

CSC 171- Introduction to Computer Programming

Lecture #4 - Looping Around Loops II : Conditional Loops

The Problem with Counting Loops

- Many jobs involving the computer require repetition, and that this can be implemented using loops.
- Counting loops allows us to perform a statement or a block of statements a certain number of times.
- The problem is that we do not always know exactly how many times to perform the statements in a loop in every situation.

The Problem with Counting Loops (continued)

- Let's take another look at our payroll program:
 - We do not always know how payroll records that we have.
 - It isn't very convenient to have to count the records, especially if it's a big number.
 - Wouldn't it be better if we could keep going until we enter some special value to tell the computer to stop?

Conditional Loops

- Conditional loops allow us to do this.
- **Conditional** loops keep repeating as long as some **condition** is true (or until some condition **becomes** true).
- Steps in solving a problem that involve **while**, **until**, **as long as** indicate a conditional loop.

While Loops

- The most common form of conditional loops are *while* loops.
- In Python, they have the form:

```
while condition :  
    statement(s)
```

A simple example – `PickPositive.py`

```
# A simple example of how while works  
  
# Get your first number  
number = int(input("Hi there. Pick a positive  
integer"))  
  
# Keep reading number as long as they are  
# positive  
while number > 0 :  
    number = int(input("Pick another positive  
integer"))  
  
print(number, " is not a positive integer")
```

The test average program revisited

- Let's take another look at our program to calculate test average:
- Write a program that can calculate a test average and grade for any number of tests. The program will finish when the user types in a grade of -1.

Sentinel Value

- Often conditional loops continue until some special value is encountered in the input which effectively tells the program to stop running the loop. This is called a *sentinel value* because it is the value for which we are watching.
- -1 is the sentinel value in the GPA algorithm's main loop

The *TestAverage* Algorithm

1. As long as there are more grades, add them to the total
2. Divide the total by the number of courses
3. Print the corresponding letter grade

Refining The *TestAverage* Algorithm

1. As long as there are more grades, add them to the total
2. Divide the total by the number of courses
3. Print the corresponding letter grade

- 1.1 Get the first grade
- 1.2 As long as the grade is not -1, add it to the total and get the next grade

Refining The *TestAverage* Algorithm

1.1 Get the first grade

1.2 As long as the grade is not -1, add it to the total
and get the next grade

3. Divide the total by the number of courses

4. Print the corresponding letter grade

1.2.1 while the grade is not -1:

1.2.2 Add it to the total

1.2.3 Get the next grade

1.2.4 Add one to the number of tests

Refining The *TestAverage* Algorithm

1.1 Get the first grade

1.2.1 while the grade is not -1:

1.2.2 Add it to the total

1.2.3 Get the next tests

1.2.4 Add one to the number of courses

2. Divide the total by the number of courses

3. Print the corresponding letter grade

```
print("What grade did you get on your first test  ?")
print("Enter -1 to end")
thisGrade = int(input ( ))
```

Refining The *TestAverage* Algorithm

```
print
    ("What grade did you get on your first test
     ?")
print("Enter -1 to end")
thisGrade = int(input ())
```

1.2.1 while the grade is not -1:

1.2.2 Add it to the total

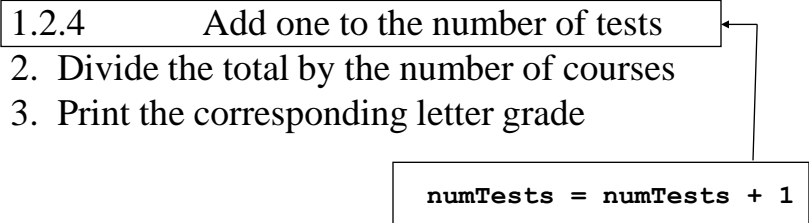
1.2.3 Get the next grade

1.2.4 Add one to the number of tests

2. Divide the total by the number of courses

3. Print the corresponding letter grade

```
numTests = numTests + 1
```



Refining The *TestAverage* Algorithm

```
print
    ("What grade did you get on your first test
     ?")
print("Enter -1 to end")
thisGrade = int(input ())
```

1.2.1 while the grade is not -1:

