Numbers, Strings and Lists

HORT 530

Lecture 9

Instructor: Kranthi Varala

Core data types

- Numbers
- Strings
- Lists
- Dictionaries
- Tuples
- Files
- Sets

Variable names

- Rules for variable names
  - Must begin with a letter or an underscore
  - May only contain letters, numbers and underscore
- Variable names are case-sensitive i.e.,
  - var ≠ Var ≠ vAr ≠ vaR ≠ VAr ≠ vAR ≠ VaR ≠ VAR
- Good practices for names:
  - Pick a name that implies the value it should contain.
    - Eg., UserName instead of user or name or n.
  - Use descriptive names (not too long)
    - Eg., log2Expression, currentLine
  - Use a consistent style.
    - UserName OR user_name
- Examples of Bad variable names:
  - Test, temp, var, x, y, z, i, j ....

Reserved keywords

- A list of keywords are built in to Python and should be avoided as variable names.
- List of builtins can be retrieved using the '__builtin__' module in Python2 OR 'builtins' module in Python 3.

```
>>> import __builtin__
>>> dir(__builtin__)
```

Python 2

```
>>> import builtins
>>> dir(builtins)
```

Python 3

Reserved keywords Python2

```
>>> import __builtin__
>>> dir(__builtin__)
['ArithmeticError', 'AssertionError', 'AttributeError', 'BaseException', 'BufferError', 'BytesWarning', '
DeprecationWarning', 'EOFError', 'Ellipsis', 'EnvironmentError', 'Exception', 'False', 'FloatingPointErro
r', 'FutureWarning', 'GeneratorExit', 'IOError', 'ImportError', 'ImportWarning', 'IndentationError', 'Ind
exError', 'KeyError', 'KeyboardInterrupt', 'LookupError', 'MemoryError', 'NameError', 'None', 'NotImpleme
nted', 'NotImplementedError', 'OSError', 'OverflowError', 'PendingDeprecationWarning', 'ReferenceError',
'RuntimeError', 'RuntimeWarning', 'StandardError', 'StopIteration', 'SyntaxError', 'SyntaxWarning', 'Syst
emError', 'SystemExit', 'TabError', 'True', 'TypeError', 'UnboundLocalError', 'UnicodeDecodeError', 'Unic
odeEncodeError', 'UnicodeError', 'UnicodeTranslateError', 'UnicodeWarning', 'UserWarning', 'ValueError',
'Warning', 'ZeroDivisionError', '__debug__', '__doc__', '__import__', '__name__', '__package__', 'abs', '
all', 'any', 'apply', 'basestring', 'bin', 'bool', 'buffer', 'bytearray', 'bytes', 'callable', 'chr', 'cl
assmethod', 'cmp', 'coerce', 'compile', 'complex', 'copyright', 'credits', 'delattr', 'dict', 'dir', 'div
mod', 'enumerate', 'eval', 'execfile', 'exit', 'file', 'filter', 'float', 'format', 'frozenset', 'getattr
', 'globals', 'hasattr', 'hash', 'help', 'hex', 'id', 'input', 'int', 'intern', 'isinstance', 'issubclass
', 'iter', 'len', 'license', 'list', 'locals', 'long', 'map', 'max', 'memoryview', 'min', 'next', 'object
', 'oct', 'open', 'ord', 'pow', 'print', 'property', 'quit', 'range', 'raw_input', 'reduce', 'reload', 'r
epr', 'reversed', 'round', 'set', 'setattr', 'slice', 'sorted', 'staticmethod', 'str', 'sum', 'super', 't
uple', 'type', 'unichr', 'unicode', 'vars', 'xrange', 'zip']
```

