PYTHON Command Syllabus

(March 23, 2016)

Variables

Variable Names – In general, variables can be given names created using any sequence of letters, digits, or underscores, subject to the following restrictions:

1. A variable name must begin with either a letter or an underscore

2. Blanks are NOT valid characters to use when creating a variable name

3. Variable names are CASE SENSITIVE (e.g., PYTHON considers "ROOT" and "root" to be two different variables)

4. A variable name cannot be the same as certain *key words* used in PYTHON. These *keywords* are reserved for specific use in PYTHON

PYTHON *keywords*

and	del	from	None	True
as	elif	global	nonlocal	try
assert	else	if	not	while
break	except	import	or	with
class	False	in	pass	yield
continue	finally	is	raise	
def	for	lambda	return	

Input

Inputting *Character* (i.e., *literal)* data:

Form: <variable> = input("prompt")

EX x = input("enter your name ")

Inputting *Arithmetic* (i.e., *numeric)* data:

Form: <variable> = eval(input("prompt")

EX temp = eval(input("Enter a real number value for the temperature ")

Output (i.e., print)

Form: print(*output item 1, output item 2, . . . , output item n)*

EX print("the value of x is ", x, "the value of y is ", y)

Comments

A *comment* is a portion of the program that is not executed. Its purpose is solely one of documentation.

Form: # <comment>

EX # This is a comment

Remark: The code "#" begins the comment. Everything that appears on the line to the right of "#" is a comment.

Remark: Typically, a comment begins at the beginning (leftmost point) of a line. However, a comment can begin elsewhere in a line. For example:

```
x = x + y;      # This computes the value of x + y
```

Remark: In the previous example, the segment "x = x + y" is actually executed by the program. The remainder of the line "# This computes the value of x + y" is interpreted as a comment.

Template of a PYTHON Program

The following is a template which shows the form of a PYTHON program:

```
def main ():
        <statement>
        <statement>
              ⋮
        <statement>
main( )
```

EX

```
def main():
    # This is Joe Blow's Program
    # This program computes the sum of the first 10 natural numbers
    sum  =  0
    for i in range(1,10):
        sum  =  sum + i
    print("the sum is  ", sum)
main()
```

Creating a PYTHON Program

1. From the Start Menu:
   click *Programs*
   then click *PYTHON 3.5*
   then click *IDLE (Python 3.5, 32-bit)*
2. This brings us to the IDLE "interactive environment."
3. Click the *File* menu
   Click *New File*
   Click *Save As*
4. This brings us to the IDLE "programming environment."
5. Write the program in the IDLE "programming environment."
6. Periodically throughout the programming process, we may want to save the most recent version of the program:
   Click the *File* menu
   Click *New File*
   Click *Save As*
6. When you are done writing the program, save the file as explained above

Editing a PYTHON Program Previously Created and Saved

1. From the Start Menu:
   click *Programs*
   then click *PYTHON 3.5*
   then click *IDLE (Python 3.5, 32-bit)*
2. This brings us to the IDLE "interactive environment."
3. Click the *File* menu
   Click *Open*
   Navigate through the directory/file structure to find your program
   Double-click on the file
4. Edit the file just as you would if you were writing it for the first time.

Running a Saved PYTHON Program

1. In order to "run" a program written in PYTHON, the program must be saved as a **.py* file.
2. Click the *Run* menu
3. Select *Run Module*

Arithmetic Operators

The operators *, /, + , – , ** are used for the operations of multiplication, division, addition, subtraction, and exponentiation, respectively.

Ex x = (2 * y + z - 3.2) / w;

EX x = 3*x**2 + 2*x + 4 (same as . . .)

The Modulo Operator

The operator % is used to perform ***modulo*** arithmetic.

Ex The expression 15 % 4 is equal to 3 (the result of 15 mod 4)

PYTHON Intrinsic Library Functions That We Should Know

Intrinsic functions are those functions that are contained in a PYTHON library of functions.

To use the mathematical functions of PYTHON, we must "import" the Math Function Library.

We do this with the statement:

>>>import math ↵

If we want to import the *math* library for use in a *program,* we would do it this way:

```
import math
def main():
    program body
        ⋮
    program body
main()
```

Intrinsic Functions for the Math Library

Function	Meaning	Example
math.exp(x)	e^x	math.exp(1) = 2.718 . . .
math.log(x)	ln(x)	math.log(2.718) = 0.9999
math.sin(x)	sin(x)	math.sin(3.1415/2) = 0.9999
math.cos(x)	cos(x)	math.cos(3.1415) = – 0.9999
math.tan(x)	tan(x)	math.tan(0.785398) = 0.9999
math.asin(x)	arcsin(x)	math.asin(1.0) = 1. 5708
math.acos(x)	arccos(x)	math.acos(1.0) = 0.0
math.atan(x)	arctan(x)	math.atan(1.0) = 0.785398
math.sqrt(x)	$\sqrt{x}$	math.sqrt(4.0) = 2.0

Logical Operators

Operator	Meaning	Example
= =	Equal to	3 = = 2 (Result is False)
! =	Not equal to	3 ! = 2 (Result is True)
>	Greater than	3 > 2 (Result is True)
<	Less than	3 < 2 (Result is False)
> =	Greater than or equal to	3 > = 2 (Result is True)
< =	Less than or equal to	3 < = 2 (Result is False)
and	AND	(3 > 2) and (3 ! = 2) (Result is True)
or	OR	(3 > 2) or (3 = 2) (Result is True)
not	NOT	not(3 = = 2) (Result is True)