1.2.2 Add it to the total

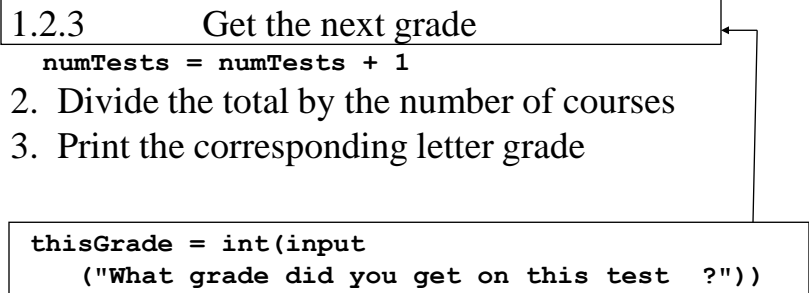
1.2.3 Get the next grade

```
numTests = numTests + 1
```

2. Divide the total by the number of courses

3. Print the corresponding letter grade

```
thisGrade = int(input
    ("What grade did you get on this test ?"))
```



Refining The *TestAverage* Algorithm

```
print
    ("What grade did you get on your first test
     ?")
print("Enter -1 to end")
thisGrade = int(input ())
```

1.2.1 while the grade is not -1:

1.2.2 Add it to the total

```
    thisGrade = int(input
        ("What grade did you get on this test ?"))
    numTests = numTests + 1
```

2. Divide the total by the number of courses
3. Print the corresponding letter grade

`total = total + thisGrade`

Refining The *TestAverage* Algorithm

```
print
    ("What grade did you get on your first test
     ?")
print("Enter -1 to end")
thisGrade = int(input ())
while thisGrade != sentinelGrade :
    total = total + thisGrade
    thisGrade = int(input
        ("What grade did you get on this test ?"))
    numTests = numTests + 1
```

2. Divide the total by the number of courses

3. Print the corresponding letter grade

`testAverage = total/numTests`

Refining The *TestAverage* Algorithm

... ..

```
testAverage = total/numTests
```

```
3. Print the corresponding letter grade
```

```
if testAverage >= 90 :
    courseGrade = 'A'
elif testAverage >= 80 :
    courseGrade = 'B'
elif testAverage >= 70 :
    courseGrade = 'C'
elif testAverage >= 60 :
    courseGrade = 'D';
else :
    courseGrade = 'F'
```

The *TestAverage* Program

```
# Calculates the average test grade and
# converts it to a letter grade assuming that
# A is a 90 average, B is an 80 average and so
# on.

sentinelGrade = -1

# Initially the number of test is 0
numTests = 0

# Initially, the total is 0
total = 0

# Get the first grade
print("What grade did you get on your first test ?")
print("Enter -1 to end")
thisGrade = int(input ())
```

```
# Add up the test grades
while thisGrade != sentinelGrade :
    # Make sure that the grades are valid percentages
    total = total + thisGrade
    numTests = numTests + 1
    thisGrade = int(input("What grade did you get on this
test  ?"))

# Find the average
testAverage = total/numTests

# Find the letter grade corresponding to the average
if testAverage >= 90 :
    courseGrade = 'A'
elif testAverage >= 80 :
    courseGrade = 'B'
elif testAverage >= 70 :
    courseGrade = 'C'
elif testAverage >= 60 :
    courseGrade = 'D';
else :
    courseGrade = 'F'

# Print the results
print("Your test average is ", testAverage)
print("Your grade will be ", courseGrade);
```

Payroll program revisited


- Let's revisit the payroll program.
- Instead of counting up the payroll records so we can count the number of times we go through the loop, why not use some sentinel value in the last entry to tell the program when we're finished?
- Since no one will ever make \$0.00 per hour, we'll use a pay rate of 0 as our sentinel value in the revised payroll program.

Our Initial Payroll Algorithm

1. Display instructions for the user
2. Keep processing payroll records as long as there are more.

Refining the Payroll Algorithm

1. Display instructions for the user
2. Keep processing payroll records as long as there are more.

- 2.1 Get the first pay rate
 - 2.2 while The Rate is positive:
 - 2.3 Process payroll record
 - 2.4 Get the next pay rate
 - 2.5 Print final count
- 