Reserved keywords Python3

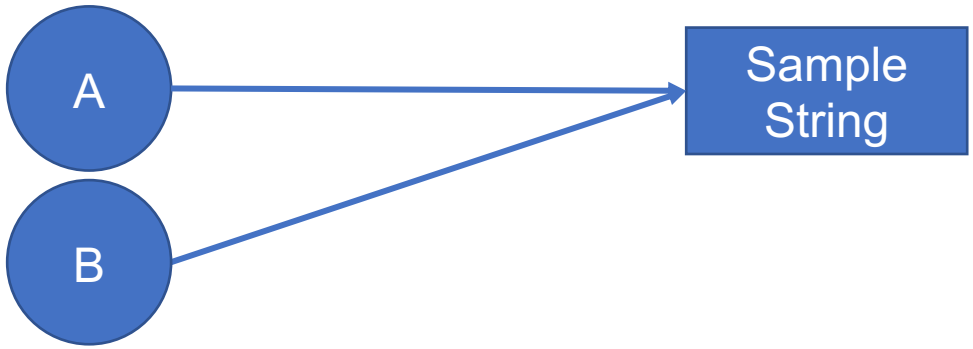
```
>>> import builtins
>>> dir(builtins)
['ArithmeticError', 'AssertionError', 'AttributeError', 'BaseException', 'BlockingIOError', 'BrokenPipeEr
ror', 'BufferError', 'BytesWarning', 'ChildProcessError', 'ConnectionAbortedError', 'ConnectionError', 'C
onnectionRefusedError', 'ConnectionResetError', 'DeprecationWarning', 'EOFError', 'Ellipsis', 'Environmen
tError', 'Exception', 'False', 'FileExistsError', 'FileNotFoundError', 'FloatingPointError', 'FutureWarni
ng', 'GeneratorExit', 'IOError', 'ImportError', 'ImportWarning', 'IndentationError', 'IndexError', 'Inter
ruptedError', 'IsADirectoryError', 'KeyError', 'KeyboardInterrupt', 'LookupError', 'MemoryError', 'Module
NotFoundError', 'NameError', 'None', 'NotADirectoryError', 'NotImplemented', 'NotImplementedError', 'OSEr
ror', 'OverflowError', 'PendingDeprecationWarning', 'PermissionError', 'ProcessLookupError', 'RecursionEr
ror', 'ReferenceError', 'ResourceWarning', 'RuntimeError', 'RuntimeWarning', 'StopAsyncIteration', 'StopI
teration', 'SyntaxError', 'SyntaxWarning', 'SystemError', 'SystemExit', 'TabError', 'TimeoutError', 'True
', 'TypeError', 'UnboundLocalError', 'UnicodeDecodeError', 'UnicodeEncodeError', 'UnicodeError', 'Unicode
TranslateError', 'UnicodeWarning', 'UserWarning', 'ValueError', 'Warning', 'ZeroDivisionError', '__build_
class__', '__debug__', '__doc__', '__import__', '__loader__', '__name__', '__package__', '__spec__', 'abs
', 'all', 'any', 'ascii', 'bin', 'bool', 'bytearray', 'bytes', 'callable', 'chr', 'classmethod', 'compile
', 'complex', 'copyright', 'credits', 'delattr', 'dict', 'dir', 'divmod', 'enumerate', 'eval', 'exec', 'e
xit', 'filter', 'float', 'format', 'frozenset', 'getattr', 'globals', 'hasattr', 'hash', 'help', 'hex', '
id', 'input', 'int', 'isinstance', 'issubclass', 'iter', 'len', 'license', 'list', 'locals', 'map', 'max'
, 'memoryview', 'min', 'next', 'object', 'oct', 'open', 'ord', 'pow', 'print', 'property', 'quit', 'range
', 'repr', 'reversed', 'round', 'set', 'setattr', 'slice', 'sorted', 'staticmethod', 'str', 'sum', 'super
', 'tuple', 'type', 'vars', 'zip']
```

Variable references an Object

- A variable name is a reference to an object in memory.
- The object has a data type, assigned either explicitly by the user or implicitly (dynamic typing) by the variable declaration.
- `A = 'Sample String'` creates a string object in memory and A becomes a reference to that object.

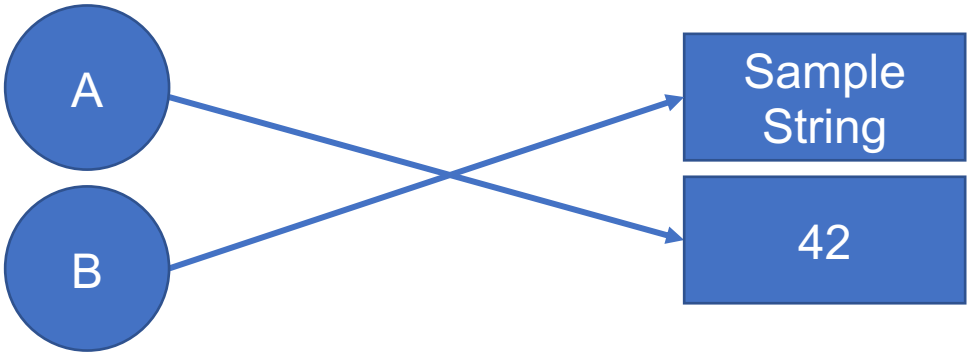
Shared References

- Copying a variable creates a new variable name that also refers to the same object in memory, called a shared reference.
- `B = A` creates a new reference to the same object in memory.