IF Structures

Variation 1

```
if <logical expression>:
    <statement>
      ⋮
    <statement>
```

Variation 2

```
if  <logical expression>:
    <statement>
      ⋮
    <statement>
else :
    <statement>
      ⋮
    <statement>
```

Variation 3

```
if  <logical expression 1>:
      <statement>
          ⋮
elif  <logical expression 2>:
      <statement>
          ⋮
elif  <logical expression n>:
      <statement>
          ⋮
else :
      <statement>
          ⋮
```

Variation 4

```
if  <logical expression 1>:
      <statement>
          ⋮
elif  <logical expression 2>:
      <statement>
          ⋮
elif  <logical expression n>:
      <statement>
          ⋮
else :
      <statement>
          ⋮
```

WHILE Structure (DO WHILE Structure)

Form:

```
while <condition> :
      <statement>
      <statement>
      <statement>
```

FOR ...LOOP Structure

Structure (default *increment* of 1):

>>>for i in range(*n):*
    <*statement*>
    <*statement*>
    <*statement*>

range(n) generates the *sequence of numbers* 0,1,2,3, . . . , (n-1)

EX

>>>for i in range(*9):*
    <*statement*>
    <*statement*>
    <*statement*>

range(9) generates the *sequence of numbers* 0,1,2,3, . . . , 9

2nd variation of the "for loop":

Form:

range(<*start*>, n)

This generates the sequence of integers: *start, start* + 1, *start* + 2, . . . , n – 2, n – 1

(i.e., *range*<*start*>, n) generates all integers > = *start* and < n)

start is the value of the first integer in the sequence of integers that increases in *increments of 1*, and ends with the integer *n – 1*

EX

>>>for i in range(*3,10):*
    <*statement*>
    <*statement*>
    <*statement*>

range(3,10) generates the *sequence of numbers* 3,4,5, . . . , 9

3rd variation of the "for loop":

Form:

for i in range(<start>, n, <step>):

step is the *increment* by which consecutive integers in the sequence increase (or decrease).

range(<start>, n, <step>)

generates the sequence of all integers > = *start,* and < n, beginning with *start* and increasing in increments of *step*

EX

>>>for i in range(*3,25,4):*
    <*statement*>
    <*statement*>
    <*statement*>

range(3,25,4) generates the *sequence of numbers* 3,7,11,15,19,23

EX

>>>for i in range(*3,25,4):*
    <*statement*>
    <*statement*>
    <*statement*>

range(3,23,4) generates the *sequence of numbers* 3,7,11,15,19

4th variation of the "for loop":

range(<start>, n, <step>) (negative step (increment) value)

generates the sequence of all integers < = *start,* and > n, beginning with *start* and increasing in increments of *step,*

EX

>>>for i in range(*25,3, – 4):*
    <*statement*>
    <*statement*>
    <*statement*>

range(*25,3, – 4)* generates the *sequence of numbers* 25,21,17,13,9,5

EX

>>>for i in range(*25,5, – 4):*
<statement>
<statement>
<statement>

range(*25,5, – 4)* generates the *sequence of numbers* 25,21,17,13,9

ARRAYS

DEFINING A COLUMN ARRAY (one column and several rows)

FORM:

>>> <array name> = [i for i in range(n)] (Where n is a positive whole number)

This statement defines an array with subscripts 0, 1, 2, 3, . . . , (n - 1)

EX

>>> col = [i for i in range(10)]

This definition statement defines an array named "col" that has entries col[0], col[1], . . . , col[9]

Remark: The subscript, or ***index***, of the array MUST begin with zero.

Therefore, we CAN'T do something like THIS:

>>> col = [i for i in range(1, 10)]

Reason: It does not allow for the array element: col[0]

DEFINING TWO-DIMENSIONAL ARRAYS

To ***define*** two-dimensional arrays, we use the definition statement, using columns subscript ***j*** and row subscript ***i.***

>>> <array name> = [[j for j in range(# of columns)] for i in range(# of rows)]

What this ***really*** is, is:

>>> <array name> = [[# of elements per row] # of rows]

EX

```
>>> sample   =   [[j for j in range(2)] for i in range(4)]
```

This creates an array with entries: sample[0][0], sample[0][1]
sample[1][0], sample[1][1]
sample[2][0], sample[2][1]
sample[3][0], sample[3][1]

User Defined Functions

Remark: These are computer programs in their own right, and they are created to be used by the "main" program or by other functions. Such programs are sometimes called *subprograms.*

Remark: User Defined Functions are defined external to the "main" program.

Defining a Function:

Form:

```
def  <function name> (formal parameters):
     <body>
     <body>
     <body>
     return <value to be returned>
```

EX

```
# The user defined function below has two parameters, x and y.
# The function returns the value of x**2 + y**3

def  ff(x, y):
     answer  = x**2 + y**3
     return answer

def  main():
          u = eval(input("enter a real-number value for u  ")
          v = eval(input("enter a real-number value for v  ")
          print("The value of our function is  ", ff(u, v))

main( )
```

```
enter a real-number value for u 2
enter a real-number value for v 3
The value of our function is 31
```

Remark: The values of u, v are passed to the function *ff* in such a way that *x* is given the value of *u,* and *y* is given the value of *v.* These assignments are done on the basis of *order of appearance.*

Remark: The value that the function returns to the main program is the value possessed by the variable that appears to the right of the word ***return.***

Index