Announcements

•   Lectures are recorded and placed on D2L

•   Homework 1 is due today

•   Homework 2 is due on Tuesday, Sept 30
Execution control structures

• if statements are examples of execution control structures

• Execution control structures are programming language statements that control which statements are executed, i.e., the execution flow of the program

• if statements are, more specifically, conditional structures

• Iteration structures are execution control structures that enable the repetitive execution of a statement or a block of statements

• The for loop statement is an iteration structure that executes a block of code for every item of a sequence
More “if” exercises

1. Print user’s age
   Input: month, day, year of birth
   Output: Age as of 9/23/2014

1. Print user’s FICA tax
   Input: salary for 2014
   Output: FICA tax for the year

1. Print user’s wage for the week
   Input: hours worked, hourly wage
   Output: wage for the week
for loop

Executes a block of code for every item of a sequence
  • If sequence is a string, items are its characters (single-character strings)

name = 'Apple'

char = 'A'
char = 'p'
char = 'p'
char = 'l'
char = 'e'
for loop

Executes a code block for every item of a sequence

- Sequence can be a string, a list, ...
- Block of code must be indented

```python
for word in ['stop', 'desktop', 'post', 'top']:
    if 'top' in word:
        print(word)
print('Done.')
```

`>>> stop
desktop
top
Done.`
Introduction to Computing Using Python

**Built-in function range()**

Function `range()` is used to iterate over a sequence of numbers in a specified range

- To iterate over the n numbers 0, 1, 2, ..., n-1
  
  for i in range(n):

- To iterate over the n numbers k, k+1, k+2, ..., n-1
  
  for i in range(k, n):

- To iterate over the n numbers k, k+c, k+2c, k+3c, ..., n-1
  
  for i in range(k, n, c):

```python
>>> for i in range(0, 16, 4):
    print(i)

0
4
8
12
>>>```

for loop with range

>>> name = ‘Apple’
for i in range(len(name)):
    print(name[i])


Exercise

Write for loops that will print the following sequences:

a) 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
b) 0, 2, 4, 6, 8
c) 20, 30, 40, 50, 60
Exercise

Write a program to find to read 2 lists, and prints their intersection. For example:

Enter a list
[1, 3, 5]
Enter another list
[2, 3, 4, 6, 5]

The Intersection is [3, 5]
More Exercises

Input: an integer n
Output: n! (n * (n-1) * (n-2) ...)

Example:
Enter a positive integer
4
4 factorial is 24

Input: an integer n
Output: a message indicating whether or not n is prime
while loops

Syntax:

```python
while <condition>:
    <body>
```

Execution:

1. Check the condition to see if it is True or False
2. If True, then execute the body
3. Repeat steps 1 and 2
Print each word in the input until you find ‘stop’

words = input('Enter a sequence of words, all in one line of input\n').split()
print('The words before stop are: ')
i = 0
while i < len(words) and words[i] != 'stop':
    print(words[i])
    i = i + 1
Nested loops

If necessary, you can write one loop in the body of another loop

Illustration:

```python
firstNumbers = [2, 4, 1]
secondNumbers = [3, 0, 1]
for number1 in firstNumbers:
    for number2 in secondNumbers:
        print(number1, number2)
```

Output:

```
2 3
2 0
2 1
4 3
4 0
4 1
1 3
1 0
1 1
```
Another example

# Compute the first x factorials

x = int(input('Type a positive integer\n'))
print('The first', x, 'factorials are')
for n in range(1,x+1):
    fact = 1
    for i in range(2,n+1):
        fact *= i
    print(fact)

Type a positive integer
5
The first 5 factorials are
1
2
6
24
120
What is the output?

```python
for i in range(6):
    stars = ''
    for j in range(i):
        stars = stars + '*'
    print(stars)
```
Defining new functions

A few built-in functions we have seen:

- abs(), max(), len(), sum(), print()

New functions can be defined using `def`

def: function definition keyword

cube: name of function

x: variable name for input argument

```
def cube(x):
    res = x ** 3
    return res
```
Function returns value of `res` which can then be used in an expression. The return value is **not** printed.

Function prints value of `res` but does not return anything.
Defining new functions

The general format of a function definition is

```python
def <function name> (<0 or more variables>):
    <indented function body>
```

Let’s develop function `hyp()` that:
- Takes two numbers as input (side lengths a and b of above right triangle)
- Returns the length of the hypotenuse c

```python
>>> hyp(3,4)
5.0
```

```python
import math
def hyp(a, b):
    res = math.sqrt(a**2 + b**2)
    return res
```
Exercise

Write function hello() that:

• takes a name (i.e., a string) as input
• prints a personalized welcome message

Note that the function does not return anything

```python
def hello(name):
    line = 'Welcome, ' + name + ', to the world of Python.'
    print(line)

