UNIVERSITÀ DI PADOVA

Prof. Mauro Conti

T.A. : Hossein Fereidooni & Moreno Ambrosin

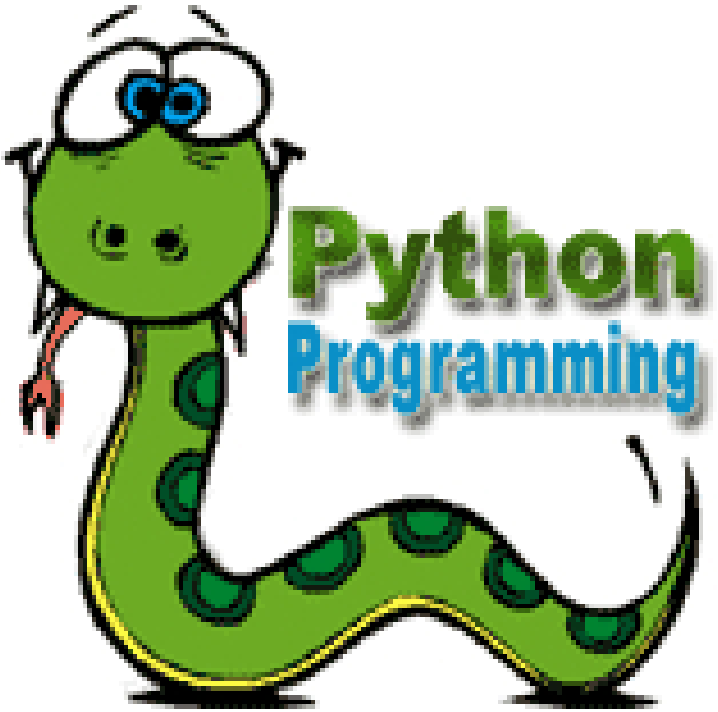
< 2014 March >

OUTLINE

What you will become familiar with during the Python programming course are as follows:

- Basic Operators
- Variable Types
- Numbers
- String
- Lists
- Tuples
- Dictionary
- Decision Making
- Loops
- Functions
- Modules
- Files I/O
- Exceptions
- Classes/Objects

Sometimes using scientific software feels like this...


Python
Programming

Python Features

- ✓ **Easy-to-learn**: Python has relatively few keywords, simple structure, and a clearly defined syntax.
- ✓ **Easy-to-read:** Python code is much more clearly defined and visible to the eyes.
- ✓ **Easy-to-maintain**
- ✓ **Interactive Mode:** Support for functional and structured programming methods as well as OOP.
- ✓ **Very high-level dynamic data types**

More information

If you want know more Information with respect to installation, documentation and so forth go to:

https://www.python.org/

OPERATION TYPES

Python language supports the following types of operators:

1. Arithmetic Operators
2. Comparison Operators
3. Assignment Operators
4. Logical Operators
5. Membership Operators
6. Identity Operators

OPERATION TYPES

Assume variable *a* holds 10 and variable *b* holds 20, then:

Operator	Description	Example
+	Addition - Adds values on either side of the operator	a + b will give 30
-	Subtraction - Subtracts right hand operand from left hand operand	a - b will give -10
*	Multiplication - Multiplies values on either side of the operator	a * b will give 200
/	Division - Divides left hand operand by right hand operand	b / a will give 2
%	Modulus - Divides left hand operand by right hand operand and returns remainder	b % a will give 0
**	Exponent - Performs exponential (power) calculation on operators	a**b will give 10 to the power 20
//	Floor Division - The division of operands where the result is the quotient in which the digits after the decimal point are removed.	9//2 is equal to 4 and 9.0//2.0 is equal to 4.0

Operator	Description	Example
in	Evaluates to true if it finds a variable in the specified sequence and false otherwise.	x in y, here in results in a 1 if x is a member of sequence y.
not in	Evaluates to true if it does not finds a variable in the specified sequence and false otherwise.	x not in y, here not in results in a 1 if x is not a member of sequence y.

