Flow of Control

- Flow of control = order in which instructions are executed
- By default, instructions are executed in sequential order.

Instructions:

\[
\begin{align*}
\text{sum} &= 0 \\
\text{num1} &= 5 \\
\text{num2} &= 10 \\
\text{sum} &= \text{num1} + \text{num2}
\end{align*}
\]

Flowchart:

- When we call a function like `input()`, we transfer control to the function until it completes.
Altering the Flow of Control

• To solve many types of problems, we need to be able to modify the order in which instructions are executed.

• We've already seen one example of this: for loops allow us to repeat a set of statements some number of times.

```python
# add five integers
sum = 0
for i in range(5):
    num = eval(input("Enter a number: "))
    sum = sum + num
print("the sum is ", sum)
```

• Now we'll see how to allow a program to decide whether to do something, based on some condition.

Example of Making Decisions

• The ability to make decisions allows us to handle invalid inputs.

• Let's say the user gives us a number, and we want to compute its square root.
  • invalid inputs include:
    • negative numbers
    • non-numeric values (we'll ignore these for now)

• Here's one way to handle negative numbers:

```python
import math
num = eval(input("Enter a number >= 0: "))
if num < 0:
    print("using the absolute value of the number")
    num = num * -1
root = math.sqrt(num)
print("the square root of", num, "is", root)
```
Simple Decisions: if Statements

• A simple if statement has the form

```
if <condition>:
<true block>
```

where <condition> is an expression that is true or false
<true block> is one or more statements

• If the condition is true, the statement(s) in the true block are executed.

• If the condition is false, the statement(s) in the true block are skipped.

Flowchart for a Simple if Statement

- condition
  - true
  - false
- true block
- next statement
Flowchart for Our Example

```
num = eval(input("Enter..."))

if num < 0:
    print("using the absolute ...")
    num = num * -1

    root = math.sqrt(num)

    print("the square root of...")
```

Expressing Simple Conditions

As in SQL, Python provides a set of operators called *relational operators* for expressing simple conditions:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Name</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;</td>
<td>less than</td>
<td>5 &lt; 10, num &lt; 0</td>
</tr>
<tr>
<td>&gt;</td>
<td>greater than</td>
<td>40 &gt; 60 (which is false!)</td>
</tr>
<tr>
<td>&lt;=</td>
<td>less than or equal to</td>
<td>average &lt;= 85.8</td>
</tr>
<tr>
<td>&gt;=</td>
<td>greater than or equal to</td>
<td>name &gt;= &quot;Jones&quot;</td>
</tr>
<tr>
<td>==</td>
<td>equal to</td>
<td>sum == 10, firstChar == &quot;P&quot;</td>
</tr>
<tr>
<td>!=</td>
<td>not equal to</td>
<td>age != myAge</td>
</tr>
</tbody>
</table>
**bool** Data Type

- A condition has one of two values: **True** or **False**.
  ```python
  >>> 10 != 20
  True
  >>> "Jones" < "Baker"
  False
  ```

- In Python, these two values are represented using a special data type called **bool**.
  ```python
  >>> type(10 != 20)
  <class 'bool'>
  ```

- This type is named after the 19th-century mathematician George Boole, who developed the system of logic called **boolean algebra**.

- An expression that evaluates to **True** or **False** is known as a **boolean expression**.

---

**Forming More Complex Conditions**

- We often need to make a decision based on more than one condition — or based on the opposite of a condition.
  - examples in pseudocode:
    ```
    if the number is even AND it is greater than 100…
    if it is NOT the case that your grade is > 80…
    ```

- Like SQL, Python provides **logical operators** for this purpose:

<table>
<thead>
<tr>
<th>Name</th>
<th>Example and Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>and</td>
<td><code>(age &gt;= 18 and age &lt;= 35)</code></td>
</tr>
<tr>
<td></td>
<td>true if both conditions are true, and false otherwise</td>
</tr>
<tr>
<td>or</td>
<td><code>(age &lt; 3 or age &gt; 65)</code></td>
</tr>
<tr>
<td></td>
<td>true if one or both of the conditions are true; false if both conditions are false</td>
</tr>
<tr>
<td>not</td>
<td><code>(not grade &gt; 80)</code></td>
</tr>
<tr>
<td></td>
<td>true if the condition is false, and false if it is true</td>
</tr>
</tbody>
</table>
Practice with Boolean Expressions

