Computer Programming Fundamentals

9/5/2017
Announcements

- Textbook: Did you get a copy?
- Tutoring schedule set: tutoring has already started
- First lab tomorrow: bring your laptop (place may change – so read Announcements before you come)

- Today: Computer programming fundamentals
- Reading: Chapter 2 of Conery

**Acknowledgement**: Some of the lecture slides are based on CSE 101 lecture notes by Prof. Kevin McDonald at SBU and the textbook by John Conery.

- Break around 11:15am
What is Python?

- Python is a computer programming language
  - It has a relatively simple **syntax**, or set of rules that programmers must follow when writing programs in the language
- With Python you can write very simple programs that do basic calculations or very complicated ones
  - You can even write basic games!
  - Python is popular with scientists because they can do complex data analysis by writing short programs
- Python can be installed on a wide variety of computer types and operating systems
- See the course website for installation instructions (Python v. 3.6.2)
  - In console, you will use `python3` to start a session
What is a computer program?

- A computer program is basically a sequence of instructions that the computer executes to solve some well-defined problem.
- The instructions or steps that the programmer writes constitute the **source code** or simply **code** of the program.
- In Python, many of these instructions look like regular, everyday English with some extra punctuation thrown in.
- There are two basic ways to give commands written in Python to the computer:
  1. We can type individual instructions via the **shell**, an interactive program that executes the commands.
  2. We can write a complete, stand-alone **application** that we can run over and over.
Python console / interactive shell

- The **console** or interactive shell is basically a window where you type a single command or short set of commands to the computer, and the computer tries to execute them.

- As we type Python instructions into the console and hit the Enter key, the Python **interpreter** reads the instructions and converts them into a form the computer’s hardware understands.

- The language that the hardware understands is called **machine language**.

- No matter what programming language you use, at some point your code has to be translated into machine code for the computer to execute it.
The PyCharm IDE

- Rather than typing console commands, in this course we will use an integrated development environment (IDE) called PyCharm.
- PyCharm is an industry-grade piece of software used by professional software developers, but is still easy enough for novice programmers to use.
- Then go to www.jetbrains.com/pycharm to download and install the free Community Edition of PyCharm.
- More details about the software installation are on the course website and in the first lab assignment, but the next slide has some of the basics.
PyCharm basics

- To create and run a stand-alone Python program:
  1. Start PyCharm and press the “Create New Project” button.
  2. Pick a “Location” for your work on your hard drive and give a name to your project (e.g., “CSE 101”).
  3. Select File Menu > New > Python File and enter the name of your file.
  4. Write your program and save your work.
  5. After saving your work, go to Run Menu > Run.
  6. Select the name of your program to run it.
- The next time you want to run the program, just hit the green triangle in the lower-left corner of the screen.
- Or, right-click the name of the file and choose Run.
Commands and expressions

- 'Hello, world!' is an expression
  - It has a value
  - In this case, it’s a **string** (a sequence of characters)
- Numbers are also expressions
  - 5 is an **integer** expression (recall that an integer is zero, or a positive or negative whole number that has no fractional part)
- 12.36 is a **floating-point** expression
  - **floating-point** is a format that computers use to represent **real** numbers (recall that a real number is zero, or a positive or negative number that might have a fractional part)
Commands and expressions

- An expression consists of operators and operands
  - $2 \times 9$ is an expression and represents a multiplication
- Python also has **Boolean** expressions, which are expressions that can be **True** or **False**
- Boolean expressions allow us to write programs that change their behavior from one run to the next. More on this later.
- So we have at least three kinds of data in Python programming: strings, numbers, and true/false values
- You will find that in computer programming we deal with a wide variety of data because there is a wide variety of problems that computers can help us solve
Arithmetic in Python

