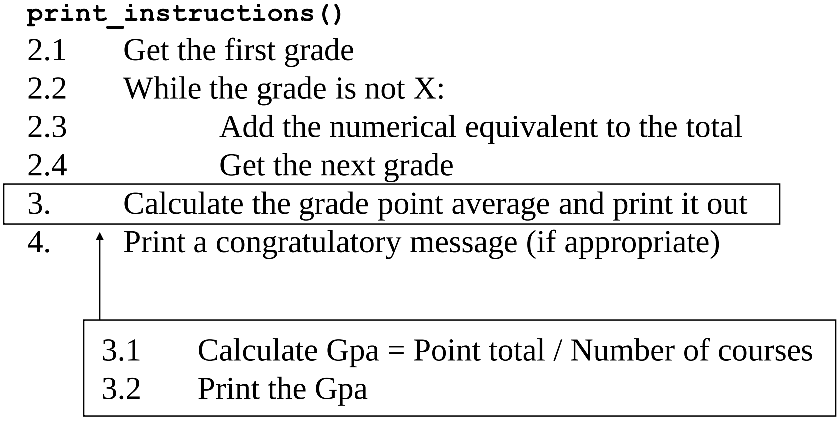
Refining the GPA Algorithm



```
print_instructions()
2.	Add up the numerical equivalents of all the grades
3.	Calculate the grade point average and print it out
4.	Print a congratulatory message (if appropriate)
```

```
2.1	Get the first grade
2.2	While the grade is not X:
2.3		Add the numerical equivalent to the total
2.4		Get the next grade
```

Refining the GPA Algorithm



```
print_instructions()
2.1	Get the first grade
2.2	While the grade is not X:
2.3		Add the numerical equivalent to the total
2.4		Get the next grade
3.	Calculate the grade point average and print it out
4.	Print a congratulatory message (if appropriate)
```

```
3.1	Calculate Gpa = Point total / Number of courses
3.2	Print the Gpa
```

Refining the GPA Algorithm

```
print_instructions()
2.1     Get the first grade
2.2     While the grade is not X:
2.3             Add the numerical equivalent to the total
2.4             Get the next grade
3.1     Calculate Gpa = Point total / Number of courses
3.2     Print the Gpa
4.      Print a congratulatory message (if appropriate)
```

```
deans_list(gpa)
```

Refining the GPA Algorithm

```
print_instructions()
2.1     Get the first grade
2.2     While the grade is not X:
2.3             Add the numerical equivalent to the total
2.4             Get the next grade
3.1     Calculate Gpa = Point total / Number of courses
3.2     Print the Gpa
deans_list(gpa)
```

```
total = total + convert_grade(grade)
num_courses = num_courses + 1
```

Refining the GPA Algorithm

```
print_instructions()
2.1     Get the first grade
2.2     While the grade is not X:
                total = total + convert_grade(grade)
                num_courses = num_courses + 1
2.4             Get the next grade
3.1     Calculate Gpa = Point total / Number of courses
3.2     Print the Gpa
deans_list(gpa)
```

```
input_str = input("What grade did you get in" \
           " your first class?")
grade = input_str[0]
```

Refining the GPA Algorithm

```
print_instructions()
input_str = input("What grade did you get in" \
           " your first class?")
grade = input_str[0]
2.2     While the grade is not X:
                total = total + convert_grade(grade)
                num_courses = num_courses + 1
2.4             Get the next grade
3.1     Calculate Gpa = Point total / Number of courses
3.2     Print the Gpa
deans_list(gpa)
```

```
while grade != 'X' :
```

Refining the GPA Algorithm

```
print_instructions()
input_str = input("What grade did you get in" \
           " your first class?")
grade = input_str[0]
while grade != 'X’ :
            total = total + convert_grade(grade)
            num_courses = num_courses + 1
2.4         Get the next grade
3.1     Calculate Gpa = Point total / Number of courses
3.2     Print the Gpa
deans_list(gpa)
```

```
#  Get the next course grade
input_str = input("What grade did you get "\
         "in the next class?")
grade = input_str[0]
```