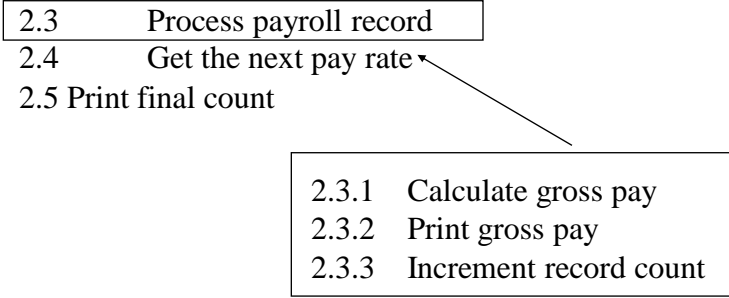
Refining the Payroll Algorithm

1. Display instructions for the user
- 2.1 Get the first pay rate
- 2.2 while The Rate is positive:
- 2.3 Process payroll record
- 2.4 Get the next pay rate
- 2.5 Print final count

```
while rate > 0.0 :
```

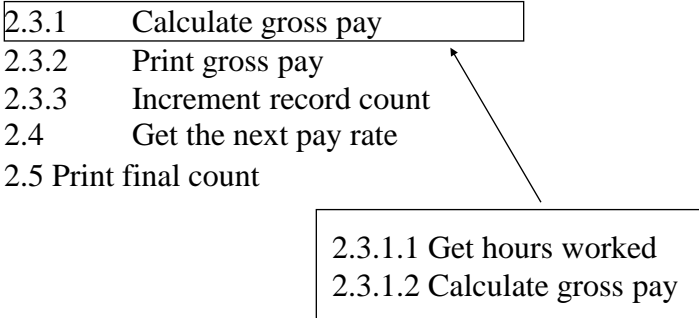
Refining the Payroll Algorithm

1. Display instructions for the user
- 2.1 Get the first pay rate
- 2.2 **while rate > 0.0 :**
- 2.3 Process payroll record
- 2.4 Get the next pay rate
- 2.5 Print final count

- 
- 2.3.1 Calculate gross pay
 - 2.3.2 Print gross pay
 - 2.3.3 Increment record count

Refining the Payroll Algorithm

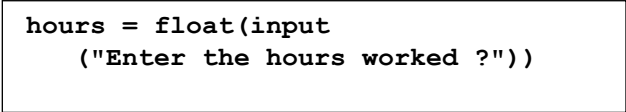
1. Display instructions for the user
- 2.1 Get the first pay rate
- while rate > 0.0 :**
- 2.3.1 Calculate gross pay
- 2.3.2 Print gross pay
- 2.3.3 Increment record count
- 2.4 Get the next pay rate
- 2.5 Print final count

- 
- 2.3.1.1 Get hours worked
 - 2.3.1.2 Calculate gross pay

Refining the Payroll Algorithm

1. Display instructions for the user
- 2.1 Get the first pay rate
- while rate > 0.0 :**
- 2.3.1.1 Get hours worked
- 2.3.1.2 Calculate gross pay
- 2.3.2 Print gross pay
- 2.3.3 Increment record count
- 2.4 Get the next pay rate
- 2.5 Print final count

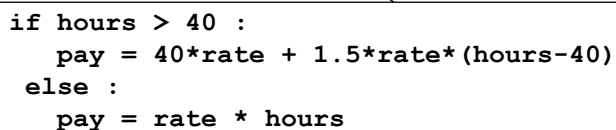
```
hours = float(input
("Enter the hours worked ?"))
```



Refining the Payroll Algorithm

1. Display instructions for the user
- 2.1 Get the first pay rate
- while rate > 0.0 :**
- hours = float(input
- ("Enter the hours worked ?"))
- 2.3.1.2 Calculate gross pay
- 2.3.2 Print gross pay
- 2.3.3 Increment record count
- 2.4 Get the next pay rate
- 2.5 Print final count

```
if hours > 40 :
    pay = 40*rate + 1.5*rate*(hours-40)
else :
    pay = rate * hours
```



Refining the Payroll Algorithm

1. Display instructions for the user
 - 2.1 Get the first pay rate
- ```
while rate > 0.0 :
 hours = float(input
 ("Enter the hours worked ?"))
 if hours > 40 :
 pay = 40*rate + 1.5*rate*(hours-40)
 else :
 pay = rate * hours
```
- 2.3.2 Print gross pay
  - 2.3.3 Increment record count
  - 2.4 Get the next pay rate
  - 2.5 Print final count

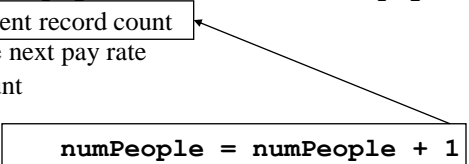
```
print("Gross pay is $%4.2f\n\n" %(pay))
```