- Some of the simplest statements in Python involve arithmetic expressions, which contain numbers (operands) and mathematical operators.
- Arithmetic in Python follows the same PEMDAS rule you learned in elementary school:
  1. First, evaluate all expressions in parentheses
  2. Then, perform exponentiations
  3. Next, perform multiplications and divisions in left-to-right order
  4. Finally, perform additions and subtractions in left-to-right order
Arithmetic in Python

- The symbols used for operators are commonly used in other languages and applications (e.g., spreadsheets)
  - add: +
  - subtract: –
  - multiplication: *
  - division for real numbers: /
  - division for integers: // (when we don't need the remainder)
  - remainder: % (gives the remainder of an integer division)
  - exponentiation: **
Examples of arithmetic in Python

- $11 + 5 \rightarrow 16$
- $11 - 5 \rightarrow 6$
- $11 \times 5 \rightarrow 55$
- $11 / 5 \rightarrow 2.2$
- $11 // 5 \rightarrow 2$

  - This example shows **integer division**. Any remainder is discarded.

- $11 \% 5 \rightarrow 1$

  - The computer divides 11 by 5 and returns the remainder (which is 1) instead of the quotient (which is 2).
  - Use the remainder operator only with integers.
Arithmetic in Python

- The ** operator lets us do exponentiation or raise a number to a power
- For example, $2 \times 5$ would be 32 because $2^5 = 32$
- Perhaps you are aware that raising a number to the power $\frac{1}{2}$ is the same as taking a square root
- So $16 \times 0.5$ would be the same as $\sqrt{16}$, which is 4
Arithmetic in Python

- The constant \( \pi \) is built into Python
- First the programmer must make it available by importing the `math` module:
  - `import math`
- Then the expression `math.pi` can be used in expressions
  - `math.pi * 2 + 1`
- A Python module is a file consisting of Python codes that are all related somehow
- For example, the `math` module contains code pertaining to mathematical functions and constants
Announcements (Sept 7)

- Get help during Prof. Lee’s and TAs’ office hours
- Text editor: emacs? Editor in PyCharm? Both?

- Today: Computer programming fundamentals (cont’d)
- Reading: Chapter 2 of Conery

- Break around 11:15am
Variables

- A variable in computer programming is similar to the concept of a variable in mathematics: it is a name for some value or quantity of interest in a given problem.
- For example, in a program we might use variables to store a person’s age, GPA, name, or virtually any other kind of information.
- The value is temporarily stored in the main memory (RAM) of the computer while the program is running.
- A variable is a kind of identifier because it identifies (names) something in source code.
- It is important to choose identifiers (e.g., variable names) that are informative and helpful.
Variables

- For example, `first_name` would be a good variable to store a person’s first name, whereas `fn` would not be as good because it’s less informative.
- Note how the underscore is used to separate words that define the identifier:
  - Spaces are not allowed in variable names.
- A Python variable name may contain lowercase letters, uppercase letters, digits, and underscores:
  - But, the first character must be a letter or underscore.
Variables

- Lowercase and uppercase letters are treated as completely different characters
  - Because of this we say that Python is a **case-sensitive language**
  - `First_Name`, `first_name` and `FIRST_NAME` would all be treated as different identifiers
- There are a number of **keywords** built into the Python language that have pre-defined meanings
  - We aren't allowed to use these predefined keywords as variables
  - I’ll point them out as we go
Assignment statements

- When we want to give a value to a variable we write an **assignment statement**
- An assignment statement consists of a variable name, the equals sign, and a value or expression
- Examples:
  - `count = 3` (“count is 3” or “count becomes 3”)
  - `total = 3.85 + 12.9`
  - `firstName = ‘Susan’`
- These examples show three different data types: an integer, a real number, and a string
Assignment statements

- After assigning a value to a variable, you can change the value of the variable with another assignment statement:
  
  ```
  total = 5 + 8 + 3
  ...
  total = 17 + 6
  ...
  ```

- Variables can also appear on the right-hand side (RHS) of an assignment statement:
  
