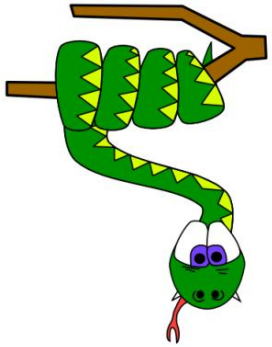
Introduction To Programming

Assist. Prof. Dr. Abdul Hadi Mohammed



python™

More Data Types

Everything is an object

- Everything means everything, including functions and classes (more on this later!)
- Data type is a property of the object and not of the variable

```
>>> x = 7
>>> x
7
>>> x = 'hello'
>>> x
'hello'
>>>
```

Numbers: Integers

- Integer – the equivalent of a C long
- Long Integer – an unbounded integer value.

```
>>> 132224
132224
>>> 132323 **
2
17509376329L
>>>
```

Numbers: Floating Point

- int(x) converts x to an integer
- float(x) converts x to a floating point
- The interpreter shows a lot of digits

```
>>> 1.23232
1.2323200000000001
>>> print 1.23232
1.23232
>>> 1.3E7
13000000.0
>>> int(2.0)
2
>>> float(2)
2.0
```

Numbers: Complex

- Built into Python
- Same operations are supported as integer and float

```
>>> x = 3 + 2j
>>> y = -1j
>>> x + y
(3+1j)
>>> x * y
(2-3j)
```

String Literals

- + is overloaded to do concatenation

```
>>> x = 'hello'
>>> x = x + ' there'
>>> x
'hello there'
```

String Literals

- Can use single or double quotes, and three double quotes for a multi-line string

```
>>> 'I am a string'
'I am a string'
>>> "So am I!"
'So am I!'
```

Substrings and Methods

```
>>> s = '012345'
>>> s[3]
'3'
>>> s[1:4]
'123'
>>> s[2:]
'2345'
>>> s[:4]
'0123'
>>> s[-2]
'4'
```

- **len**(String) – returns the number of characters in the String

- **str**(Object) – returns a String representation of the Object

```
>>> len(x)
6
>>>
str(10.3)
'10.3'
```

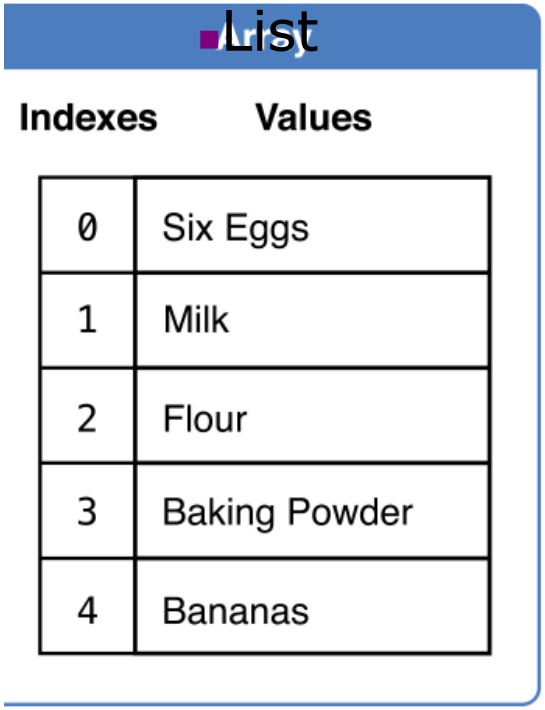
String Formatting

- Similar to C's printf
- <formatted string> % <elements to insert>
- Can usually just use %s for everything, it will convert the object to its String representation.

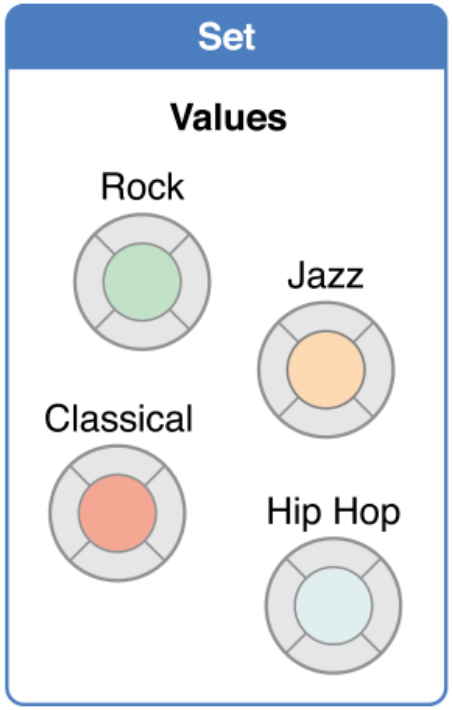
```
>>> "One, %d, three" % 2
'One, 2, three'
>>> "%d, two, %s" % (1,3)
'1, two, 3'
>>> "%s two %s" % (1, 'three')
'1 two three'
>>>
```