## Refining the Payroll Algorithm

1. Display instructions for the user
  - 2.1 Get the first pay rate
- ```
while rate > 0.0 :  
    hours = float(input  
        ("Enter the hours worked ?"))  
    if hours > 40 :  
        pay = 40*rate + 1.5*rate*(hours-40)  
    else :  
        pay = rate * hours  
    print("Gross pay is $%4.2f\n\n" %(pay))
```
- 2.3.3 Increment record count
 - 2.4 Get the next pay rate
 - 2.5 Print final count

```
numPeople = numPeople + 1
```



Refining the Payroll Algorithm

1. Display instructions for the user
 - 2.1 Get the first pay rate
- ```
while rate > 0.0 :
 hours = float(input
 ("Enter the hours worked ?"))
 if hours > 40 :
 pay = 40*rate + 1.5*rate*(hours-40)
 else :
 pay = rate * hours
 print("Gross pay is $%4.2f\n\n" %(pay))
 numPeople = numPeople + 1
```
- 2.4 Get the next pay rate
- 2.5 Print final count

```
rate = float(input
 ("What is the pay rate for the next employee ? "))
```

## Refining the Payroll Algorithm

1. Display instructions for the user
  - 2.1 Get the first pay rate
- ```
while rate > 0.0 :  
    hours = float(input  
        ("Enter the hours worked ?"))  
    if hours > 40 :  
        pay = 40*rate + 1.5*rate*(hours-40)  
    else :  
        pay = rate * hours  
    print("Gross pay is $%4.2f\n\n" %(pay))  
    numPeople = numPeople + 1  
    rate = float(input  
        ("What is the pay rate for the next employee ? "))
```
- 2.5 Print final count

Refining the Payroll Algorithm

```
1. Display instructions for the user
2.1 Get the first pay rate
while rate > 0.0 :
    ... ..
    rate = float(input
        ("What is the pay rate for the next employee ? "))
2.5 Print final count
```

```
numPeople = 0

# Display instructions
print("To stop the program, enter a pay rate",
      " of zero or less \n\n")
```

Refining the Payroll Algorithm

```
numPeople = 0

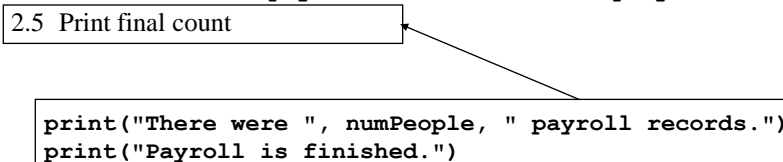
# Display instructions
print("To stop the program, enter a pay rate",
      " of zero or less \n\n")
2.1 Get the first pay rate
while rate > 0.0 :
    ... ..
    rate = float(input
        ("What is the pay rate for the next employee ? "))
2.5 Print final count
```

```
rate = float(input(
    "What is rate of pay for the employee ? "))
```

Refining the Payroll Algorithm

```
numPeople = 0

# Display instructions
print("To stop the program, enter a pay rate",
      " of zero or less \n\n")
rate = float(input(
    "What is rate of pay for the employee ? "))
while rate > 0.0 :
    ... ..
    rate = float(input
        ("What is the pay rate for the next employee ? "))
2.5 Print final count
print("There were ", numPeople, " payroll records.")
print("Payroll is finished.")
```



The Payroll Program

```
# Processes a payroll for a given number of
# employees. The user indicates that (s)he is
# finished by entering a pay rate that is zero
# or negative.

numPeople = 0

# Display instructions
print("To stop the program, enter a pay rate",
      " of zero or less \n\n")

# Ask the user for the first employee's
# pay rate
rate = float(input
    ("What is rate of pay for the employee ? "))
```

```

# Calculate gross salary for everyone on the
# payroll
while rate > 0.0 :
    # Enter the hours worked
    hours = float(input("Enter the hours worked ?"))

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

print("Gross pay is $%4.2f\n\n" %(pay))
numPeople = numPeople + 1

```

Compound Interest program revisited

- Our earlier program showed how much interest is compounded over a given number of years.
- Let's see how long your money would have to earn interest to reach a million dollars

Redesigning Our Compound Interest Program

Input – Input deposit

Output – The final year and exact balance

Other information

$$\text{New Principle} = (1 + \text{Interest Rate}) * \text{Old Principle}$$

Initial Algorithm:

