

Python Basics

Data Types



In this video, we will discuss how to work with numbers and strings in Python.

Numbers

- Two types are commonly used:
 - Integer: whole numbers with no decimal (i.e. 5)
 - Floating point: decimal numbers (i.e. 5.736)
- Expressions can include different numbers types...
 - output will be the most complex number type.

Expression contains:	Output will be:
only integers	integer
only float	float
float and integer	float



Python recognizes several different number types, but most scripts work with just **integer** and **floating point** numbers. Integers are whole numbers (i.e. no decimal point) whereas floating point numbers have decimals. In Python expressions, you can mix integers and decimal numbers – the result will have the most complex number type used in the expression. So for an expression that contains both integer and decimal numbers, the result will be a decimal number. It is important to note that if an expression only contains integers, then the result will always be an integer - this is true even if the result should actually be a decimal number. For example, if you divide the integers 3 by 2, then result will be 1 (instead of 1.5).

Number operations

Add

```
>>> 3 + 2
5
```

Subtract

```
>>> 3 - 2
1
```

Multiply

```
>>> 3 * 3
9
```

Division

```
>>> 11.0 / 3
3.6666666666666666
```

Negative

```
>>> -2 + 3
1
```

Exponent

```
>>> 3 ** 3
27
```

Division (truncate)

```
>>> 11.0 // 3
3.0
```

Division remainder

```
>>> 11.0 % 3
2.0
```



Python uses standard symbols as operators for addition, subtraction, multiplication, division, and negative signs. Note that the exponent operator is a **. The truncated division operator rounds the result down to the nearest whole number. The remainder of a division can be obtained by using the % sign in the division.

Strings

- Strings are text - anything that is enclosed by quotes.
 - Can contain any letters, symbols, and numbers.
- Single or double quotes are fine but consistency is needed for a given string.

```
"This is a string"
```

```
'This is also a string'
```

```
"This 'string' is in a string"
```



Strings consist of letters, digits, and symbols that are enclosed by quotes. You can use either double quotes or single quotes to define strings as long as you are consistent. Note that you can embed strings within strings – this will be important when using certain ArcTools.

String formatting

- Backslashes (\) in a string have special significance...
 - they initiate formatting commands.
- The backslash is followed by a letter that specifies the command to be performed.
 - “\n” starts a new line

`print "line 1\nline 2"` → line 1
line 2

- “\t” inserts a tab

`print "\tline 1\nline 2"` → line 1
line 2



When Python finds a backslash in a string, it is interpreted as a formatting command. The letter following the backslash indicates the type of formatting that should occur. The most common formatting commands are `\n` which inserts a line break and `\t` which inserts a tab. Formatting commands are executed when the string is used by a method or function.

Disabling string formatting

- File pathnames use backslashes to separate folders, subfolders, and the file name...
 - backslashes can trigger formatting command
- Must disable formatting for file names.
- Two ways to disable formatting:
 - put an `r` in front of the string:

```
r"C:\NRME387\FeatClass.shp"
```

- use `\\` instead of `\`:

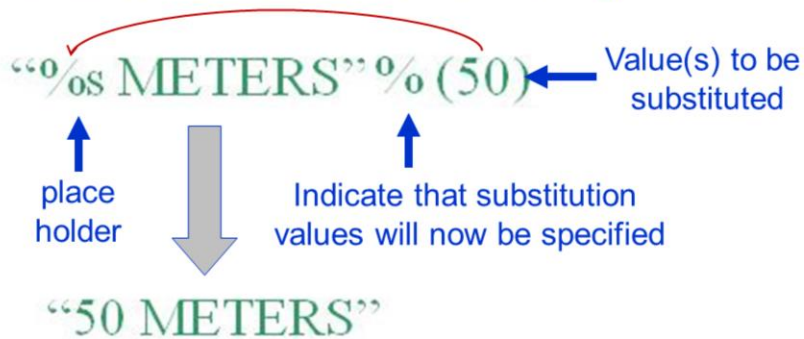
```
"C:\\NRME387\\FeatClass.shp"
```



File workspaces contain backslashes which Python could mistake as a formatting commands. In order to interpret file names correctly, Python needs to be instructed to disregard any formatting commands that it finds in a file pathname string. To disable formatting commands for a string, you can either type the letter “r” before the string or use double backslashes instead of single backslashes.

