Procedural programming in 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

Liaqat Ali, Summer 2018.
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.

Liaqat Ali, Summer 2018.
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.

Liaqat Ali, Summer 2018.
Course Topics

1. General introduction
2. Algorithms, flow charts and pseudocode
3. **Procedural programming in Python**
4. Data types and control structures
5. 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. Operators
   i. Arithemetic Operators (+, -, *, /)
   ii. Comparison operators ( <, >, <=, >=, ==, !=)
   iii. Binary and unary operators
   iv. Logical Operators (and, or, not)

2. Variables / Variable Names

3. Assignment Statements

4. Statement
Review

Can you name or describe the following?

In Python...

1. 5 is ____
2. total is _____
3. + is ____
4. == is a
5. a + b is ____
6. midterm > 95 is ____

1. a value
2. a variable
3. an arithmetic operator
4. a conditional or relational operator
5. an arithmetic expression
6. a Boolean expression

Liaqat Ali, Summer 2018.
In Python:

1. \(x - 5\) is ____
2. \(X \neq 5\) is _____
3. \(\%\) is ____
4. \((a \geq 60)\) is a
5. \((a < 70)\) is ____
6. \((a \geq 60)\) and \((a < 70)\) is ____

1. an arithmetic expression
2. a Boolean expression
3. an arithmetic operator
4. a Boolean expression
5. a Boolean expression
6. a Boolean expression.

• More specifically, a complex Boolean expression.
Binary and Unary Operators

- **Binary operator**: The operator that requires two operands.
  - To add we need two numbers or two operands, so \(+\) is a binary operator. For example, \(10 + 6\). Other examples:
    - All arithmetic operators. We need two operands for \(+\), \(-\), \(*\), \(/\), \(//\), and \(\%\).
    - All the conditional operators (\(<\), \(<=\), \(>\), \(>=\), \(==\), \(!+\)).

- **Unary operator**: The operator that requires only one operand.
  - For example, positive or negative operator: \(-50\) or \(+30\). Here \(-\) and \(+\) are act as unary operators.
Logical Operators
Logical Operators

- The symbols `and`, `or`, and `not` are called **logical operators**.
- We use `and` and `or` logical operators to create **compound** Boolean expressions.
- We use `not` logical operator to reverse the result of its operand. `not(a>70)`
  - By compound we mean to join **two or more** Boolean expressions.
  - For example in a Boolean expression `total >= 85 and total < 90`:
    - the symbol `and` is a **logical operator**,
    - `total >= 85` and `total < 90` are two **Boolean expressions**.
- A compound Boolean expression returns a **True** or **False** result.
Logical Operators: In Compute Grade Example

```python
midterm = 0
final = 0

midterm = input("Enter midterm:"")
final = input("Enter final:"")

total = float(midterm) + float(final)

if total>=95: print("A+")
elif total>=90 and total<95: print("A")
elif total>=85 and total<90: print("A-")
elif total>=80 and total<85: print("B+")
elif total>=75 and total<80: print("B")
elif total>=70 and total<75: print("B-")
elif total>=65 and total<70: print("C+")
elif total>=60 and total<65: print("C")
elif total>=55 and total<60: print("C-")
else: print("F")
```

Logical operators
Logical Operator: \textit{and}

- \textit{and}: The \textit{and} is a \textit{binary} logical operator that connects two Boolean expressions into one compound expression.
  - The result of \textit{and} compound binary expression is \textit{true} when all the sub expressions are \textit{true}.
  - For example, the result of a compound Boolean expression \texttt{marks} \texttt{>=90} \textit{and} \texttt{marks} \texttt{<95} will be true, when the results of:
    - \texttt{marks} \texttt{>=90} Boolean expression is true, and
    - \texttt{marks} \texttt{<95} Boolean expression is true.
Logical Operator: *and* Truth Table

- We can simplify the and results, or decision structure, using a table. We call it a **Truth Table**.
- Truth table for the *and* operator:

<table>
<thead>
<tr>
<th>Expression 1</th>
<th>Expression 2</th>
<th>Expression 1 and expression 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>marks &gt;= 90</td>
<td>marks &lt; 95</td>
<td>marks &gt;= 90 and marks &lt; 95</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
</tbody>
</table>
Logical Operator: **or**

- **or**: The **or** is a *binary* logical operator that connects two Boolean expressions into one compound expression.
  - The result of **or** compound binary expression is **true** when **either** of the sub expressions is **true**.
  - For example, the result of a compound Boolean expression **gpa >= 2.5 or height >= 7** (closing hours) will be true, when either the result of:
    - **gpa > 2.5** Boolean expression is true, or
    - **height > =7** Boolean expression is true.
Logical Operator: *or* Truth Table

- We can simplify the and results, or decision structure, using a table. We call it a **Truth Table**.
- Truth table for the **or** operator:

<table>
<thead>
<tr>
<th>Expression 1</th>
<th>Expression 2</th>
<th>Expression 1 <strong>and</strong> expression 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>gpa &gt;= 2.5</td>
<td>height &gt;= 7</td>
<td>gpa &gt;= 2.5 <strong>or</strong> height &gt;= 7</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
</tbody>
</table>

Liaqat Ali, Summer 2018.
Short-Circuit Evaluation (I would call it a smart evaluation)

- **Short-circuit** is deciding the value of a compound Boolean expression after evaluating only one sub expression.
- Performed by the *and* and *or* operators.
- For *and* operator: If left operand is *false*, compound expression is false. Otherwise, evaluate right operand.
- For *or* operator: If left operand is *true*, compound expression is true. Otherwise, evaluate right operand.
Logical Operator: *not* its and Truth Table

- **not**: It reverses the logical value of the Boolean expression.
  - It turns a **true** into **false**, and a **false** into a **true**.
  - It takes only one operand. So **or** is a unary operator.
  - It is recommended to place parentheses around a Boolean expression to clarify to what you are applying the not operator.

  - **not**(*gpa >= 2.5*)

<table>
<thead>
<tr>
<th>Expression</th>
<th>Not(expression)</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
</tr>
</tbody>
</table>
Variables / Assignment Statement
Variables

- **Variable** is a name that represents a value stored in the computer memory (RAM).
  - We store data in computer memory via variable names.
  - We access and manipulate data in memory via variables.
  - Examples: `marks`, `midterm`, `sum`, or `total`.

- **Assignment Statement** assigns a value to a variable. (Variable that receive value should be on left.)
  - midterm = 50
  - sum = 100
  - name = “Joe”

- You can only use a variable if a value is assigned to it.
Variable Naming Rules

• Rules for naming variables in Python:
  □ Variable name cannot be a Python key word, like `input`, `print`, `if`
  □ Variable name cannot contain spaces.
    • `total marks` is invalid, but `totalmarks` is valid.
  □ First character must be a letter or an underscore.
    • `7stars` is invalid, but `_7stars` is valid.
  □ After first character may use letters, digits, or underscores.
    • `a7_b3` is valid.
  □ Variable names are case sensitive.
    • `Abc` is different from `abc`.

• Variable name should reflect its use.
  □ `xyz` is not good but `midterm_marks` is better.
Statement

• **Statement** is a unit of code that has an effect, like creating a variable or displaying a value.

• For example,

  \[
  n = 17 \text{ is a statement.}
  \]
  \[
  \text{print}(n) = 17 \text{ is a statement.}
  \]
  \[
  \text{marks} = \text{input}("\text{Enter marks: }\") \text{ is a statement.}
  \]
Questions?