- Set the initial principle at \$24
- For every year since 1625, add 5% interest to the principle until the principle reaches \$1,000,000 and print the principle every twenty years and when it reaches \$1 million
- Print the values for the final year

Refining The Compound Interest Algorithm

1. Set the initial principle at \$24
2. For every year since 1625, add 5% interest to the principle until the principle reaches \$1,000,000 and print the principle every twenty years and when it reaches \$1 million
3. Print the values for the final year ↗

2.1 Set Year to 1625
2.2 While Principle < \$1,000,000
2.3 Add 5% Interest to the Principle
2.4 If the Year % 20 = 5
2.5 then print the principle

Refining The Compound Interest Algorithm

1. Set the initial principle at \$24
2. For every year since 1625, add 5% interest to the principle until the principle reaches \$1,000,000 and print the principle every twenty years and when it reaches \$1 million
3. Print the values for the final year ↗

```
2.1 Set Year to 1625
2.2 While Principle < $1,000,000
2.3     Add 5% Interest to the Principle
2.4     If the Year % 20 = 5
2.5         then print the principle
```

Refining The Compound Interest Algorithm

1. Set the initial principle at \$24
- 2.1 Set Year to 1625
- 2.2 While Principle < \$1,000,000
- 2.3 Add 5% Interest to the Principle
- 2.4 If the Year % 20 = 5
- 2.5 then print the principle
3. Print the values for the final year

```
while principle < target :
```

Refining The Compound Interest Algorithm

1. Set the initial principle at \$24
- 2.1 Set Year to 1625
- while principle < target :**
- 2.3 Add 5% Interest to the Principle
- 2.4 If the Year % 20 = 5
- 2.5 then print the principle
3. Print the values for the final year

```
interest = rate * principle;
principle = principle + interest;
```

Refining The Compound Interest Algorithm

1. Set the initial principle at \$24
- 2.1 Set Year to 1625
- while principle < target :**
- interest = rate * principle;**
- principle = principle + interest;**
- 2.4 If the Year % 20 = 5
- 2.5 then print the principle
3. Print the values for the final year

```
if year % 20 == 5 :
    print(
        "year = %4d\tinterest = %13.2f\tprinciple = %15.2f\n"
        %(year, interest, principle))
```

Refining The Compound Interest Algorithm

1. Set the initial principle at \$24
- 2.1 Set Year to 1625

```
while principle < target :  
    interest = rate * principle;  
    principle = principle + interest;  
    if year % 20 == 5 :  
        print(  
            "year = %4d\tinterest = %13.2f\tprinciple = %15.2f\n"  
            %(year, interest, principle))
```

3. Print the values for the final year

```
print(  
    "year = %4d\tinterest = %13.2f\tprinciple = %15.2f\n"  
    %(year, interest, principle))
```

Refining The Compound Interest Algorithm

1. Set the initial principle at \$24

- 2.1 Set Year to 1625

... ..

```
print(  
    "year = %4d\tinterest = %13.2f\tprinciple = %15.2f\n"  
    %(year, interest, principle))
```

```
principle = 24
```

Refining The Compound Interest Algorithm

```
principle = 24
```

```
2.1 Set Year to 1625
```

```
... ..
```

```
print(
```

```
"year = %4d\tinterest = %13.2f\tprinciple = %15.2f\n"
```

```
%(year, interest, principle))
```

```
Year = 1625
```

The Revised Compound Interest Program

```
# Calculate the interest that the Canarsie Indians
# could have accrued if they had deposited the $24
# in an bank account at 5% interest.
year = 1625
rate = 0.05
target = 1000000

# Set the initial principle at $24
principle = 24

# For every year since 1625, add 5% interest
# to the principle until the principle
# reaches $1,000,000
# Print the principle every twenty years
# and when it reaches $1 million
# There has to be two fixed places for the
# principle
```



```

interest = 0
while principle < target :
    interest = rate * principle
    principle = principle + interest

    if year % 20 == 5 :
# Use 15 places for printing the principle
    print(
        "year = %4d\tinterest = %13.2f\tprinciple = %15.2f\n"
        %(year, interest, principle))
    year = year + 1

# Print the values for the final year
print(
    "year = %4d\tinterest = %13.2f\tprinciple = %15.2f\n"
    %(year, interest, principle))

```

Magic Number Problem

- The magic number game involves guessing a number and with each wrong guess, the player is told “too high” or “too low”. The goal is to guess the number in the smallest number of tries.
- We need a method for having the computer pick a number at random for the player to guess.
- We will need to learn about how to use “library functions” to provide us with the magic number.

