Announcements

Finish Reading Chapter 2

- Sections 2.4 - 2.7 Pages 46 – 87

- Turn-In Questions
  - 2 questions on reading due next Tuesday
Often we will want our programs to do something only if a certain condition is true. That is the flow of our computer programs often needs to branch.

For example:

\[ f(x) = \begin{cases} 
0, & x \leq 0 \\
1, & x > 0 
\end{cases} \]
The `if` Statement

```python
if condition:
    # "if" statements executed if condition is True
    ...
    # next program statements
```

- The `if` statements must be indented by spaces (use 4 spaces).
- The next statement without indentation is the continuation of the program after the `if`.
The *if, elif, and else* Statements

```python
if condition1:
    # “if” statements executed if `condition1` is True
...
elif condition2:
    # “else if” statements executed if `condition1` is False &
    # `condition2` is True
    ....
else:
    # “else” statements executed if all conditions are False
    ....
# next program statements
```

The **elif** and **else** statements are optional extensions of the **if** statement.
Using if Statements

\[ F(x) = \begin{cases} 
0, & x \leq 0 \\
x, & 0 < x < 1 \\
1, & x \geq 1 
\end{cases} \]

```python
x = float(input("Enter a decimal number (i.e. float): "))
if x <= 0:
    Fx = 0
elif x > 0 and x < 1:
    Fx = x
else:
    Fx = 1
print("F(", x, ") is ", Fx, sep='')
```
Continuation lines are needed when entering a multi-line construct. As an example, take a look at this if statement:

```python
>>> the_world_is_flat = True
>>> if the_world_is_flat:
...     print("Be careful not to fall off!")
...     print("Be careful not to fall off!")
```

Typing an end-of-file character (Control-D on Unix, Control-Z on Windows) at the primary prompt causes the interpreter to exit with a zero exit status. If that doesn't work, you can exit the interpreter by typing the following command: `quit()`.
Boolean Expression

Boolean expressions evaluate to bool type values of True or False

- x == 13  # x equals 13
- x != 13  # x does not equal 13
- x >= 13  # x is greater than or equal to 13
- x <= 13  # x is less than or equal to 13
- x > 13   # x is greater than 13
- x < 13   # x is less than 13
The key words *and*, *or*, or *not* can be used in the boolean expressions

```
hpc-login 400% python
>>> x, y = 0, 1.2
>>> x >= 0 and y < 1
False
>>> x >= 0 or y < 1
True
>>> x > 0 or not y > 1
False
>>> -1 < x <= 0   # -1 < x and x <= 0
True
>>> not( x > 0 or y > 0 )
False
>>> x > 0 or y > 1
True
>>> bool(5)  # bool(0 or neg.) is False
True
```
The `while` Statement

```python
while condition:
    # "while" statements executed if condition is True
    ...
    # next program statements
```

- The `while` statements must be indented by spaces (use 4 spaces).
- The next statement without indentation is the continuation of the program after the `while`.
The **break** and **continue** Statements

```python
while condition:
    ...
    break
    ...
# next program statements
```

The **break** statement allows us to break out of a loop even if the condition in the while statement is not met.

```python
while condition:
    ...
    continue
    ...
# next program statements
```

The **continue** statement makes the program skip the rest of the indented code in the **while** loop but then goes back to the beginning of the loop.

*The continue statement is rarely used.*
The **break** and **continue** Statements

```python
x = 11
while x>10:
    # This loop will continue until one enters a number not
    # greater than 10, except if one enter the number 111.
    x = int(raw_input("Enter a number no greater than ten: "))
    if x==111:
        # “if” statements executed only if condition is True
        break
    # The value of x is either less than 10 or exactly 111.
```

This is an example of **nesting** an **if** statement in a while loop. The nested block of statements must be further indented (+4 spaces).
User Defined Functions

Python allows you to define your own functions

```python
import numpy as np
# In cylindrical coordinates calculate the
# distance "d" between a point and the origin
#
def distance(r, theta, z):
    x = r*np.cos(theta)
    y = r*np.sin(theta)
    d = np.sqrt(x**2 + y**2 + z**2)
    return d
```

- The function statements must be indented by spaces
  - use 4 spaces
- The next statement without indentation is the continuation of the program after the function
from __future__ import division, print_function
from math import sqrt, sin, cos, radians

def distance(r, theta, z):
    x = r*cos(theta)
    y = r*sin(theta)
    d = sqrt(x**2 + y**2 + z**2)
    return d

# Enter get a cylindrical point from the user
r = float(input("Enter the r cylindrical coordinate: "))
theta = radians(float(input("Enter the angle in degrees for the theta\ncylindrical coordinate: ")))
z = float(input("Enter the z cylindrical coordinate: "))

print("The distance between the point and the origin is", distance(r, theta, z))
Let's get working
cylindricalDistance.py

```python
#! /usr/bin/env python
""

cylindricalDistance.py is program which calculates
the distance of a point in cylindrical coordinates to the origin
The results are printed to the screen.

Paul Eugenio
PHZ4151C
Jan 23, 2018
""

from __future__ import division, print_function
import numpy as np

def distance(r, theta, z):
""
In cylindrical coordinates calculate the
distance d between a point and the origin
""
    x = r*np.cos(theta)
    y = r*np.sin(theta)
    d = np.sqrt(x**2 + y**2 + z**2)

    return d

# Get cylindrical point data from the user
r = float(input("Enter the r cylindrical coordinate: "))
theta = radians(float(input("Enter the angle in degrees for the theta\n    cylindrical coordinate: ")))
z = float(input("Enter the z cylindrical coordinate: "))

print("The distance between the point and the origin is", distance(r, theta, z))
```