# Programming Logic & Pseudocode

Python Bootcamp, Day 1

Anna Rosen

# Programming 101

Computer programming involves having the user formulate commands that the computer can run for a specific purpose.

The computer executes commands line by line (i.e., sequence) and program control (i.e., what lines to execute) is controlled by true-false statements.

This involves using formal logic to control how the computer program makes decisions.

# Pseudocode

Pseudocode is a type of structured english for describing algorithms (a sequence of steps).

It allows the designer to focus on the logic and structure of their algorithm without making the algorithm language specific.

# Pseudocode

Pseudocode can be broken down into five components:

- Variables
- Assignment
- Input/output
- Selection
- Repetition

Always decide on a descriptive name for the program.

# Pseudocode: General Structure

Program ProgramName:
    statements to execute
END.

# Variables

A variable has a name, data type, and a value. ALWAYS use identifying variable names.

Assignment is the physical act of placing a value into a variable. It can be used by simply placing a value or with an expression.

```
x = 5                   (datatype int)
y = 10.1                (datatype float)
z  = x+y                (datatype float)
myName = “Anna”         (datatype string)
```

# Input/Output

Input/output both deal with an outside source. The two main sources are the user (i.e., someone entering text in real-time) or from a file.

Output - Write / Display / print

Input - Read / get / input

# Selection (Conditional Logic)

We use logic in our everyday lives.

```
if it is raining outside then
    bring an umbrella
```

```
if my wallet has $40 then
    go to store
else go to ATM then
    go to store
```

# Programming Logic

Conditional logic gives the computer the ability to control the flow of your program.

It allows the user to control which statements to execute based on true/false statements.

true/false values are known as booleans

Boolean results are obtained by comparing different variables
Note that: True = 1, False = 0

# Comparison Operators

| Operator | Description | Example |
|---|---|---|
| == | Checks if the value of two operands are equal or not, if yes then condition becomes true. | (a == b) is not true. |
| != | Checks if the value of two operands are equal or not, if values are not equal then condition becomes true. | (a != b) is true. |
| <> | Checks if the value of two operands are equal or not, if values are not equal then condition becomes true. | (a <> b) is true. This is similar to != operator. |
| > | Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true. | (a > b) is not true. |
| < | Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true. | (a < b) is true. |
| >= | Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true. | (a >= b) is not true. |
| <= | Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true. | (a <= b) is true. |

# The *if-then-else* statement

The *if* statement allows you to control if a program enters a section of code based on whether a given condition is true

*Pseudocode format*
if-then statement
```
IF (boolean condition) THEN
      do command 1
      do command 2
      etc.
ENDIF
```

Adding an *else* statement allows you to execute code when the condition is false

*Pseudocode format*
if-then-else statement
```
IF (boolean condition) THEN
      do commands
ELSE
      do other commands
ENDIF
```

# Embedded *if-then-else* statements

The nice thing about the *if-then-else* statement is that you can add and embed as many if-elses as possible

*Pseudocode format*

if-then statement

```
IF (boolean condition) THEN
      do commands
ELSE IF (boolean condition) THEN
      do commands
ELSE IF (boolean condition) THEN
      IF (boolean condition) THEN
            do commands
      ENDIF
ENDIF
```

# Testing multiple boolean conditions

a=10, b=20

| Operator | Description | Example |
|---|---|---|
| and | Called Logical AND operator. If both the operands are true then then condition becomes true. | (a and b) is true. |
| or | Called Logical OR Operator. If any of the two operands are non zero then then condition becomes true. | (a or b) is true. |
| not | Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false. | not(a and b) is false. |

Logical operators allow you to test multiple conditions

# Programming Logic Examples

a=10, b=20, c=15

if-then statement (1)

```
IF (b-c < a) THEN
      print “Hello”
ENDIF
```

if-then statement (2)

```
IF (b-c < a and c > 20) THEN
      print “Hello”
ENDIF
```

if-then-else statement (1)

```
IF (a+c > b+c) THEN
      print “a > b”
ELSE
      print “a < b”
ENDIF
```

if-then-else statement (2)

```
IF (not(a+c > b+c)) THEN
      print “a < b”
ELSE
      print “a > b”
ENDIF
```