Designing the Magic Number Algorithm

Input – The player’s guess(es)

Output – A clue (“too high” or “too low”) and the number of guesses that it took.

Initial Algorithm

1. Use the random number function to pick a number
 - Let the player make a guess
 - As long as the player hasn’t guessed the number, give the appropriate clue and let him/her guess again.
 - Print the number of tries

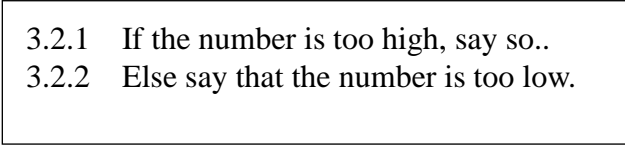
Refining the Magic Number Algorithm

1. Use the random number function to pick a number
2. Let the player make a guess
3. As long as the player hasn’t guessed the number, give the appropriate clue and let him/her guess again.
4. Print the number of tries

- | | |
|-----|---|
| 3.1 | While the guess isn’t the magic number: |
| 3.2 | Give the appropriate clue |
| 3.3 | Increment the count of tries |
| 3.4 | Let the player make another guess. |

Refining the Magic Number Algorithm

1. Use the random number function to pick a number
2. Let the player make a guess
- 3.1 While the guess isn't the magic number:
 - 3.2 Give the appropriate clue
 - 3.3 Increment the count of tries
 - 3.4 Let the player make another guess.
4. Print the number of tries

- 
- 3.2.1 If the number is too high, say so..
 - 3.2.2 Else say that the number is too low.

Refining the Magic Number Algorithm

1. Use the random number function to pick a number
2. Let the player make a guess
- 3.1 While the guess isn't the magic number:
 - 3.2.1 If the number is too high, say so..
 - 3.2.2 Else say that the number is too low.
 - 3.3 Increment the count of tries
 - 3.4 Let the player make another guess.
4. Print the number of tries



```
while guess != magic :
```

1. Use the random number function to pick a number
2. Let the player make a guess

```
while guess != magic :
```

3.2.1 If the number is too high, say so..

3.2.2 Else say that the number is too low.

- 3.3 Increment the count of tries
 - 3.4 Let the player make another guess.
4. Print the number of tries

```
if guess > magic :  
    print(".. Wrong .. Too high\n")  
else :  
    print(".. Wrong .. Too low\n")
```

1. Use the random number function to pick a number
2. Let the player make a guess

```
while guess != magic :
```

```
    if guess > magic :  
        print(".. Wrong .. Too high\n")  
    else :  
        print("..Wrong .. Too low\n")  
        # Let the user make another guess  
        guess = int(input("Guess ?"))
```

3.3 Increment the count of tries

3.4 Let the player make another guess.

4. Print the number of tries

```
tries = tries + 1
```

1. Use the random number function to pick a number
2. Let the player make a guess

```
while guess != magic :  
    if guess > magic :  
        print(".. Wrong .. Too high\n")  
    else :  
        print(".. Wrong .. Too low\n")  
        # Let the user make another guess  
        guess = int(input("Guess ?"))  
        tries = tries + 1
```

3.4 Let the player make another guess.

4. Print the number of tries

```
guess = int(input("Guess ?"))
```

1. Use the random number function to pick a number
2. Let the player make a guess

```
while guess != magic :  
    if guess > magic :  
        print(".. Wrong .. Too high\n")  
    else :  
        print(".. Wrong .. Too low\n")  
        # Let the user make another guess  
        guess = int(input("Guess ?"))  
        tries = tries + 1
```

```
guess = int(input("Guess ?"))
```

4. Print the number of tries

```
print("** Right!! ** ")  
print(magic, " is the magic number\n");  
  
# Tell the user how many guesses it took  
print("You took ", tries, " guesses\n");
```

1. Use the random number function to pick a number

2. Let the player make a guess

```
while guess != magic :  
    if guess > magic :  
        print(".. Wrong .. Too high\n")  
    else :  
        print(".. Wrong .. Too low\n")  
    # Let the user make another guess  
    guess = int(input("Guess ?"))  
    tries = tries + 1  
print("*** Right!! ** ")  
print(magic, " is the magic number\n");  
print("You took ", tries, " guesses\n");
```

```
guess = int(input("Guess ?"))
```

import and Python Modules

- It is frequently helpful to be able to use software routines that have already been written for common tasks.
- A ***library*** is a collection of code that someone else wrote and translated.
- A ***standard library*** is a library that is part of the language. Standard libraries are expected to be included with a Python system.
- Python's standard libraries are organized into modules. Each of these must be imported before its components can be used in a program.