  ```
  next_year = this_year + 1
  total_bill = subtotal + tax + tip
  ```
Example: Area calculation

- Suppose we wanted to compute the area of a square countertop with one corner cut off, as shown in the image below.
Example: Area calculation

- Assume that the triangular cut-out begins halfway along each edge.
- If we needed to perform the computation only once, say for a 100 cm-long countertop, we might write a statement like this:
  \[ \text{area} = 100^{**2} \text{ } - \text{ } 50*50/2 \]
- Note that this code has a few issues with it:
  - It’s just a formula of sorts with no explanation of what the numbers mean.
  - The code works only for countertops exactly 100 cm long. What if we had countertops of other sizes?
Example: Area calculation

- Let’s address the first issue: lack of clarity
  
  \[
  \text{area} = \text{area of square} - \text{area of triangle} \\
  \text{area of triangle is } 1/2 \text{ base*height} \\
  \text{area} = 100^{\text{2}} - 50\times50/2
  \]

- The lines beginning with the \# symbol are called **comments**
  
  - Comments are notes that the programmer writes to explain what the program does
  - Comments do not affect the input or output of the program or anything about how it runs
Example: Area calculation

- Now let’s address the other issue: lack of generality
  
  ```
  side = 100
  square = side**2
  triangle = (side/2)**2 / 2
  area = square - triangle
  ```

- To compute the area for a countertop of a different size, all we need to change is the first line: `side = 100`

- This code is also more readable; comments aren’t needed
  - This is an example of **self-documenting code**

- The spacing in between variables, numbers, and operator is optional, but is included here to make the formulas easier to read
Aside: `input` statements

- To improve this code even further, let’s make it interactive so that the user can provide the value for `side`
- To that end we will write an **input statement**
- An input statement reads a string from the keyboard
- As part of an input statement, the programmer must give a **prompt** message that tells the user what the user is actually supposed to enter
- Example: `name = input('What is your name? ')`
- The person’s name will be **assigned to** the `name` variable
  - You could also say that we are **saving** the person’s name in the `name` variable
Example: Area calculation

- In our case, the user should enter a number, not a string
- If we want the user to enter an integer, we should type: `side = int(input('Enter side length: '))`
- But if we want a floating-point number, we should type: `side = float(input('Enter side length: '))`
- Which one we choose – `int` vs. `float` – depends on the application
- For this program we will read in a float so that we could enter a fraction of a centimeter if we wanted
- The last piece of the puzzle is how to display the final result on the computer screen. Let’s look at that.
Aside: `print statements`

- `print` is a Python command
- It tells Python to do something, namely, to display some text on the screen
  - All Python commands are lowercase
- The syntax to print a basic message is just this:
  ```python
  print('Hello, world!')
  ```
- Any text printed after this will appear on a new line
- If you want the next output to be on the same line, do this instead:
  ```python
  print('Hello, world!', end='')  # for python3
  ```
- This means *print this message, but do not automatically go to the next line*
Aside: print statements

- To print a number we need to convert it first into a string, like so:
  ```python
  print('The area is ' + str(area))
  ```
  - The assumption here is that `area` is a variable that contains the value we want to print
- When used in this fashion, the + symbol performs **string concatenation**, which is just a fancy way of saying we are joining two strings together into one
- We can now complete our area calculation program
Example: countertop.py

# This program prints the area of a 
# countertop formed by cutting the 
# corner off a square piece of material 
# (e.g., granite).

side = float(input('Enter side length: ')) 
square = side**2 
triangle = (side/2)**2 / 2 
area = square - triangle 
print('The area is ' + str(area))
Example: coins.py