• Let's say that we wanted to express the following English condition in Python:
  
  "num is not equal to either 0 or 1"

• Which of the following boolean expression(s) would work?
  a) \( \text{num} \neq 0 \text{ or } 1 \)
  b) \( \text{num} \neq 0 \text{ or } \text{num} \neq 1 \)
  c) \( \neg (\text{num} = 0 \text{ or } \text{num} = 1) \)

• Is there a different boolean expression that would work here?

Warning: Python May Not Catch Your Mistakes!

• It turns out that
  
  \( \text{num} \neq 0 \text{ or } 1 \)

  is actually a valid boolean expression in Python!

• That's because Python allows you to use values from other types to represent True and False.

• For numeric types:
  • 0 is equivalent to False
  • any other number is equivalent to True

• Therefore:
  
  \( \text{num} \neq 0 \text{ or } 1 \)

  is equivalent to

  \( \text{num} \neq 0 \text{ or } \text{True} \)

  What's the value of this expression for different values of num?
Sample Problem: Number Analyzer

• Read in an integer from the user, and report whether it is even or odd.
  • what operator can we use to determine if a number is even?

• One possible approach (fill in the conditions):
  
  ```python
  num = eval(input("enter an integer: "))
  # if num is even, say so.
  if _________________:
    print(num, "is even.")
  # if num is odd, say so.
  if _________________:
    print(num, "is odd.")
  ```

• The second condition is redundant. Why?

Sample Problem: Number Analyzer (cont.)

• A better approach (fill in the same first condition as before):
  ```python
  num = eval(input("enter an integer: "))
  if num % 2 == 0:
    print(num, "is even.")
  else:
    print(num, "is odd.")
  ```
Two-Way Decisions: `if-else` Statements

- In general, an `if-else` statement has the form:

  ```
  if <condition>:
    <true block>
  else:
    <false block>
  ```

- If the condition is true:
  - the statement(s) in the true block are executed
  - the statement(s) in the false block are skipped

- If the condition is false:
  - the statement(s) in the false block are executed
  - the statement(s) in the true block are skipped

Flowchart for an `if-else` Statement:

```
condition

true block

false block

next statement
```
Flowchart for Our Example

```
true
num % 2 == 0
false

print(num, "is even.")

... 

print(num, "is odd.")
```

Extended Number Analyzer

- Let's add code to check if the value entered is really an integer.
  - what built-in function can we use?
  - if it isn't an int, we'll print an error message
  - if it is an int, we'll say whether it's even or odd as before

- We could do something like this:
  ```python
  num = eval(input("enter an integer: "))
  if type(num) != int:
    print(num, "is not an integer.")
  else:
    if num % 2 == 0:
      print(num, "is even.")
    else:
      print(num, "is odd.")
  ```

- We've nested our previous if-else statement in the false block of another if-else statement!
Extended Number Analyzer (cont.)

```python
num = eval(input("enter an integer: "))
if type(num) != int:
    print(num, "is not an integer."
else:
    if num % 2 == 0:
        print(num, "is even."
    else:
        print(num, "is odd."
```

• Instead of using nesting, Python allows us to combine an else followed immediately by an if as follows:

```python
num = eval(input("enter an integer: "))
if type(num) != int:
    print(num, "is not an integer."):
elif num % 2 == 0:
    print(num, "is even."
else:
    print(num, "is odd."
```
Multi-Way Decisions: if-elif-else Statements

• In general, an if-elif-else statement has the form

```python
if <condition1>:
    <true block for condition1>
elif <condition2>:
    <true block for condition2>
elif <condition3>:
    <true block for condition3>
...
else:
    <false block for all conditions>
```

• The conditions are evaluated in order. The true block of the first true condition is executed.

• If none of the conditions are true, the false block is executed.