`import` and the `Random` Module

- To use the random number function, we need to include `import random`
- This tells the computer it needs to use the `random` module.
- A module will include data types and procedures that we will need to use. We make it available by writing `import random` at the beginning of the program
- The name of the random number function that we want is `randint(start, finish)` – it will provide a random integer value in the range start to finish.
- In our program, the range will be 1 to 100.

The Magic Number Program

```
import random

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

tries = 1;

# 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 :
    # Otherwise tell him whether it's too high
    # or too low
    if guess > magic :
        print(".. Wrong .. Too high\n")
    else :
        print(".. Wrong .. Too low\n")
        # Let the user make another guess

    guess = int(input("Guess ?"))
    tries = tries + 1

# 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");

```

Declaring Boolean Constants

- If we want to work with *true* and *false* we can work with **boolean** variables.
- We can write:

```

boolean    married = true;
... ..
if (married)
    System.out.println("The employee is
        married\n");

```


not operator

- Sometimes we want to test to see if a condition is ***not*** true.
- We can do this by using the **not** operator:

```
if not married :  
    print("Do you want to bring a date?")
```

and and or Operators

- Sometimes there may be two or more conditions to consider. For this reason we have the **and** and **or** operators.
- If we have two variables, **p**, **q** :
 - Both **p** and **q** must be true for **p and q** to be true.
 - **p or q** is true unless both **p** and **q** are false.

Nim

- The game Nim starts out with seven sticks on the table.
- Each player takes turns picking up 1, 2 or 3 sticks and cannot pass.
- Whoever picks up the last stick loses (the other player wins).

The Nim Problem

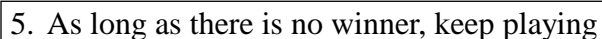
- Input
 - The number of sticks the player is picking up
- Output
 - The number of sticks on the table
 - Who won (the player or the computer)
- Other Information
 - Whoever leaves 5 sticks for the other player can always win if they make the right followup move:
 - If the other player takes 1, you pick up 3
 - If the other player takes 2, you pick up 2
 - If the other player takes 3, you pick up 1

Designing The Nim Algorithm

1. Print the instructions
2. Set the number of stick at 7 and initialize other values
3. Find out if the user wants to go first or second
4. If the user goes second, have the computer take two sticks and the user goes second, have the computer take two sticks.
5. As long as there is no winner, keep playing

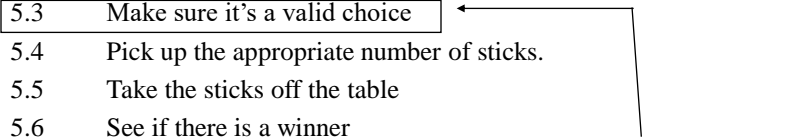
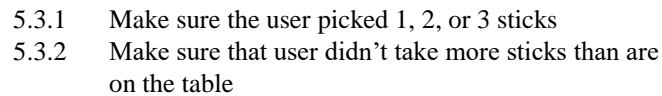
1. Print the instructions
2. Set the number of stick at 7 and initialize other values
3. Find out if the user wants to go first or second
4. If the user goes second, have the computer take two sticks and the user goes second, have the computer take two sticks.

5. As long as there is no winner, keep playing



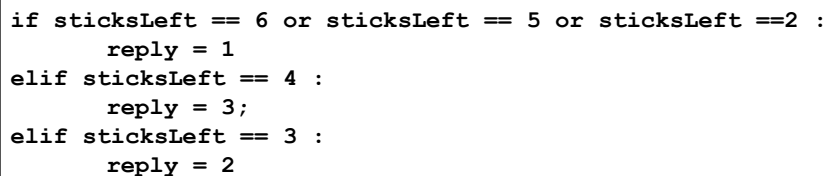

- | | |
|-----|---|
| 5.1 | While there is no winner |
| 5.2 | Find out how many sticks the user is taking |
| 5.3 | Make sure it's a valid choice |
| 5.4 | Pick up the appropriate number of sticks. |
| 5.5 | Take the sticks off the table |
| 5.6 | See if there is a winner |