Refining the GPA Algorithm

```
print_instructions()
input_str = input("What grade did you get in" \
           " your first class?")
grade = input_str[0]
while grade != 'X’ :
      total = total + convert_grade(grade)
      num_courses = num_courses + 1
      #  Get the next course grade
      input_str = input("What grade did you get "\
         "in the next class?")
      grade = input_str[0]
3.1     Calculate Gpa = Point total / Number of courses
3.2     Print the Gpa
deans_list(gpa)
```

```
gpa = total / num_courses
```

Refining the GPA Algorithm

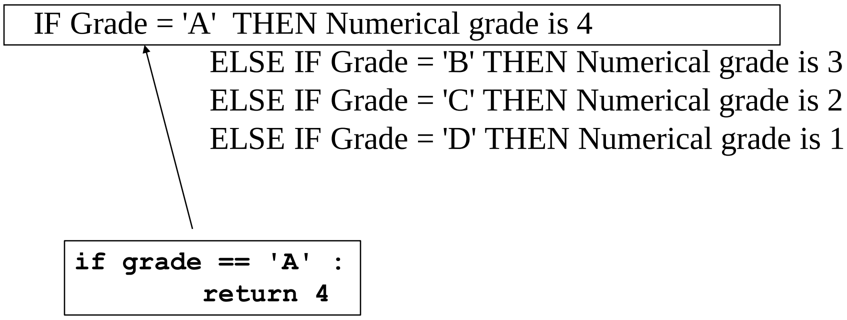
```
print_instructions()
input_str = input("What grade did you get in" \
           " your first class?")
grade = input_str[0]
while grade != 'X’ :
      total = total + convert_grade(grade)
      num_courses = num_courses + 1
      #  Get the next course grade
      input_str = input("What grade did you get "\
         "in the next class?")
      grade = input_str[0]
gpa = total / num_courses
3.2    Print the Gpa
deans_list(gpa)
```

```
print("Your grade point average is %4.2f\n" % gpa)
```

The Main Program

```
print_instructions()
input_str = input("What grade did you get in" \
           " your first class?")
grade = input_str[0]
while grade != 'X’ :
      total = total + convert_grade(grade)
      num_courses = num_courses + 1
      #  Get the next course grade
      input_str = input("What grade did you get "\
         "in the next class?")
      grade = input_str[0]
gpa = total / num_courses
print("Your grade point average is %4.2f\n" % gpa)
deans_list(gpa)
```

Converting the Grade

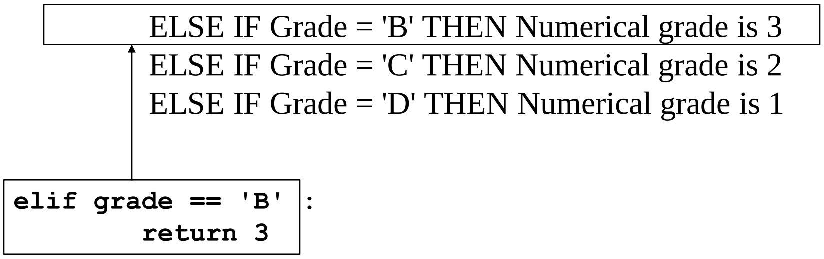


IF Grade = 'A' THEN Numerical grade is 4
ELSE IF Grade = 'B' THEN Numerical grade is 3
ELSE IF Grade = 'C' THEN Numerical grade is 2
ELSE IF Grade = 'D' THEN Numerical grade is 1

```
if grade == 'A' :
        return 4
```