- Let’s see an example of the remainder operator in integer division
- Given a total number of cents, we want the computer to tell us how many dimes, nickels, and pennies are needed to make that change while minimizing the number of coins
- We’ll also make good use of variables
- We will use the `str` command to print variables containing numbers to the screen
  - Recall that `str` converts a number to a string so that it can be concatenated with other strings
Example: coins.py

cents = int(input("Enter the number of cents: "))

dimes = cents // 10
cents = cents % 10
nickels = cents // 5
cents = cents % 5
pennies = cents

print("That number of cents is equal to "+str(dimes) + " dimes, " + str(nickels) + " nickels and " + str(pennies) + " pennies.")
Escape sequences

- Escape sequences in programming languages like Python allow you to print characters (symbols) on the screen that let you do some special things with print statements.
- In Python, some of the escape sequences are:
  - `\t`: shifts the text to the right by one tab stop
  - `\n`: prints a newline
  - `\"`: prints a double quotation mark
  - `'`: prints a single quotation mark
- A lone backslash character is called the **line-continuation character** (it’s not really an escape sequence, though):
  - This symbol is a signal to the Python interpreter that the statement we are writing spans two or more lines of a file.
Example: limerick.py

Source code:

```python
print('There was an old man with a beard

Who said, "It's just how I feared!"

Two owls and a hen

Four larks and a wren

Have all built their nests in my beard.
')
```

Output:

There was an old man with a beard
Who said, "It's just how I feared!"
  Two owls and a hen
  Four larks and a wren
Have all built their nests in my beard.
Functions

• Earlier we saw Python has a math module
  • The library has numbers (e, π, etc.)
  • It also has a variety of useful mathematical functions
• In *programming*, a **function** is a name given to a set of statements that perform a well-defined task
• For example, the **input** function performs a task (getting user input) and also returns (gives us) the value entered by the user
• **print**, **int**, **float**, and **str** are also functions
• In the next example, we will see a new function, called **format**, that will let us format numerical output in a desired way
Example: BMI calculator

- As we have seen, once numbers are stored in variables, we can perform calculations with them.
- The Body Mass Index (BMI) is a metric used to gauge a person’s general health.
- Given a person’s weight in pounds and total height in inches, a person’s BMI is calculated as $BMI=\frac{weight}{703/height^2}$.
- A BMI in the range 18.5 – 24.9 is considered “healthy”.
- We will write a program that calculates and prints a person’s BMI based on entered numbers.
- The result will be printed to 15 digits of accuracy, which is more digits than necessary.
Example: BMI calculator

- To print a number to a designed number of digits we can use the `format` function

- Suppose we have a variable `total_due` that we want to print to two decimal places. Here is how you would do it:
  ```python
  print("Total due: $" + "{:2f}".format(total_due))
  ```

- If we wanted four digits, we would write `{:.4f}` instead

- Note that when using the `format` method, you do not also use `str` to print a number

- In the code for this program, you will see two print statements: one giving the BMI to the full accuracy, and a second that rounds the result to three decimal places
Example: bmi_v1.py

```python
weight = float(input('Enter weight in pounds: '))
feet = float(input('Enter feet portion of height: '))
inches = float(input('Enter inches portion of height: '))

total_inches = feet * 12 + inches

bmi = (weight * 703) / total_inches ** 2

print('Your BMI is ' + str(bmi))
print('Your BMI is ' + '{:.3f}'.format(bmi))
```

- The blank lines you see here were inserted to make the code more readable. They do not affect program execution in any way.
Announcements (Sept 12)

- Problem Set 1 (PS 1) is ready and up on course web
  - Two due dates
- Using Python Console within PyCharm
  - Try Tools -> Python Console
- Exam dates are set (tentative)
- Get help during Prof. Lee’s and TAs’ office hours

- Today: Computer programming fundamentals (cont’d)
- Reading: Chapter 2 of Conery

- Break around 11:15am
More on `format` function

```python
x = 12345.6789
y = 765.4321
print('x={0:10.3f}, y={1:8.2f}'.format(x, y))
Output:
x= 12345.678, y=  765.43
```