1. Print the instructions
2. Set the number of stick at 7 and initialize other values
3. Find out if the user wants to go first or second
4. If the user goes second, have the computer take two sticks and the user goes second, have the computer take two sticks.
- 5.1 While there is no winner
 - 5.2 Find out how many sticks the user is taking
 - 5.3 Make sure it's a valid choice
 - 5.4 Pick up the appropriate number of sticks.
 - 5.5 Take the sticks off the table
 - 5.6 See if there is a winner

- 
- 
- 5.3.1 Make sure the user picked 1, 2, or 3 sticks
 - 5.3.2 Make sure that user didn't take more sticks than are on the table

.....

- 5.1 While there is no winner
 - 5.2 Find out how many sticks the user is taking
 - 5.3.1 Make sure the user picked 1, 2, or 3 sticks
 - 5.3.2 Make sure that user didn't take more sticks than are on the table
 - 5.3 Make sure it's a valid choice
 - 5.4 Pick up the appropriate number of sticks.
 - 5.5 Take the sticks off the table
 - 5.6 See if there is a winner



```
if sticksLeft == 6 or sticksLeft == 5 or sticksLeft ==2 :
    reply = 1
elif sticksLeft == 4 :
    reply = 3;
elif sticksLeft == 3 :
    reply = 2
```

```

... ..
5.1 While there is no winner
... ..
    if sticksLeft == 6 or sticksLeft == 5 \
        or sticksLeft == 2 :
        reply = 1
    ... ..
5.5 Take the sticks off the table
5.6 See if there is a winner

```

```

elif sticksLeft == 1 :    {
    print("Congratulations!  You won!")
    winner = True

elif sticksLeft == 0 :    {
    print(" Sorry, the computer has won ",\
        "- you have lost...")
    winner = True
}

```

checking for valid input

- Very often, people will enter data that is not valid.
- An example of this is a negative number of hours worked, a name that contains numbers, a ten-digit zip code, etc.
- It is very important to learn how to check for bad data and what to do when you encounter it.

... ..

5.1 While there is no winner

5.2

5.3 Make sure it's a valid choice

```
    if sticksLeft == 6 or sticksLeft == 5
        or sticksLeft == 2 :
        reply = 1
    ... ..
    else if sticksLeft == 0 :
        ... ..
```

5.5 Take the sticks off the table

5.3.1 Make sure the user picked 1, 2, or 3 sticks

5.3.2 Make sure that user didn't take more sticks than are on the table

The *Nim* Program

```
# A program for the class game of Nim

# Print the instructions
print("There are seven (7) sticks on the table.")
print("Each player can pick up one, two , or ")
print ("three sticks in a given turn. A player")
print("cannot pick up more than three stick nor")
print("can a player pass.\n")

# Initialize values
sticksLeft = 7
pickUp = 0
reply = 0
winner = False
answer = " "
```

```

# Find out if the user wants to go
# first or second
while answer != 'f' and answer != 'F' \
    and answer != 's' and answer != 'S' :
    answer = input("Do you wish to go " +
        "(f)irst or (s)econd ?")
    answer = answer[0]

# If the user goes second, have the computer
# take two sticks.
if answer == 's' or answer == 'S' :
    reply = 2
    sticksLeft = sticksLeft - reply
    print("The computer took ", reply, \
        " sticks leaving ", sticksLeft, " on the table.")

# If the user goes first, tell him how many
# sticks are on the table
else :
    print("There are ", sticksLeft, " on the table.")

# As long as there is no winner, keep playing
while not winner :
    move = False

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

        # Make sure its 1, 2 or 3
        if pickUp < 1 or pickUp > 3 :
            println(pickUp, \
                " is not a legal number of sticks")

        # Make sure that there are
        # enough sticks on the table
        elif pickUp > sticksLeft :
            println("There are not ", pickUp, \
                " sticks on the table" )
        else:
            move = True

```

```

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

# 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

# See if the user won
elif 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

# If neither happened,
# get ready for the next move
if not winner :
    sticksLeft = sticksLeft - reply
    print("The computer picked up ", reply, \
        " sticks.")
    print("There are now ", sticksLeft, \
        " sticks left on the table.\n\n")

```