SFU SIMON FRASER UNIVERSITY
ENGAGING THE WORLD

CMPT 120: Introduction to Computing Science and Programming 1

Control Structures: Loops

python™

Reminders

One-Stop Access To Course Information

- **Course website: One-stop access** to all course information.

  http://www2.cs.sfu.ca/CourseCentral/120/liaqata/WebSite/index.html

- Course Outline
- Exam Schedule
- Python Info
- CourSys/Canvas link
- Learning Outcomes
- Office Hours
- Textbook links
- and more…
- Grading Scheme
- Lab/Tutorial Info
- Assignments

- **Canvas**: Discussions forum - https://canvas.sfu.ca/courses/39187
- **CourSys**: Assignments submission, grades - www.coursys.sfu.ca

How to Learn in This Course?

A **Attend** Lectures & Labs

R **Read** / review Textbook/Slides/Notes

R **Reflect** and ask Questions

O **Organize** – your learning activities on weekly basis,

and finally…

W **Write** Code, Write Code, and Write Code.

Deliverables

1. Deliverables are due by the given date and time.
2. For the course, we are using IDLE to write and run our Python code.
3. You can use the CSIL lab computers outside your lab hours.
4. Plan ahead your assignments and other deliverables. Computer crash, network problems etc. are not acceptable excuses for delays in deliverables.
5. You may use online Python interpreters for running and testing your codes, such as:

   https://repl.it/languages/Python3

Labs

1. Each lab has an assigned TA.
2. Attend your assigned lab and show your work to your TA for the participation marks.
3. Class enrolments and lab swaps are closed now.

Course Topics

1. General introduction
2. Algorithms, flow charts and pseudocode
3. Procedural programming in Python
4. **Data types and Control Structures**
5. **Functions and Fundamental algorithms**
6. **Binary encodings**
7. **Basics of computability and complexity**
8. **Basics of Recursion**
9. **Subject to time availability:**
   - **Basics of Data File management**

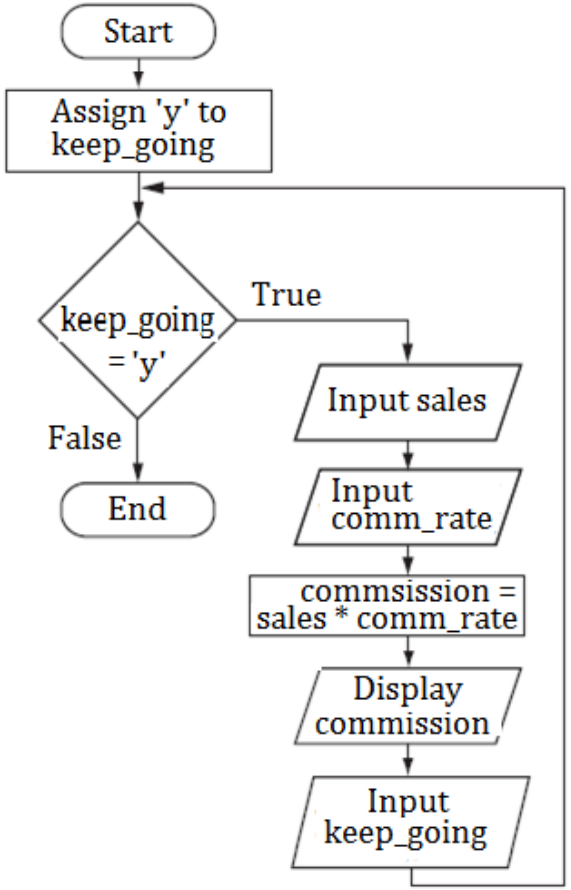
Today's Topics

1. Introduction to Loops: Repetition Structures
   a. The `for` Loop: a Count-Controlled Loop
   b. continue & break
   c. The `while` Loop: a Condition-Controlled Loop
2. Sentinels
3. Input Validation Loops
4. Nested Loops

1

Introduction to Loops: Repetition Structures - while

Liaqat Ali, Summer 2018.




```
# This program calculates sales commissions.

# Create a variable to control the loop.
keep_going = 'y'

# Calculate a series of commissions.
while keep_going == 'y':
    # Get a salesperson's sales and commission rate.
    sales = float(input('Enter the amount of sales: '))
    comm_rate = float(input('Enter the commission rate: '))

    # Calculate the commission.
    commission = sales * comm_rate

    # Display the commission.
    print('The commission is {}$'.format(commission)

    # See if the user wants to do another one.
    keep_going = input('Do you want to continue (Enter y for yes): ')
```