References and dynamic typing

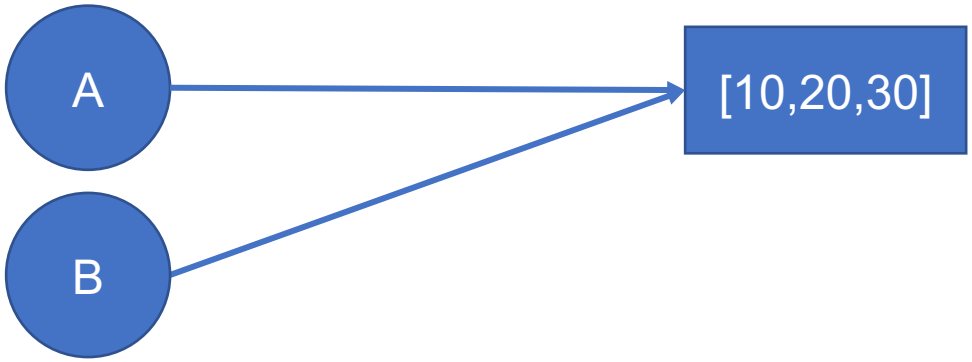
- Dynamic typing allows changing the type of a variable.
- `A = '42'` now changes the apparent data type of A to an integer.



- The reference from A to 'Sample String' is removed.
- B still points to the 'Sample String' object.
- If all variable reference are removed from an object, the object will be marked for removal by Python.
- The process of removing dereferenced objects is called garbage collection

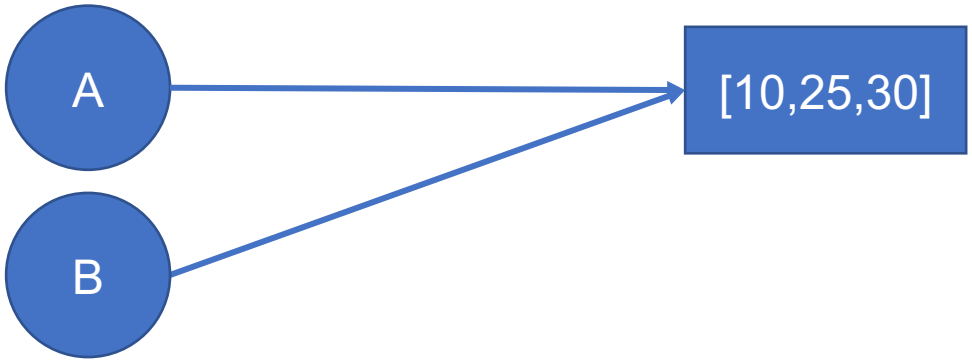
Shared references with mutables

- Shared reference works differently with mutable data types i.e., lists, dictionaries and sets.
- `A = [10,20,30]` creates a List object in memory.
- `B = A` creates a new reference to the List object.



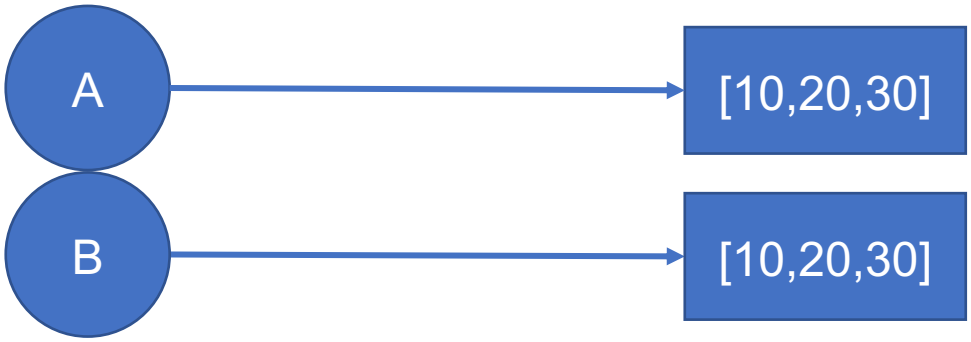
Shared references with mutables

- In-place changes to a mutable object are conveyed to all variables referencing it.
- `A[1] = 25` changes the list object in memory.
- As a result, the value of B[1] is also 25.