OPERATORS PRECEDENCE

- The following table lists all operators from highest precedence to lowest:

Operator	Description
**	Exponentiation (raise to the power)
~ + -	Complement, unary plus and minus
* / % //	Multiply, divide, modulo and floor division
+ -	Addition and subtraction
>> <<	Right and left bitwise shift
<= < > >=	Comparison operators
<> == !=	Equality operators
= %= /= //= -= += *= **=	Assignment operators
in not in	Membership operators
not or and	Logical operators

ASSIGNING VALUE TO VARIABLELS

- The operand to the left of the = operator is the name of the variable and the operand to the right of the = operator is the value stored in the variable.

```
#!/usr/bin/python

counter = 100           # An integer assignment
miles   = 1000.0        # A floating point
name    = "John"        # A string

print counter
print miles
print name
```

While running this program, this will produce the following result:

```
100
1000.0
John
```

STANDARD DATA TYPES

Python has five standard data types:

1-Numbers
2-String
3-List
4-Tuple
5-Dictionary

VARIABLE TYPE NUMBERS

- Python supports different numerical types:
  - int / float / complex (complex numbers)

- Number data types store numeric values. They are immutable data types which means that changing the value of a number data type results in a newly allocated object.
- You can also delete the reference to a number object by using the del statement.

```
var1 = 1
var2 = 10
del var
del var_a, var_b
```

VARIABLE TYPE STRING

- Strings in are identified as a contiguous set of characters in between quotation marks.
- Subsets of strings can be taken using the slice operator ([ ] and [:])
- The plus (+) sign is the string concatenation operator
- The asterisk (*) is the repetition operator

```
#!/usr/bin/python

str = 'Hello World!'

print str           # Prints complete string
print str[0]        # Prints first character of the string
print str[2:5]      # Prints characters starting from 3rd to 5th
print str[2:]       # Prints string starting from 3rd character
print str * 2       # Prints string two times
print str + "TEST" # Prints concatenated string
```

This will produce the following result:

```
Hello World!
H
llo
llo World!
Hello World!Hello World!
Hello World!TEST
```

CREATING STRINGS

- Creating a strings is as simple as assigning a value to a variable.

```
var1 = 'Hello World!'

var2 = "Python Programming"
```

ESCAPE CHARACTERS

- Following table is a list of escape or non-printable characters that can be represented with backslash notation.

\b	0x08	Backspace
\e	0x1b	Escape
\n	0x0a	Newline
\r	0x0d	Carriage return
\s	0x20	Space
\t	0x09	Tab
\v	0x0b	Vertical tab
\x		Character x
\xnn		Hexadecimal notation, where n is in the range 0.9, a.f, or A.F

STRING SPECIAL OPERATIONS

- Assume string variable *a* holds 'Hello' and variable *b* holds 'Python', then:

Operator	Description	Example
+	Concatenation - Adds values on either side of the operator	a + b will give HelloPython
*	Repetition - Creates new strings, concatenating multiple copies of the same string	a*2 will give -HelloHello
[]	Slice - Gives the character from the given index	a[1] will give **e**
[:]	Range Slice - Gives the characters from the given range	a[1:4] will give **ell**
in	Membership - Returns true if a character exists in the given string	**H in a** will give 1
not in	Membership - Returns true if a character does not exist in the given string	**M not in a** will give 1
%	Format - Performs String formatting	

STRING FORMATTING OPERATORS

```
#!/usr/bin/python

print "My name is %s and weight is %d kg!" % ('Zara', 21)
```

When the above code is executed, it produces the following result:

```
My name is Zara and weight is 21 kg!
```

Format Symbol	Conversion
%s	string conversion via str() prior to formatting
%i	signed decimal integer
%d	signed decimal integer
%f	floating point real number

RAW STRINGS

- Raw strings don't treat the backslash as a special character at all

```
#!/usr/bin/python

print 'C:\\nowhere'
```

When the above code is executed, it produces the following result:

```
C:\nowhere
```

Now let's make use of raw string. We would put expression in **r'expression'** as follows:

```
#!/usr/bin/python

print r'C:\\nowhere'
```

When the above code is executed, it produces the following result:

```
C:\\nowhere
```

UNICODE STRING

- As you can see, Unicode strings use the prefix u, just as raw strings use the prefix r.

```
#!/usr/bin/python

print u'Hello, world!'
```

When the above code is executed, it produces the following result:

```
Hello, world!
```