Types for Data Collection List, Set, and Dictionary

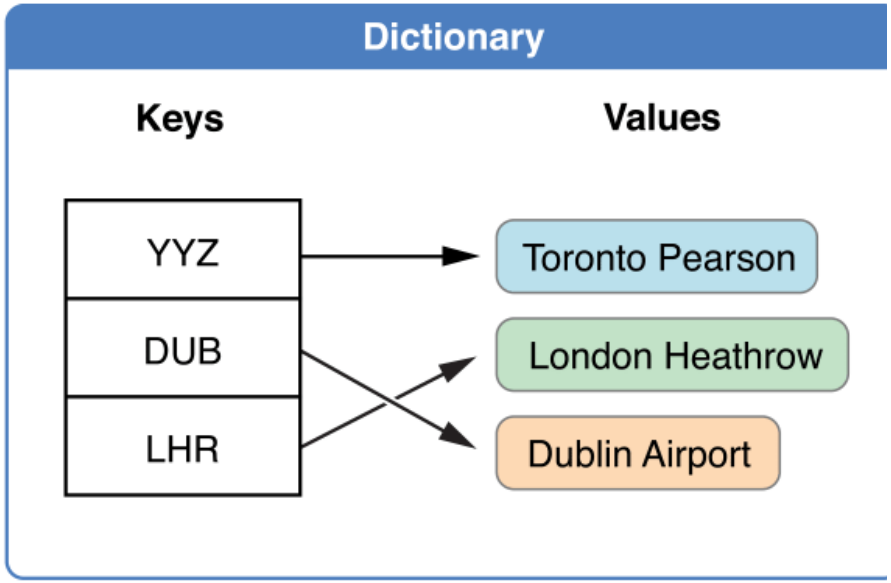
- List



- Ordered



- Unordered list



Pairs of values

Lists

- Ordered collection of data
- Data can be of different types
- Lists are *mutable*
- Issues with shared references and mutability
- Same subset operations as Strings

```
>>> x = [1,'hello', (3 + 2j)]
>>> x
[1, 'hello', (3+2j)]
>>> x[2]
(3+2j)
>>> x[0:2]
[1, 'hello']
```

List Functions

- list.append(x)
  - Add item at the end of the list.
- list.insert(i,x)
  - Insert item at a given position.
  - Similar to a[i:i]=[x]
- list.remove(x)
  - Removes first item from the list with value x
- list.pop(i)
  - Remove item at position I and return it. If no index I is given then remove the first item in the list.
- list.index(x)
  - Return the index in the list of the first item with value x.
- list.count(x)
  - Return the number of time x appears in the list
- list.sort()
  - Sorts items in the list in ascending order
- list.reverse()
  - Reverses items in the list

Lists: Modifying Content

- **x[i] = a** reassigns the ith element to the value a
- Since x and y point to the same list object, *both* are changed
- The method **append** also modifies the list

```
>>> x = [1,2,3]
>>> y = x
>>> x[1] = 15
>>> x
[1, 15, 3]
>>> y
[1, 15, 3]
>>> x.append(12)
>>> y
[1, 15, 3, 12]
```

Lists: Modifying Contents

- The method **append** modifies the list and returns **None**
- List addition (**+**) returns a new list

```
>>> x = [1,2,3]
>>> y = x
>>> z = x.append(12)
>>> z == None
True
>>> y
[1, 2, 3, 12]
>>> x = x + [9,10]
>>> x
[1, 2, 3, 12, 9, 10]
>>> y
[1, 2, 3, 12]
>>>
```

Using Lists as Stacks

- You can use a list as a stack

```
>>> a = ["a", "b", "c“,”d”]
>>> a
['a', 'b', 'c', 'd']
>>> a.append("e")
>>> a
['a', 'b', 'c', 'd', 'e']
>>> a.pop()
'e'
>>> a.pop()
'd'
>>> a = ["a", "b", "c"]
>>>
```

Tuples

- Tuples are *immutable* versions of lists
- One strange point is the format to make a tuple with one element:

  ',' is needed to differentiate from the mathematical expression (2)

```
>>> x = (1,2,3)
>>> x[1:]
(2, 3)
>>> y = (2,)
>>> y
(2,)
>>>
```

Sets

- A set is another python data structure that is an unordered collection with no duplicates.

```
>>> setA=set(["a","b","c","d"])
>>> setB=set(["c","d","e","f"])
>>> "a" in setA
True
>>> "a" in setB
False
```

Sets

```
>>> setA - setB
{'a', 'b'}
>>> setA | setB
{'a', 'c', 'b', 'e', 'd', 'f'}
>>> setA & setB
{'c', 'd'}
>>> setA ^ setB
{'a', 'b', 'e', 'f'}
>>>
```

Dictionaries

- A set of key-value pairs
- Dictionaries are *mutable*

```
>>> d= {‘one’ : 1, 'two' : 2, ‘three’ : 3}
>>> d[‘three’]
3
```