Flowchart for an if-elif-else Statement
Example Problem: Ticket Sales

- Rules for ticket sales:
  - persons younger than 13 are not allowed to buy a ticket
  - persons 13-24 or 65 and older pay a discounted price of $35
  - everyone else pays the regular price of $50

Example Problem: Ticket Sales (cont.)

```python
age = eval(input("Enter your age: "))
if age < 13:
    print("You're too young to buy a ticket.")
else:
    print("You may buy a ticket.")
    if ______________________:
        price = 35
    else:
        price = 50
    print("The price is", price, "dollars.")
```

- What condition should be used to fill in the blank?
- Why can't we use an if-elif-else statement here?
Example Problem: Ticket Sales (cont.)

- Here's another version of the ticket-sales program that nests an if-else statement in the true block:

```python
age = eval(input("Enter your age: "))
if age >= 13:
    print("You may buy a ticket.")
if age <= 24 or age >= 65:
    price = 35
else:
    price = 50
print("The price is", price, "dollars.")
else:
    print("You're too young to buy a ticket.")
```

Warning: Indentation Matters!

- Consider the following two code fragments:

```python
if age > 24:
    if age < 65:
        price = 50
else:
    price = 35

if age > 24:
    if age < 65:
        price = 50
else:
    price = 35
```

- When age == 18, what will the value of price be:
  - when we use the first version?
  - when we use the second version?
Warning: Indentation Matters! (cont.)

- An else or elif clause must be indented the same number of spaces as the corresponding if clause.
- If you’re not careful, you can get an indentation error.
  - example:
    ```python
    if age > 24:
        price = 50
    else:
        price = 35
    ```
    which gives the following error message:
    ```
    SyntaxError: unindent does not match any outer indentation level
    ```
    - this error occurs because the else is indented 2 spaces more than the corresponding if

Warning: Don't Forget the Colon!

- There must be a colon at the end of each if, elif, and else clause.
- If you forget to include it, you'll get an error.
  ```python
  if age > 24:
      price = 50
  else   # this is missing a colon
  SyntaxError: invalid syntax
  ```
  - When you get a syntax error, the problematic code is highlighted.
    - in the code above, the space after the else is highlighted, indicating that something is missing!
Practice: Expanding Our Ticket-Sales Program

• Different prices for balcony seats and orchestra seats

• Expanded rules for ticket sales:
  • persons younger than 13 are not allowed to buy a ticket
  • persons 13-24 or 65 and older receive discounted prices:
    • $20 for balcony seats
    • $35 for orchestra seats
  • everyone else pays the regular prices:
    • $30 for balcony seats
    • $50 for orchestra seats

Expanding Our Ticket-Sales Program (cont.)

```python
age = eval(input("Enter your age: "))
if age >= 13:
    print("You may buy a ticket.")
    type = eval(input("(1) orchestra | (2) balcony? "))
    # what should go here? (assume input is 1 or 2)
    print("The price is", price, "dollars.")
else:
    print("You're too young to buy a ticket.")
```
Avoid Overly Complicated Code

• The following also involves decisions based on a person’s age:

```python
age = eval(input("Enter your age: "))
if age < 13:
    print('You are a child.')
elif age >= 13 and age < 20:
    print('You are a teenager.')
elif age >= 20 and age < 30:
    print('You are in your twenties.')
elif age >= 30 and age < 40:
    print('You are in your thirties.')
else:
    print('You are really old.')
```

• How could it be simplified?

Practice: A Simple Calculator

• Ask the user to enter:
  • two numbers
  • the operation they want to perform (add, subtract, multiply)
and compute and print the result.

• We can use a single `input` statement to get both numbers!

```python
>>> a, b = eval(input("Enter two numbers: "))
Enter two numbers: 5, 10
>>> a
5
>>> b
10
```

• This makes use of what is known as `simultaneous assignment`.
  • general form:
    
    `<var1>, <var2>, <var3>, … = <expr1>, <expr2>, <expr3>, …`

Practice: A Simple Calculator

```python
a, b = eval(input("Enter two numbers: "))

print("Choose an operation by number:")
print("  (1) addition")
print("  (2) subtraction")
print("  (3) multiplication")
choice = eval(input())

# what should go here?

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