# Programming Logic Examples

a=10, b=20, c=15

Embedded if-then-else statements

```
IF (a == b) THEN
      print “a equals b”
ELSE IF (a > b or c != b) THEN
      IF (a > b) THEN
            print “a > b”
      ELSE IF (c != b) THEN
            print “c != b”
      ENDIF
ELSE IF ( not(a) ) THEN
      print “a=0”
ENDIF
```

Embedded if-then-else statements

```
IF (FALSE) THEN
      print “a equals b”
ELSE IF (FALSE or TRUE) THEN
      IF (FALSE) THEN
            print “a > b”
      ELSE IF (TRUE) THEN
            print “c != b”
      ENDIF
ELSE IF (TRUE) THEN
      print “a=0”
ENDIF
```

# Repetition (loops)

Loops repeat a statement a certain number of times or until a condition is met.

In a repetition problem

- Count is initialized
- Some condition is tested
- some variable is incremented

# Repetition (loops)

Loops repeat a statement a certain number of times or until a condition is met.

A *FOR* loop is a "counting" loop

*Pseudocode*
FOR loop
FOR iteration bounds
    sequence
ENDFOR

*Examples:*
FOR each month of the year
    print "month"
ENDFOR

FOR X =1 to 10
    IF game[X] == 0 THEN
        Do nothing
    ELSE
        game[X] =100
    ENDIF
    increment x by 1
ENDFOR

A *WHILE* loop is a repetition with a conditional test at its beginning.

This loop is only run if the condition is true and will terminate once it is false.

*Pseudocode*
WHILE loop
WHILE (boolean condition)
    sequence
ENDWHILE

*Examples:*
x=0
WHILE(x<10)
    x = x+1
ENDWHILE

# Warning about loops: Infinite Loops

What is wrong with these examples?

```
x=0
WHILE(x<10)
    print “Hello world!”
    x = x-1
ENDWHILE
```

```
x=0
WHILE(x >= 0)
    IF (x < 0) THEN
        x = x +1
ENDIF
ENDWHILE
```

# More Flow Controls: *Break* and *Continue*

Break statements allow you to break out of the smallest enclosing of a *for* or *while* loop

```
x=0
WHILE(x < 10)
    IF (x > 5) THEN
        BREAK
    ENDIF
    x = x+1
ENDWHILE
```

# More Flow Controls: *Break* and *Continue*

Continue statements allow you to not complete the following statements in the loop but to *continue* to the next iteration of the smallest enclosing of a *for* or *while* loop

```
x=0
WHILE(x < 10)
    IF (x == 5) THEN
        x = x + 2
        CONTINUE
    ENDIF
    x = x+1
ENDWHILE
```

# Pseudocode Example 1

Consider the problem of searching for an entry in a phonebook with only one condition

```
requiredEntry = “Anna Rosen”
N = firstEntry
WHILE N is NOT(requiredEntry)
    N = nextEntry
ENDWHILE
```

# Pseudocode Example 2

Consider the problem of checking if a number is even or odd.

```
PROGRAM IsOddOrEven:
    Read A
    IF (A/2 gives a reminder) THEN
        PRINT “It’s odd”
    ELSE
        PRINT “It’s Even”
    ENDIF
END.
```

# Pseudocode Example 3

Consider the problem of checking if a number from a list of numbers is even or odd.

```
PROGRAM IsOddOrEvenList:
    Read Alist
    FOR A in Alist
        IF (A/2 gives a reminder) THEN
            PRINT “It’s odd”
        ELSE
            PRINT “It’s Even”
        ENDIF
    ENDFOR
END.
```

# Proper formatting (subjective)

Code (and pseudocode) can be annoying to read at times so it is crucial to use good proper formatting

Always use indentation! Indent styles control flow and blocks of code.

Use plenty of white space (but not too much)

Code in “paragraph” form

Comments, comments, COMMENTS!

# Class Activity 1

Now it’s time for you to come up with some pseudocode examples. Let’s think of some problems we can tackle.

# Class Activity 2

Write pseudocode for sorting a list of numbers (where the number is arbitrary) in ascending or descending order.