Strings

- A string is a sequence of characters
- A string literal uses quotes 'Hello' or “Hello”
- For strings, + means “concatenate”
- When a string contains numbers, it is still a string
- We can convert numbers in a string into a number using int()

```
>>> str1 = "Hello”
>>> str2 = 'there'
>>> bob = str1 + str2
>>> print bob
Hellothere
>>> str3 = '123'
>>> str3 = str3 + 1
Traceback (most recent call last): File
"<stdin>", line 1, in <module>TypeError:
cannot concatenate 'str' and 'int' objects
>>> x = int(str3) + 1
>>> print x
124
```

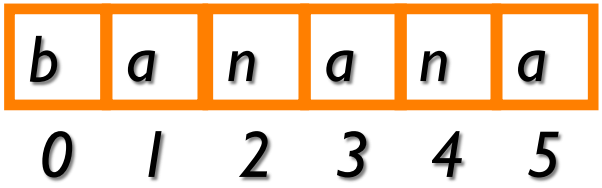
Strings

- We prefer to read data in using strings and then parse and convert the data as we need
- This gives us more control over error situations and/or bad user input
- Raw input numbers must be converted from strings

```
>>> name = raw_input('Enter:')
Enter:Chuck
>>> print name
Chuck
>>> apple = raw_input('Enter:')
Enter:100
>>> x = apple – 10
Traceback (most recent call last): File
"<stdin>", line 1, in <module>TypeError:
unsupported operand type(s) for -: 'str'
and 'int'
>>> x = int(apple) – 10
>>> print x
90
```

Looking Inside Strings

- We can get at any single character in a string using an index specified in square brackets
- The index value must be an integer and starts at zero
- The index value can be an expression that is computed




```
>>> fruit = 'banana'
>>> letter = fruit[1]
>>> print letter
a
>>> n = 3
>>> w = fruit[n - 1]
>>> print w
n
```

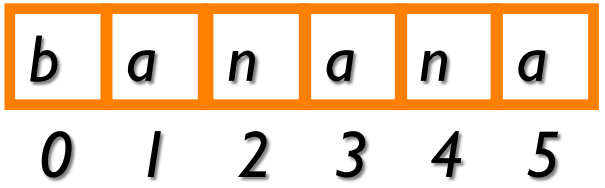
Looking Inside Strings

- You will get a python error if you attempt to index beyond the end of a string.
- So be careful when constructing index values and slices

```
>>> zot = 'abc'
>>> print zot[5]
Traceback (most recent call last):  File
"<stdin>", line  1, in
<module>IndexError: string index out
of range
>>>
```

Strings Have Length

- There is a built-in function len that gives us the length of a string

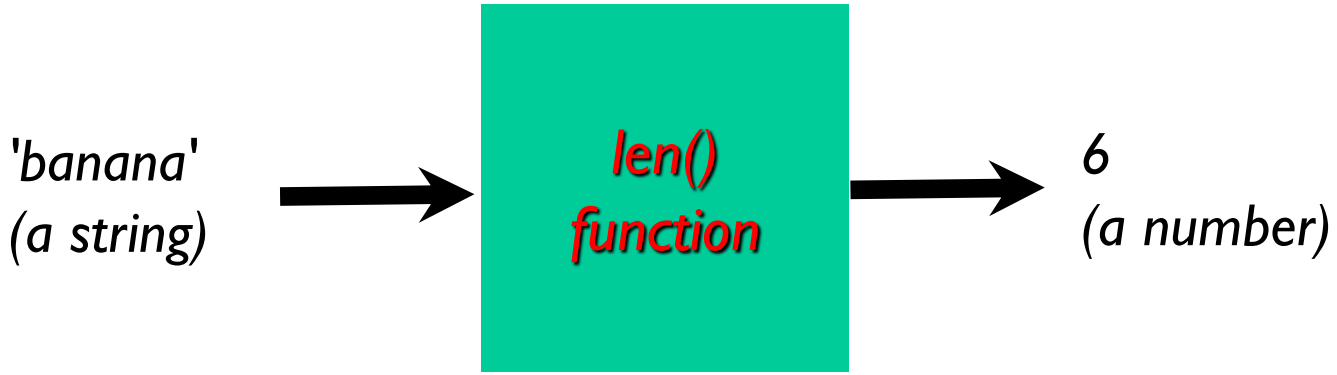



```
>>> fruit = 'banana'
>>> print len(fruit)
6
```