From: https://www.python-course.eu/python3_formatted_output.php

The following diagram with an example usage depicts how the string method “format” works for positional parameters:
Other functions in Python

- Some examples:
  ```python
type(45)
int(34.56)
float(45)
str(3421)
len('apple')
round(2.32)
abs(-45)
pow(2, 3)  # cf. 2**3
help(pow)
...
```

```python
import math
math
math.log(10)
math.log10(10)
math.log10(1e6)
radians = 0.7
math.sin(radians)
math.sqrt(3)
...
```

```python
import random
random.random()
random.randint(0, 100)
```

Try these on a Python Console or as part of a program
Function composition

- Can compose functions as we do in math, e.g., $f(g(x, y))$

```python
import math
radians = 0.7
math.radians(math.degrees(radians))

radians = 0.3
math.acos(math.cos(radians))

pow(abs(-3), round(5.6))
```
Defining new functions

- Functions in program have many benefits, including:
  - They make code easier to read and understand because we don’t need to know the details of how or why a function works
  - They allow code to be used more than once (code re-use)

- To define a new function in Python we use a `def` statement
  - For example, suppose we want to write a function that computes a person's Body Mass Index
  - We could then call this function as many times as we want
  - The alternative would be to copy and paste the code multiple times
  - First rule of programming: don’t repeat yourself!
Creating new functions

- What we know from **math**: a 2 step process
  1. Define a function, once
     \[ f(x, y) = x * y + 1 \]
  2. Apply/Use/Invoke/Call the function, as many times as you wish
     \[ f(2, 3) = 2 * 3 + 1 = 7 \]

- We do the same in **programming**: again a 2 step process
  1. Define a function, once
     ```python
     def f(x, y):
         return x * y + 1
     ```
  2. Apply/Use/Invoke/Call the function, as many times as you wish
     \[ f(2, 3) \]
Mechanics of defining/calling a function

```python
def foo(x, y):
    product = x * y
    return product + 1

foo(2, 3)
```

- **Function name**
- **Parameters**
- **Function body**
- **Required elements**
- **Indention required** for every line of the Function body

- () after a function name means a **function call**, in this case with two arguments
Parameters and arguments

- Function can have **zero or more** parameters
  - Function may be defined with **formal parameters**
  - Then called with **actual arguments**
  - How many? As many as your function needs!

- Example:

```python
def multAdd(a, b, c):
    return a * b + c

print(multAdd(1, 2, 3))
print(multAdd(2.1, 3.4, 4.3))
print(multAdd(abs(pow(2, 3)), 3.2 + 2.3, 45.34))
```
Program: flow of execution

```python
def message():
    print(1)
message1()
print(2)

def message1():
    print('a')
message2()
print('b')
def message2():
    print('middle')
message()
```

Output:

```
1
a
middle
b
2
```

Note: three functions and a call to message on the left is a *program*!
Void functions vs. fruitful functions

- **announce** below is an example of a *void function*
  - It does **not return** any useful value when it is called; it only prints a value
- **square** is an example of a *fruitful function*
  - It **returns** a value when it is called

```python
// void function
def announce(msg):
    print(msg)
announce('hello!')

// fruitful function
def square(n):
    return n * n
print(square(3))
```

See what gets printed by the print statement in each case!
You shouldn’t really expect a void function to return any useful value and `announce('hello')` returns `None` to indicate that.
So if you `print(announce('hello!'))` it will print `None`. 
Example: bmi_v2.py

# Function definition
def bmi(w, h):
    return (w * 703) / (h ** 2)

# main is to use the function defined above.
def main():
    weight = float(input('Enter weight in pounds: '))
    feet = float(input('Enter feet portion of height: '))
    inches = float(input('Enter inches portion of height: '))

    total_inches = feet * 12 + inches
    my_bmi = bmi(weight, total_inches)
    print('Your BMI is ' + '{:.3f}'.format(my_bmi))