Dictionaries: Add/Modify

- Entries can be changed by assigning to that entry

```
>>> d
{1: 'hello', 'two': 42, 'blah': [1, 2, 3]}
>>> d['two'] = 99
>>> d
{1: 'hello', 'two': 99, 'blah': [1, 2, 3]}
```

- Assigning to a key that does not exist adds an entry

```
>>> d[7] = 'new entry'
>>> d
{1: 'hello', 7: 'new entry', 'two': 99, 'blah': [1, 2, 3]}
```

Dictionaries: Deleting Elements

- The **del** method deletes an element from a dictionary

```
>>> d
{1: 'hello', 2: 'there', 10: 'world'}
>>> del(d[2])
>>> d
{1: 'hello', 10: 'world'}
```

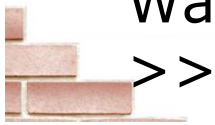
Iterating over a dictionary

```
>>>address={'Wayne': 'Young 678', 'John': 'Oakwood 345',
  'Mary': 'Kingston 564'}
 >>>for k in address.keys():
            print(k,":", address[k])

Wayne : Young 678
John : Oakwood 345
Mary : Kingston 564
>>>

>>> for k in sorted(address.keys()):
   print(k,":", address[k])

John : Oakwood 345
Mary : Kingston 564
Wayne : Young 678
>>>
```



Copying Dictionaries and Lists

- The built-in **list** function will copy a list
- The dictionary has a method called **copy**

```
>>> l1 = [1]
>>> l2 = list(l1)
>>> l1[0] = 22
>>> l1
[22]
>>> l2
[1]
```

```
>>> d = {1 : 10}
>>> d2 = d.copy()
>>> d[1] = 22
>>> d
{1: 22}
>>> d2
{1: 10}
```

Data Type Summary

Integers: 2323, 3234L

Floating Point: 32.3, 3.1E2

Complex: 3 + 2j, 1j

Lists: l = [1,2,3]

Tuples: t = (1,2,3)

Dictionaries: d = {'hello' : 'there', 2 : 15}

- Lists, Tuples, and Dictionaries can store any type (including other lists, tuples, and dictionaries!)
- Only lists and dictionaries are mutable
- All variables are references

String Formatting in Python

String formatting is the process of embedding variables, expressions, or data into strings. Python provides multiple ways to achieve this.

```
1. name = "Alice"
2. age = 25
3. result = "My name is " + name + " and I am " + str(age) +
" years old."
4. print(result)
```

1. The + Operator (String Concatenation)

This is the simplest way to combine strings and variables.

2. `%` Operator (Old-Style Formatting)

This is an older way to format strings, inspired by the C language.

Syntax:

```
"format string" % (values)
```

Format Specifiers:

- `%s`: String
- `%d`: Integer
- `%f`: Float
- `%.2f`: Float with 2 decimal places

Example:

```
name = "Alice"
age = 25
result = "My name is %s and I am %d years
old." % (name, age)
print(result)
```

3. `str.format()` Method (New-Style Formatting)

Introduced in Python 2.6 and Python 3.0, this is a more powerful and flexible way to format strings.

Syntax:

```
"format string".format(values)
```

Example:

```
name = "Alice"
age = 25
result = "My name is {} and I am {} years
old.".format(name, age)
print(result)
```

Positional and Keyword Arguments:

```
result = "My name is {0} and I am {1} years old.".format(name, age)  # Positional
result = "My name is {name} and I am {age} years old.".format(name=name, age=age)  # Keyword
```

4. f-Strings (Formatted String Literals)

Introduced in Python 3.6, **f-strings** are the most modern and preferred way to format strings.

Syntax:

```
f"string {expression}"
```

Features:

- Directly embed variables and expressions.
- Support for inline expressions:

Example:

```
name = "Alice"
age = 25
result = f"My name is {name} and I am {age} years old."
print(result)
```

```
result = f"I will be {age + 5} years old in 5 years."
print(result)  # Output: I will be 30 years old in 5 years.
```

- Format numbers easily:

```
value = 123.456
result = f"Value: {value:.2f}"  # Output: Value: 123.46
```

5. Template Strings (Using the `string` Module)

Template strings are a simpler alternative for formatting, suitable for cases where the formatting source is external.

Syntax:

```
from string import Template

template = Template("My name is $name and I am $age years old.")
result = template.substitute(name="Alice", age=25)
print(result)
```

Comparison of Methods

Method	Flexibility	Readability	Performance	Recommended Use
+ Operator	Low	Low	High	Simple concatenation with few variables.
% Operator	Medium	Medium	Medium	Avoid for new code (outdated).
`str.format()`	High	Medium	Medium	Complex formatting with reordering.
f-Strings	**High**	**High**	**Highest**	**Preferred method for most cases.**
Template Strings	Low	High	Medium	Safe external template processing.

Practice Examples

Try these exercises to solidify your understanding:

1. **Concatenate variables into a string using all methods.**
2. **Format a floating-point number to two decimal places using `str.format()` and f-strings.**
3. **Create a template string and substitute values into it.**