Sentinels

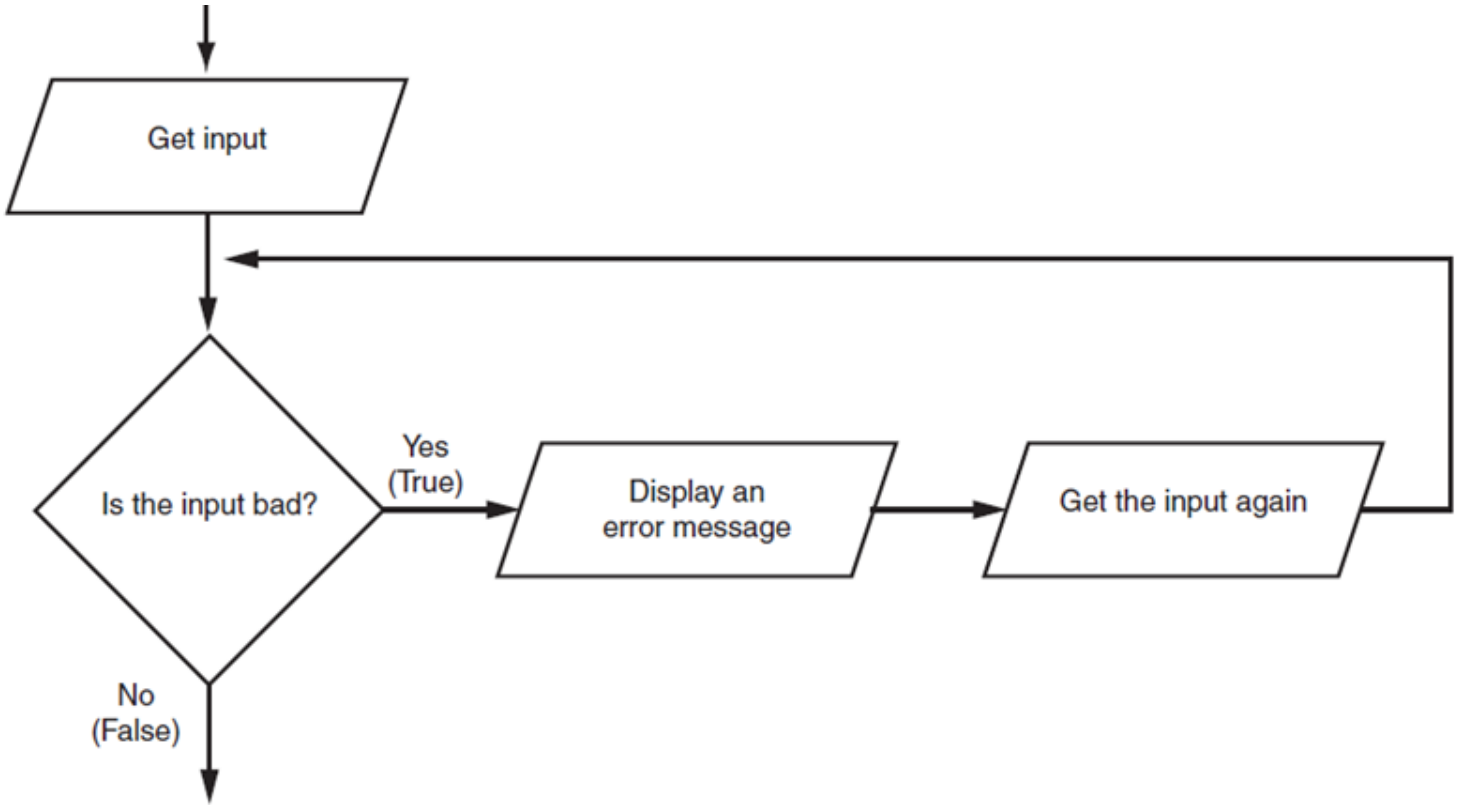
- **Sentinel**: special value used to mark end of a sequence of items or loop.
  - When program reaches a sentinel, it knows that the end of the sequence of items was reached, and the loop terminates.
  - Must be distinctive enough so as not to be mistaken for a regular value in the sequence. Example: We used -99 as our sentinel value to end loop below.

```
user_input = 1
sum = 0
while user_input != -99:
    user_input = int(input("Enter your number or -99 to end."))
    sum = sum + user_input
print("The sum of numbers is:  {}".format(sum))
```

Input Validation Loops

- Computer cannot tell the difference between good data and bad data
  - If user provides bad input, program will produce bad output
  - GIGO: garbage in, garbage out
  - It is important to design program such that bad input is never accepted.
- **Input validation**: inspecting input before it is processed by the program
  - If input is invalid, prompt user to enter correct data
  - Commonly accomplished using a `while` loop which repeats as long as the input is bad.
    - If input is bad, display error message and receive another set of data
    - If input is good, continue to process the input.

Input Validation Loops - 2



Input Validation Loops - 2

```
# This program calculates retail prices.

mark_up = 2.5  # The markup percentage
another = 'y'  # Variable to control the loop.

# Process one or more items.
while another == 'y' or another == 'Y':

   # Get the item's wholesale cost.
    cost = float(input("Enter the item's " + \
                       "cost: "))

# Validate the wholesale cost.
   while wholesale < 0:
      print('ERROR: the cost cannot be negative.')
      cost = float(input('Enter the correct cost:'))

# Calculate the retail price.
   retail = cost * mark_up

   # Display the retail price.
   print('Retail price: $', format(retail, ',.2f'))

   # Do this again?
   another = input('Do you have another item? ' + \
                   '(Enter y for yes): ')
```

Nested Loops

- Nested loop: loop that is contained inside another loop
  - **Example**: analog clock works like a nested loop
    - Hours hand moves once for every twelve movements of the minutes hand: for each iteration of the "hours," do twelve iterations of "minutes"
    - Seconds hand moves 60 times for each movement of the minutes hand: for each iteration of "minutes," do 60 iterations of "seconds"
- Key points about nested loops:
  - Inner loop goes through all of its iterations for each iteration of outer loop
  - Inner loops complete their iterations faster than outer loops
  - Total number of iterations in nested loop:

```
number_iterations_inner  x number_iterations_outer
```

Questions?