# This sets up a call to the function main.
main()
Why functions? Abstraction

- One of the most important concepts in computer science is abstraction
  - Give a name to a group of statements and use it, e.g., `bmi(...)`
- From the outside we forget about the details: all we care about is the fact that we can call this function and it will do a computation
- Functions thereby allow us to solve a complex problem by subdividing it into smaller, more manageable sub-problems
  - This process is called problem decomposition
- Often programmers use functions to engage in top-down software design, which means that they design the software as a series of steps, each of which corresponds to one or more functions
Example: bmi_v3.py

- Next, we will look at an alternative way of implementing the \texttt{bmi} function that illustrates proper indentation and relies on two local variables, \texttt{numerator} and \texttt{denominator}

- A \textit{local variable} is a variable accessible only inside the function where it is created
```python
Example: bmi_v3.py

    def bmi(w, h):
        numerator = w * 703
        denominator = h ** 2
        return numerator / denominator

    def main():
        weight = float(input('Enter weight in pounds: '))
        feet = float(input('Enter feet portion of height: '))
        inches = float(input('Enter inches portion of height: '))

        total_inches = feet * 12 + inches
        my_bmi = bmi(weight, total_inches)
        print('Your BMI is ' + '{:.3f}'.format(my_bmi))

    main()
```
Example: Distance calculator

- Suppose we are given a distance traveled in miles, yards, and feet, such as 3 miles, 68 yards, 16 feet
- We would like to convert this distance into total inches traveled and print the result
- To that end we need to perform some unit conversions
- Recall the following equivalences:
  - 1 foot = 12 inches
  - 1 yard = 3 feet
  - 1 mile = 5,280 feet
- Finally, to print a comma every three digits we can use the formatting string '{:,}' when printing an integer
def distance(m, y, f):
    return (m * 5280 * 12) + (y * 3 * 12) + (f * 12)

def main():
    miles = int(input('Enter the number of miles: '))
    yards = int(input('Enter the number of yards: '))
    feet = int(input('Enter the number of feet: '))

    inches = distance(miles, yards, feet)

    print('Distance in inches: ' + '{:,}'.format(inches))

main()
Example: Mortgage calculator

- The monthly payment on a fixed-rate mortgage can be calculated using this formula: \( pmt = \frac{Pr}{1-1/(1+r)^n} \)
  where \( P \) is the principal (the amount we borrowed), \( r \) is the monthly interest rate as a decimal (i.e., the annual interest rate as a decimal divided by 12), and \( n \) is the number of months the loan will last.

- To include a comma every three digits, write your format string as `\{:, .2f\}` for floats.

- Also, you can save a format string in a variable if you want to format a bunch of numbers in the same way:
  \( \text{fmt} = \{:, .2f\} \)

- Let’s write a function to compute \( pmt \)
def monthly_payment(borrow_amt, monthly_rate, num_months):
    return (borrow_amt * monthly_rate) /
            (1 - 1 / (1 + monthly_rate) ** num_months)

def main():
    principal = float(input('Enter principal: '))
    annual_rate = float(input('Enter annual interest rate as a percentage: '))
    years = int(input('Enter term of mortgage in years: '))

    payment = monthly_payment(principal, annual_rate / 12 / 100, years * 12)
    totalPaid = payment * years * 12
    totalInterest = totalPaid - principal

    fmt = '{:.2f}'  # formatter string
    print('Principal: $' + fmt.format(principal))
    print('Annual interest rate: ' + fmt.format(annual_rate) + '%%')
    print('Term of loan in years: ' + str(years))
    print('Monthly payment: $' + fmt.format(payment))
    print('Total money paid back: $' + fmt.format(totalPaid))
    print('Total interest paid: $' + fmt.format(totalInterest))

main()
Conditional execution

- Often an algorithm needs to make a decision
- The steps which are executed next depend on the outcome of the decision
- Example: a person’s income range determines the income taxation rate
  - If the income is above a certain minimum, use one tax rate; otherwise, use a lower rate
- In Python, an **if-statement** allows us to test conditions and execute different steps depending on the outcome
Example: Tuition calculator

- Suppose part-time students (< 12 credits) at a fictional college pay $600 per credit and full-time students pay $5,000 per semester.