Converting the Grade

```
if grade == 'A' :
        return 4
```

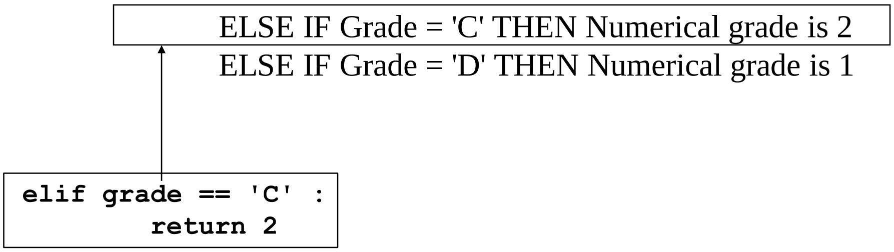


ELSE IF Grade = 'B' THEN Numerical grade is 3
ELSE IF Grade = 'C' THEN Numerical grade is 2
ELSE IF Grade = 'D' THEN Numerical grade is 1

```
elif grade == 'B' :
        return 3
```

Converting the Grade

```
if grade == 'A' :
        return 4
elif grade == 'B' :
        return 3
```



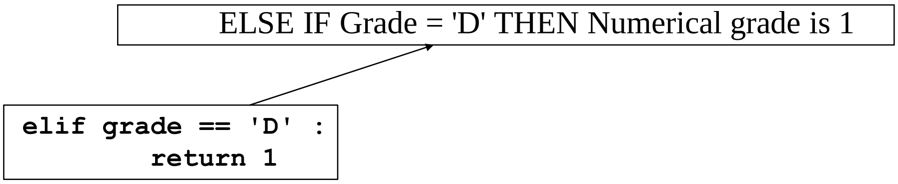
ELSE IF Grade = 'C' THEN Numerical grade is 2

ELSE IF Grade = 'D' THEN Numerical grade is 1

```
elif grade == 'C' :
        return 2
```

Converting the Grade

```
if grade == 'A' :
        return 4
elif grade == 'B' :
        return 3
elif grade == 'C' :
        return 2
```



ELSE IF Grade = 'D' THEN Numerical grade is 1

```
elif grade == 'D' :
        return 1
```

Converting the Grade

```
def convert_grade(letter_grade) :
    if grade == 'A' :
        return 4
    elif grade == 'B' :
        return 3
    elif grade == 'C' :
        return 2
    elif grade == 'D' :
        return 1
    elif grade != 'F' :
        print("A grade of ", grade
            + " is assumed to be an F\n")
    return 0
```

The **`deanLists()`** method

```
IF gpa >= 3.2
      Print congratulatory message
```

The Entire **`DeansList`** Program

```
#  Calculates a grade point average assuming
#  that all courses have the same point value
#  and that A, B, C and D are passing grades and
#  that all other grades are failing.

#  print_instructions() - Prints instructions
#                        for the user
def print_instructions() :
    #  Print an introductory message
    print("This program calculates your grade " \
            + "point average assuming that all")
    print("courses have the same point value.")
    print("It also assumes that grades of " \
            + A, B, C and D")
    print("are passing and that all other "\
            + "grades are failing.")
    print("To indicate that you are finished, " \
                 + "enter a grade of \'X\'\n\n")
```

```
#  convert_grade - Convert an A to a 4, B to a 3,
#                  etc. so it can be added it to
#                  the total
def convert_grade(letter_grade) :
    if letter_grade == 'A' :
        return 4
    elif letter_grade == 'B' :
        return 3
    elif letter_grade == 'C' :
        return 2
    elif letter_grade == 'D' :
        return 1
    elif letter_grade != 'F' :
        print("A grade of ", grade
            + " is assumed to be an F\n")
    return 0
```

```
#  deans_list() - Print a message if (s)he made
#                 dean's list
def deans_list(gpa) :
    if gpa >= 3.2 :
        print("Congratulations!! You made" \
          + " dean\'s list!!\n\n");

# Main Program
num_courses = 0
total = 0;

#  Tell the user how to use the program
print_instructions()

#  Get the first course grade
input_str = input("What grade did you get in" \
           " your first class?")
grade = input_str[0]




#  Add up the numerical equivalents of
#  the grades
while grade != 'X' :
    total = total + convert_grade(grade)
    num_courses = num_courses + 1

    #  Get the next course grade
    input_str = input("What grade did you get "\
             "in the next class?")
    grade = input_str[0]

#  Divide the point total by the number of
#  classes to get the grade point average
#  and print it.
gpa = total / num_courses
print("Your grade point average is %4.2f\n" % gpa)
deans_list(gpa)
```