Variable substitution in strings

- Values can be substituted into a string...



- Multiple substitutions...



We can insert variables into strings using “variable substitution” (note: this referred to as string formatting in some books). In variable substitution, a special character (e.g. %s) is inserted in a string that serves as a place holder for an actual value. After the string, a % sign is followed by the place holder values. When Python processes the string, the value(s) will be substituted in place of the placeholder(s). One value needs to be specified for each placeholder in the string. The place holder values need to be specified in the order that they will appear in the string.

Examples of variable substitution

- 1) Use to specify workspace in a file name

```
OutFolder = r"C:\Python_Workshop"
```

```
FileName = "%s\\testFile.txt" % (OutFolder)
```

(Note: geoprocessor workspace does not work for non-GIS files)

- 2) Allow user or script conditions to determine value in an expression

```
Buffer_distance = raw_input("Specify buffer distance")  
"%s Feet" % (Buffer_distance)
```

Request value from user at run-time

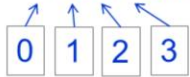


Variable substitution is convenient to use when specifying the workspace part of a file pathname – this allows the script to automatically update file names when a workspace is changed. Variable substitution also helps in incorporating user-defined variables into expressions. The **raw_input** function enables a script to prompt a user to define specific variables when the script is run.

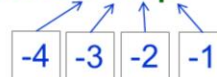
Sequences

- Sequences are ordered collections of items (i.e. characters) – strings are sequences.
- Characters can be retrieved from sequence by specifying item position(s).
- Position can be counted from...
 - the left...
 - or the right...

“Towns.shp”



“Towns.shp”



A sequence is an ordered collection of items – a string is an example of a sequence in which the ordered items are letters, digits, and other characters. Items in a sequence can be retrieved by their position in the sequence. The position can be determined either by 1) starting at 0 and counting up from the 1st item in the sequence, or 2) starting at -1 and counting down from the last item in the sequence.

Retrieving characters from a string

filename = "Towns.shp"

- To retrieve single character:

filename[0] → "T"

- To retrieve multiple characters:

filename[0:5] → "Towns" end position
not inclusive

filename[:5] → "Towns" omit start
pos. if zero

filename[-4:] → ".shp" omit end pos. if
last character

To retrieve a specific item from a sequence (e.g. string), specify the variable that corresponds to the item followed by the position of the item enclosed in brackets.


You can retrieve multiple items by specifying the starting and ending positions in the brackets.

The start position is assumed to be 0 (i.e. the 1st item) if it is omitted. The end position is assumed to be -1 (i.e. the last item) if it is omitted.

When retrieving items from the end of a sequence, it is best to use the last item as the reference position. For example, file extensions are always the last 3 characters in a file name – if the last item in the sequence is used as the reference position, then the positions will be correct regardless of the length of the file name.

String methods

```
string = "NRE 5585: Python"
```

- Make lowercase...
string.lower() → "nre 5585: python"
 - Make uppercase...
string.upper() → "NRE 5585: PYTHON"
 - Capitalize 1st letter...
string.capitalize() → "Nre 5585: python"
 - Replace substring...
string.replace("5585", "387") → "NRE 387: PYTHON"
 - Split string by character...
string.split(":") → ["NRE 5585", "Python"]
- list (more next week) 

Strings have a number of methods (i.e. actions) that they can perform such as making all characters lowercase or uppercase, capitalizing only the 1st letter, replacing a sequence of characters within the string, or splitting the string based on a specified character or characters.

Getting more information on numbers and strings

- Help in the python shell...
 - >>> `help(int)` ← integer
 - >>> `help(float)` ← decimal number
 - >>> `help(str)` ← string
- Python tutorial...
 - Type F1 in the python shell, then click Tutorial
 - Scroll down to sections and click links
 - 3.1.1. Numbers
 - 3.1.2. Strings



You can find out more about the capabilities of numbers and strings by using the **help** function in the python shell or by finding the appropriate section of the python tutorial which can be accessed through the python shell.