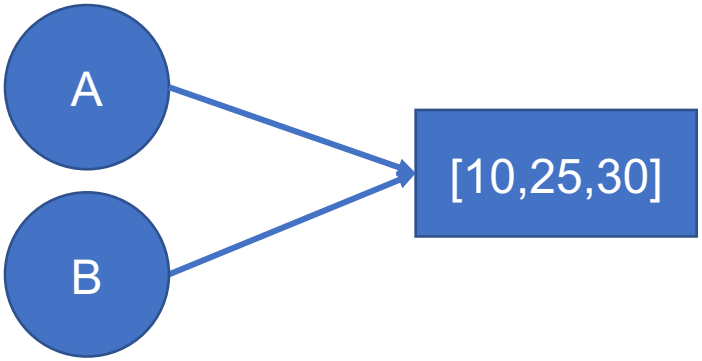


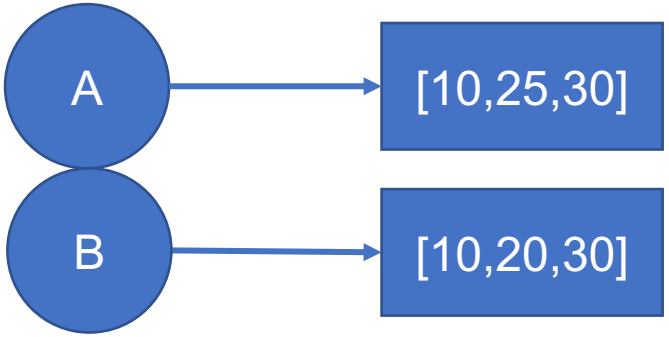
Forcing a copy

- B can be forced to be a separate object, by using the 'copy' module in python
- `A = [10,20,30]` changes the list object in memory.
- B = copy.copy(A) results in a new object that is a physically separate copy of A.



Copies of mutables produce new objects






```
>>> myList=[10,20,30]
>>> listCopy=myList
>>> myList[1]=25
>>> myList
[10, 25, 30]
>>> listCopy
[10, 25, 30]
```

```
>>> import copy
>>> trueCopy=copy.copy(myList)
>>> myList[1]=25
>>> myList
[10, 25, 30]
>>> trueCopy
[10, 20, 30]
```

Numeric type objects

- Can be integers (int), floating points (float), or complex numbers (complex).
- Simple assignment creates an object of number type such as:
  - `myInt = 3` Integer
  - `myFloat = 4.56` Floating-point
  - `myFloat = 2E-10` Floating-point
  - `myComplexNum = 1+2j` Complex number
- Can convert types using int(), float() functions.
- Common numeric operations: +, -, /, *

Numbers : Special operators

- `X ** Y` Raise X to the Power of Y
- `X % Y` Divide X by Y and return the remainder
- `X // Y` Floor of X/Y. Floor is the closest smaller integer.
- Division can be tricky because dividing one integer by another integer can return a float.

Python 2

```
>>> myNum=10/4
>>> myNum
2
>>> myNum=10/4.0
>>> myNum
2.5
```

Python 3

```
>>> myNum=10/4
>>> myNum
2.5
>>> myNum=10/4.0
>>> myNum
2.5
```

Numbers : Operator precedence

- Operators can be mixed together in a single expression.
- Python resolves the use of operators using precedence order: ** > / > % > * > - > +
- `A * B + C / D` : (C/D) first, then (A*B), then add these two values.

```
>>> A,B,C,D = 2,4,3,6.0
>>> A*B+C/D
```

```
>>> A,B,C,D = 2,4,3,6.0
>>> A*B+C/D
8.5
>>> A*B
8
>>> C/D
0.5
```

Other numeric tools

- The 'math' module in python gives you access to more numeric functions.
  - Constants: math.pi, math.e return the values of natural constants *pi* and *e* respectively.
- Functions: pow(), sqrt(), abs(), min(), max(), round(), math.ceil(), math.floor(), math.trunc(), etc.