Len Function

```
>>> fruit = 'banana'
>>> x = len(fruit)
>>> print x
6
```

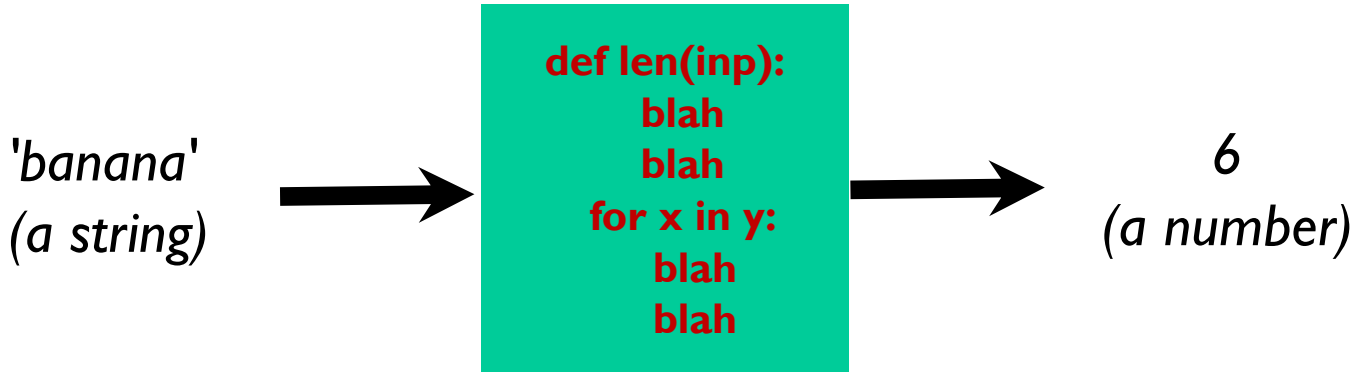
- A function is some stored code that we use.
- A function takes some input and produces an output.



Len Function

```
>>> fruit = 'banana'
>>> x = len(fruit)
>>> print x
6
```

- A function is some stored code that we use.
- A function takes some input and produces an output.



Slicing Strings

M	o	n	t	y		P	y	t	h	o	n
0	1	2	3	4	5	6	7	8	9	10	11

- We can also look at any continuous section of a string using a colon operator
- The second number is one beyond the end of the slice - "up to but not including"
- If the second number is beyond the end of the string, it stops at the end

```
>>> s = 'Monty Python'
>>> print s[0:4]
Mont
>>> print s[6:7]
P
>>> print s[6:20]
Python
```

Slicing Strings

M	o	n	t	y		P	y	t	h	o	n
0	1	2	3	4	5	6	7	8	9	10	11

- If we leave off the first number or the last number of the slice, it is assumed to be the beginning or end of the string respectively

```
>>> s = 'Monty Python'
>>> print s[:2]
Mo
>>> print s[8:]
thon
>>> print s[:]
Monty Python
```

String Concatenation

- When the + operator is applied to strings, it means "concatenation"

```
>>> a = 'Hello'
>>> b = a + 'There'
>>> print b
HelloThere
>>> c = a + '  ' + 'There'
>>> print c
Hello There
>>>
```

Using in as an Operator

- The in keyword can also be used to check to see if one string is "in" another string
- The in expression is a logical expression and returns True or False and can be used in an if statement

```
>>> fruit = 'banana'
>>> 'n' in fruit
True
>>> 'm' in fruit
False
>>> 'nan' in fruit
True
>>> if 'a' in fruit :
...     print 'Found it!'
...
Found it!
>>>
```

String Comparison

```
if word == 'banana':
    print  'All right, bananas.'

if word < 'banana':
    print 'Your word,' + word + ', comes before banana.'
elif word > 'banana':
    print 'Your word,' + word + ', comes after banana.'
else:
    print 'All right, bananas.'
```

String Library

- Python has a number of string functions which are in the string library
- These functions are already *built into* every string - we invoke them by appending the function to the string variable
- These functions do not modify the original string, instead they return a new string that has been altered

```
>>> greet = 'Hello Bob'
>>> zap = greet.lower()
>>> print zap
hello bob
>>> print greet
Hello Bob
>>> print 'Hi There'.lower()
hi there
>>>
```