>>> hello('Julie')
Welcome, Julie, to the world of Python.
```
Exercise

Write function `rng()` that:

- takes a list of numbers as input
- returns the range of the numbers in the list

The range is the difference between the largest and smallest number in the list

```python
def rng(lst):
    res = max(lst) - min(lst)
    return res

>>> rng([4, 0, 1, -2])
6
>>>```
Exercises

1. Write a function `vowels` that returns the number of vowels in a word

2. Write a function `isPrime` that is passed a parameter $n$, and returns True or False, depending on whether or not $n$ is prime (only divisible by 2 and $n$)

3. Write a function `primesLessThan` that is passed a parameter $n$ and returns a list containing all prime number less than $n$
Comments and docstrings

Python programs should be documented

• So the developer who writes/maintains the code understands it
• So the user knows what the program does

Comments

```python
def f(x):
    res = x**2 + 10  # compute result
    return res  # and return it
```

Docstring

```python
def f(x):
    'returns x**2 + 10'
    res = x**2 + 10  # compute result
    return res  # and return it
```
Assignment statement: a second look

A variable does not exist before it is assigned

1. `<expression>` is evaluated and its value put into an object of appropriate type
2. The object is assigned name `<variable>`

```python
>>> a
Traceback (most recent call last):
  File "<pyshell#66>", line 1, in <module>
    a
NameError: name 'a' is not defined
>>> a = 3
>>> b = 2 + 1.3
>>> c = 'three'
>>> d = [1, 2] + [3]
```
Assignment statements

• They look like algebra statements

\[
\begin{align*}
x &= 3 \\
y &= x + 2
\end{align*}
\]

• But they really mean...

\[
\begin{align*}
x &= 3 \quad \# \text{ store the value 3 in } x \\
y &= x + 2 \quad \# \text{ retrieve the value of } x, \text{ add 2, store in } y
\end{align*}
\]

• This is a valid Python statement

\[
\begin{align*}
x &= x + 1 \quad \# \text{ retrieve the (old) value of } x, \text{ add 1. New value of } x \text{ is } 4
\end{align*}
\]
Mutable and immutable types

The object (3) referred to by variable `a` does not change; instead, `a` refers to a new object (6)
  - Integers are immutable

The object ([1, 2, 3]) referred to by `d` changes
  - Lists are mutable
Assignment and mutability

The list that \( c \) refers to changes; \( d \) refers to the same list object, so it changes too.

- Because lists are mutable, a change to \( d \) affects \( c \) also.

\[
\begin{align*}
>>> & a \\
& 6 \\
>>> & b \\
& 3.3 \\
>>> & a = 9 \\
>>> & b \\
& 3.3 \\
>>> & c = d \\
& [1, 7, 9] \\
>>> & d[2] = 9 \\
>>> & c \\
& [1, 7, 9]
\end{align*}
\]

\( a \) and \( b \) refer to the same integer object.

\( a \) now refers to a new object (9); \( b \) still refers to the old object (6).

- Because integers are immutable, change to one does not affect the value of the other.

\[
\begin{align*}
>>> & a \\
& 6 \\
>>> & b \\
& 3.3 \\
>>> & a = 9 \\
>>> & b \\
& 3.3 \\
>>> & c = d \\
& [1, 7, 9] \\
>>> & d[2] = 9 \\
>>> & c \\
& [1, 7, 9]
\end{align*}
\]
Swapping values

Have:

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>tmp</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>6</td>
<td></td>
</tr>
</tbody>
</table>

Want:

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>6</td>
</tr>
</tbody>
</table>

```python
>>> a
3
>>> b
6
>>> tmp = b
>>> b = a
>>> a = tmp
```
Immutable parameter passing

```
def g(x):
    x = 5
```

Variable `x` inside `g()` refers to the object `a` refers to.

Function `g()` did not, and cannot, modify the value of `a` in the interactive shell.

As if we executed `x = a`

This is because `a` refers to an immutable object.
Mutable parameter passing

```python
def h(l):
    l[0] = 5
```

This is because `lst` and `l` refer to an mutable object.

Function call `h(lst)`

If we execute `lst = [1, 2, 3]`, `h(lst)` modifies the value object in the interactive shell.

As if we execute `lst = [5, 2, 3]`...
Exercise

Write function `swapFS()` that:

- takes a list as input
- swaps the first and second element of the list, but only if the list has at least two elements

The function does not return anything

```python
def swapFS(lst):
    if len(lst) > 1:
        lst[0], lst[1] = lst[1], lst[0]

>>> mylst = ['one', 'two', 'three']
>>> swapFS(mylst)
>>> mylst
['two', 'one', 'three']
>>> mylst = ['one']
>>> swapFS(mylst)
>>> mylst
['one']
>>> 
```