Revising the ***Nim*** program

- Let’s revise the Nim program to use functions.
- We’ll create the following functions to subdivide the work:

```
print_instructions
get_move
plan_move
update_sticks
```

Nim2.java

```
#  printInstructions() - Print instructions for
#                        the player
def print_instructions() :
    #  Print the instructions
    print("There are seven (7) sticks on the " \
            ‶         + table.")
    print("Each player can pick up one, two , or" \
                      + " three sticks")
    print("in a given turn.  A player cannot pick"\
                      + " up more than");
    print("three stick nor can a player pass.\n")
```

```
#  get_move() - Get the player's next move,
#              testing to ensure that it is legal
#              and that there are enough sticks
#              on the table.
def get_move(sticksLeft) :
    pick_up = 0
    move = False

    #  How many sticks is the user taking
    while not move :
        pick_up = int(input("How many sticks do " \
                       + " you wish to pick up ?"))
```

```
        #  Make sure its 1, 2 or 3
        if pick_up < 1 or pick_up > 3 :
            print(pick_up, \
                " is not a legal number of sticks")
        #  Make sure that there are enough sticks
        #  on the table
        elif pick_up > sticksLeft :
            print("There are not ", pick_up, \
                 " sticks on the table")
        else :
            move = True
    return pick_up
```

```
# plan_move() - Plan the computer's next move
def plan_move(sticksLeft)  :
    reply = 0
    #  Plan the computer's next move
    if sticksLeft == 6 or sticksLeft == 5 \
          or sticksLeft ==2 :
        reply = 1
    elif sticksLeft == 4 :
        reply = 3
    elif sticksLeft == 3 :
        reply = 2
    return reply
```

```
#  updateSticks() - Update the count of sticks
#                   left on the table and
#                   determine if either the
#                   player or the computer has
#                   won.
def updateSticks(sticksLeft, reply) :
    #  If neither player won, get ready for the
    #  next move
    sticksLeft = sticksLeft - reply
    print("The computer picked up ", \
               reply, " sticks.")
    print("There are now ", sticksLeft, \
             " sticks left on the table.\n\n\n")
    return sticksLeft
```

```
#  Main Program
#  Play the game Nim against the computer
sticksLeft = 7
pick_up = 0
reply = 0
winner = False
answer = ' '

print_instructions()

#  Find out if the user wants to go first or
#  second

while answer.lower() != 'f’  \
                and answer.lower() != 's' :
    answerString = input("Do you wish to go" \
                        + " (f)irst or (s)econd ?")
    answer = answerString[0]
```

```
#  If the user goes second, have the computer
#  take two sticks.
if  answer.lower() == 's' :
    reply = 2
    sticksLeft = sticksLeft - reply
    print("The computer took ", reply, \
          " sticks leaving ",sticksLeft,\
          " on the table.")
else :
    #  If the user goes first, tell him how many
    #  sticks are on the table
    print("There are ", sticksLeft, \
                     " on the table.")
```

```
#  As long as there is no winner, keep playing
while not winner :
    pick_up = get_move(sticksLeft)

    #  Take the sticks off the table
    sticksLeft = sticksLeft - pick_up

    #  See if the user won
    if sticksLeft == 1 :
        print("Congratulations!  You won!")
        winner = True;
    #  See if the user lost
    elif sticksLeft == 0 :
        print("Sorry, the computer has ", \
                "won - you have lost...")
        winner = True
    else :
        reply = plan_move(sticksLeft)

    if not winner :
        sticksLeft = updateSticks(sticksLeft, reply)
```

Preconditions and Postconditions

- Preconditions – are conditions that we expect and require to be true ***before entering*** the procedure
- Postconditions– are conditions that we expect and require to be true ***after exiting*** the procedure
- Examples in square3:
  - getinput has a **postcondition** that a value was read in and that the value is set.
  - find average has a **precondition** that all value1, value2 and value al have values.