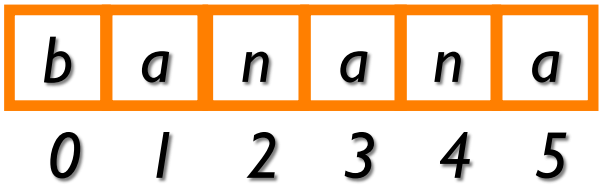
String Library

str.capitalize()
str.title(width[])
str.join()
str.find(sub[, start[, end]])

str.replace(old, new[, count])
str.lower()
str.rstrip([chars])
str.upper()

Searching a String

- We use the find() function to search for a substring within another string
- find() finds the first occurrence of the substring
- If the substring is not found, find() returns -1
- Remember that string position starts at zero




```
>>> fruit = 'banana'
>>> pos = fruit.find('na')
>>> print pos
2
>>> aa = fruit.find('z')
>>> print aa
-1
```

Making everything UPPER CASE

- You can make a copy of a string in lower case or upper case
- Often when we are searching for a string using find() - we first convert the string to lower case so we can search a string regardless of case

```
>>> greet = 'Hello Bob'
>>> nnn = greet.upper()
>>> print nnn
HELLO BOB
>>> www = greet.lower()
>>> print www
hello bob
>>>
```

Search and Replace

- The replace() function is like a "search and replace" operation in a word processor
- It replaces all occurrences of the search string with the replacement string

```
>>> greet = 'Hello Bob'
>>> nstr = greet.replace('Bob','Jane')
>>> print nstr
Hello Jane
>>> nstr = greet.replace('o','X')
>>> print nstr
HellX BXb
>>>
```

Stripping Whitespace

- Sometimes we want to take a string and remove whitespace at the beginning and/or end
- lstrip() and rstrip() to the left and right only
- strip() Removes both begin and ending whitespace

```
>>> greet = '   Hello Bob   '
>>> greet.lstrip()
'Hello Bob   '
>>> greet.rstrip()
'   Hello Bob'
>>> greet.strip()
'Hello Bob'
>>>
```

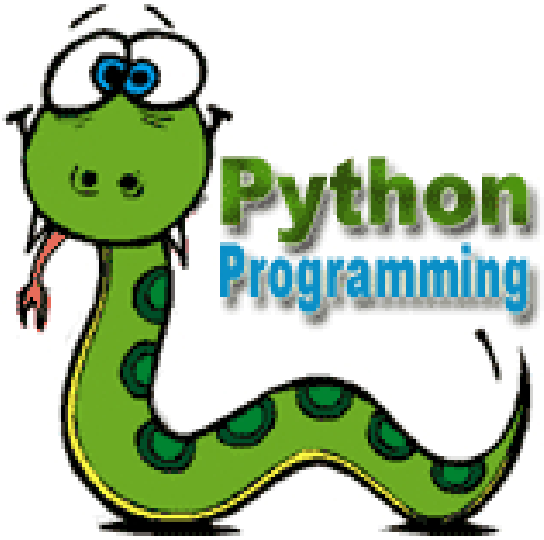
Prefixes

```
>>> line = 'Please have a nice day'
>>> line.startswith('Please')
True
>>> line.startswith('p')
False
```

TYPE CONVERSION

- Sometimes, you may need to perform conversions between the built-in types. To convert between types, you simply use the type name as a function.

Function	Description
int(x)	Converts x to an integer.
long(x)	Converts x to a long integer
float(x)	Converts x to a floating-point number.
complex(real ,imag)	Creates a complex number.
str(x)	Converts object x to a string representation.
tuple(s)	Converts s to a tuple.
list(s)	Converts s to a list.
set(s)	Converts s to a set.
dict(d)	Creates a dictionary.
hex(x)	Converts an integer to a hexadecimal string.


Python
Programming

REFERENCES

1. http://www.tutorialspoint.com/index.htm
2. http://docs.python.org/2/library/stdtypes.html
3. http://docs.python.org/lib/string-methods.html

Contact

- **Website: http://www.math.unipd.it/~hossein/fereidooni.htm**
- **E-mail: hossein@math.unipd.it**
- **Skype: fereidooni1983**