Strings

- A string object is a 'sequence', i.e., it’s a list of items where each item has a defined position.
- Each character in the string can be referred, retrieved and modified by using its index.
- Strings can be defined with single or double quotes

```
>>> myString = 'Hello'
>>> len(myString)
5
>>> myString[0]
'H'
>>> myString[4]
'o'
>>> myString[5]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: string index out of range
```

```
>>> myString[-1]
'o'
>>> myString[3:]
'lo'
>>> myString[2:5]
'llo'
```

Strings: Operations

- + and * are used for string concatenation and repetition.
- len() returns the length of string.
- Index : S[i] returns the character at index i
- Slice: S[i:j] returns the characters from i to j. S[:] returns a copy of the string S
- Slice with step: S[i:j:k] returns all characters from i to j but moves by k characters instead of 1.

```
>>> myString='VeryLongString'
>>> myString[10]
'r'
>>> myString[8:14]
'String'
>>> myString[8:14:2]
'Srn'
```

Strings: Special characters

- String special characters are defined by a preceding backslash, also called the escape character.
- '\n' = newline
- '\r' = carriage return
- '\t' = tab
- '\' itself can be captured in the string by using '\\' This is called escaping the escape.

Strings: Conversions

- Conversion to and from other data types is done using the appropriate conversion function: int(), str(), float() etc.
- Characters can be converted to and from their ASCII code using the ord() and chr() functions.
- Useful when dealing with FASTQ files

```
>>> S='VeryLongString'
>>> S[10]
'r'
```

```
>>> ord(S[10])
114
>>> chr(114)
'r'
```

String: Methods

- Since every String is an object it has a set of methods that can operate on its value.

```
>>> dir(S)
['__add__', '__class__', '__contains__', '__delattr__', '__doc__', '__eq__', '__format__', '__ge__', '__getattrib
ute__', '__getitem__', '__getnewargs__', '__getslice__', '__gt__', '__hash__', '__init__', '__le__', '__len__', '
__lt__', '__mod__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__rmod__', '__rmu
l__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', '_formatter_field_name_split', '_formatter_pars
er', 'capitalize', 'center', 'count', 'decode', 'encode', 'endswith', 'expandtabs', 'find', 'format', 'index', 'i
salnum', 'isalpha', 'isdigit', 'islower', 'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip', 'p
artition', 'replace', 'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip', 'split', 'splitlines', 'start
swith', 'strip', 'swapcase', 'title', 'translate', 'upper', 'zfill']
```

- Common string methods useful to you are: find(), replace(), split(), lstrip(), rstrip(), capitalize(), lower() etc.
- Since strings are immutable, methods that modify the string, such as replace, concatenate etc. return a new string object.

String: Methods

```
>>> myString="LongString"
>>> myString.upper()
'LONGSTRING'
>>> myString
'LongString'
>>> myString.lower()
'longstring'
>>> myString
'LongString'
>>> myString.capitalize()
'Longstring'
>>> myString
'LongString'
>>> newString=myString.upper()
>>> newString
'LONGSTRING'
```

Lists

- List is a more general sequence object that allows the individual items to be of different types.
- Equivalent to arrays in other languages.
- Lists are mutable, i.e., a list can be changed without having to create a new list object

```
>>> myList=[25,1,2,15]
>>> myList
[25, 1, 2, 15]
```

Lists: Methods

- Built-in methods for Lists:

```
>>> dir(myList)
['__add__', '__class__', '__contains__', '__delattr__', '__delitem__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__', '__gt__', '__hash__', '__iadd__', '__imul__', '__init__', '__init_subclass__', '__iter__', '__le__', '__len__', '__lt__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__reversed__', '__rmul__', '__setattr__', '__setitem__', '__sizeof__', '__str__', '__subclasshook__', 'append', 'clear', 'copy', 'count', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort']
```

- Methods operate on the list in-place.
- For example, sort() will change the order of items in the current list.

```
>>> myList=[25,1,2,15]
>>> myList
[25, 1, 2, 15]
>>> myList.sort()
>>> myList
[1, 2, 15, 25]
```

Lists: Common Methods

- myList.append() : Adds one item to the end of the list.
- myList.extend() : Adds multiple items to the end of the list.
- myList.pop() : Remove last item from the list.
- myList.reverse() : Reverse the order of items in list.
- myList.insert(i,item): Inserts ‘item’ at position i.
- myList.remove(item) : Finds ‘item’ in list and deletes it from the list.

Summary: Data types and their methods

- All variables in Python reference an object with a fixed data type.
- When the variable value is changed, its dynamically assigned an object with a matching data type.
- Be aware of whether you are creating a reference or a copy of an object.
- Python data objects have built-in methods to simplify common operations performed on such data types.
- Use dir() on object to retrieve its associated methods.