- Let’s use an if-statement to write a short program that implements this logic.
numCredits = int(input('Enter number of credits: '))

if numCredits < 12:
    cost = numCredits*600
    print('A student taking ' + str(numCredits) + ' credits is part-time and will pay $' + str(cost) + ' in tuition."
else:
    print('A student taking ' + str(numCredits) + ' credits is full-time and will pay $5,000 in tuition."
}
Conditional execution

- if-statements can also appear in functions:
  ```python
def tax_rate(income):
    if income < 10000:
        return 0.0
    else:
        return 5.0
  ```
- Note that the value returned by this function depends on the value passed as an argument to the parameter
- The words `if` and `else` are keywords
- Note the `colon (:)` at the end of the if and else clauses
- Note also how the statements to be executed are indented
Announcements (Sept 14)

- How are you doing with PS 1?
- Get help during Prof. Lee’s and TAs’ office hours

- Lab 2: How did you like?

- Today: Computer programming fundamentals (cont’d)
- Reading: Chapter 2 of Conery

- Break around 11:15am
Multi-way if-statements

- When an algorithm needs to choose among more than two alternatives it can use `elif` clauses
- `elif` is short for “else if”
- This function distinguishes between three tax brackets:

  ```python
  def marginal_tax_rate(income):
      if income < 10000:
          return 0.0
      elif income < 20000:
          return 5.0
      else:
          return 7.0
  ```
- We can use as many `elif` parts as we want or need
Boolean expressions

- The expressions inside `if` and `elif` statements are special kinds of expressions
- The result of these expressions is either `True` or `False`
- An expression that evaluates to `True` or `False` is called a `Boolean expression`
- Boolean expressions often involve relational operators:
  - equal to / not equal to
  - greater than / greater than or equal to
  - less than / less than or equal to
Boolean expressions

- The notation `>=` means “greater than or equal to” and is one of six relational operators supported by Python:

<table>
<thead>
<tr>
<th>Mathematica Operator</th>
<th>Python Equivalent</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>`=</td>
<td><code>==</code></td>
<td>is equal to</td>
</tr>
<tr>
<td>`≠</td>
<td><code>!=</code></td>
<td>is not equal to</td>
</tr>
<tr>
<td><code>&gt;</code></td>
<td><code>&gt;</code></td>
<td>is greater than</td>
</tr>
<tr>
<td><code>≥</code></td>
<td><code>&gt;=</code></td>
<td>is greater than or equal to</td>
</tr>
<tr>
<td><code>&lt;</code></td>
<td><code>&lt;</code></td>
<td>is less than</td>
</tr>
<tr>
<td><code>≤</code></td>
<td><code>&lt;=</code></td>
<td>is less than or equal to</td>
</tr>
</tbody>
</table>
Example: Overtime calculator

- Someone who works more than 40 hours a week is entitled to “time-and-a-half” overtime pay.
- How could we determine: (1) Whether or not an employee is entitled to overtime pay, and (2) if so, how much?
- #1 is pretty simple: use an if-statement.
- For #2, we have to do a different calculation depending on whether the employee will earn overtime pay or not.
- Regular pay formula: hourly wage × hours worked.
- The overtime formula has two parts:
  - The pay for first 40 hours.
  - The pay for additional overtime hours.
def compute_pay(hours, wage):
    if hours <= 40:
        paycheck = hours * wage
    else:
        paycheck = 40 * wage + (hours - 40) * 1.5 * wage
    return paycheck

def main():
    hours_worked = float(input('Enter # of hours worked: '))
    hourly_wage = float(input('Enter hourly wage: '))

    pay = compute_pay(hours_worked, hourly_wage)
    print('Your pay is $' + '{:.2f}'.format(pay))

main()
Example: Hiring decisions

- A hiring manager is trying to decide which candidates to hire
- Each potential hire is evaluated based on GPA, interview performance, and an aptitude exam
- A GPA of at least 3.3 is worth 1 point
- An interview score of 7 or 8 (out of 10) is worth 1 point; a score of 9 or 10 is worth 2 points
- An aptitude test score above 85 is worth 1 point
- Hiring decisions are then based on point totals:
  - 0, 1 or 2 total points: Not hired
  - 3 total points: hired as a Junior Salesperson
  - 4 points: hired as a Manager-in-Training
Example: Hiring decisions

- Let’s look at a function that takes these three values and returns the hiring decision as a string.
- The following Python capabilities/features will help us:
  - The `+=` operator can be used to increment a variable by some amount.
  - `-=`, `*=` and `/=` also exist and perform analogous operations.
  - We can use a variable to maintain a tally or running total.
  - An if-statement can contain `elif` clauses without a final `else` clause.
def decision(gpa, interview, test):
    points = 0  # Point total accumulator

    if gpa >= 3.3:
        points += 1

    if interview >= 9:
        points += 2
    elif interview >= 7:
        points += 1  # note: no else clause

    if test > 85:
        points = points + 1

    if points <= 2:
        return 'Not hired'
    elif points == 3:
        return 'Junior Salesperson'
    else:
        return 'Manager-in-Training'
Ranges and relational operators

- The relational operators can be used to express ranges of values
- Examples:
  - An age in the range 1 through 25, inclusive: $0 \leq age \leq 25$
  - A length in the range 15 (inclusive) through 27: $15 \leq length < 27$
  - A year in the range 1900 through 1972, exclusive of both: $1900 < year < 1972$
More on strings

- Python strings can begin and end with single quote or double quotes
- For example, ‘Stony Brook’ and “Stony Brook” are both valid ways of defining the same string
- We saw earlier that the plus symbol joins two strings into a single longer string (concatenation)
- The asterisk repeats a string a specified number of times
- Example: 'Hello' * 3 will evaluate to 'HelloHelloHello'
String functions

- Because strings are so fundamental to programming, most languages support many functions and other operations for strings. Python is no exception.
- The Python function named `len` (short for “length”) will count the number of characters in a string.
- `len` counts every character in a string, including digits, spaces, and punctuation marks.
- Example:
  ```python
  school = 'Stony Brook University'
  n = len(school)  # n will equal 22
  ```
String methods

- Many other functions on strings are called using a different syntax
- Instead of writing `func(s)` we write `s.func()`
- In this new form, the name of the string is written first, followed by a period, and then the function name is written after the period
- Functions that are called using this syntax are referred to as `methods`
String methods

- As an example of a string method, consider how we might figure out how many words are in a sentence.

- If there is exactly one space between each word we just need to count the number of space characters.

- The method named `count` does exactly that:
  ```python
  sentence = 'It was a dark and stormy night.'
  sentence.count(' ') + 1  # equals 7
  ```

- Note that the argument passed to `count` is a string that contains exactly one character: a single space character.
String methods

- Two other useful methods are **startswith** and **endswith**

- They are both Boolean functions and return **True** or **False** depending on whether a string begins or ends with a specified value

- Examples:

```python
sentence = 'It was a dark and stormy night.'
sentence.startswith('It')    # True
sentence.startswith('it')    # False
sentence.startswith("It's")  # False
sentence.endswith('?')       # False
sentence.endswith('.')       # True
```
String methods

- Another example:

```python
filename = input('Enter a filename: ')
if filename.endswith('.py'):
    print('The file contains a Python program.')
else:
    print('The file does not contain a Python program.')
```