Python Data Types

Python Tuples

- A tuple is a sequence of immutable Python objects.
- Tuples are sequences, just like lists.
- The differences between tuples and lists are, the tuples cannot be changed unlike lists and tuples use parentheses, whereas lists use square brackets.

- tup1 = ('physics', 'chemistry', 1997, 2000)
- tup2 = (1, 2, 3, 4, 5, 6, 7)
- print (tup2[1:5])
- (2, 3, 4, 5)
- ----------
- >>> tup1 = (12, 34.56);
- >>> tup2 = ('abc', 'xyz')
- >>> tup3 = tup1 + tup2
- >>> print(tup3)
- (12, 34.56, 'abc', 'xyz')
- >>>

Delete Tuple Elements

- Removing individual tuple elements is not possible.
- To explicitly remove an entire tuple, just use the **del** statement.

- >>> tup = ('physics', 'chemistry', 1997, 2000);
- >>> print(tup)
- ('physics', 'chemistry', 1997, 2000)
- >>> del tup;

Basic Tuples Operations

Python Expression	Results	Description
len((1, 2, 3))	3	Length
(1, 2, 3) + (4, 5, 6)	(1, 2, 3, 4, 5, 6)	Concatenation
('Hi!',) * 4	('Hi!', 'Hi!', 'Hi!', 'Hi!')	Repetition
3 in (1, 2, 3)	True	Membership
for x in (1, 2, 3): print x,	1 2 3	Iteration

Built-in Tuple Functions

- **Max() ,Min() Method:**
- tuple2 = (123, 200),
- >>> max(tuple2)
- 200
- min(tuple2)
- 123

- **tuple() Method:**
- aList = [123, 'xyz', 'zara', 'abc']
- >>> aTuple = tuple(aList)
- >>> print(aTuple)
- (123, 'xyz', 'zara', 'abc')

Python Lists

- >>> list = ['physics', 'chemistry', 1997, 2000];
- >>> print (list[2])
- 1997

- >>> list[2] = 2001
- >>> print(list)
- ['physics', 'chemistry', 2001, 2000]

Delete List Elements

- >>> list1 = ['physics', 'chemistry', **1997**, 2000];
- >>> del list1[2];
- >>> print (list1)
- ['physics', 'chemistry', 2000]

- len(list)
- max(list)
- min(list)
- list(seq)

Methods

- The method **append()** appends a passed *obj* into the existing list.

- >>> aList = [123, 'xyz', 'zara', 'abc'];
- >>> aList.append(2009);
- >>> print (aList)
- [123, 'xyz', 'zara', 'abc', 2009]
- >>>

- The method **count()** returns count of how many times *obj* occurs in list.

- [123, 'xyz', 'zara', 'abc', 2009]
- >>> print ("Count for 123 : ", aList.count(123))
- Count for 123 : 1

- >>> print ("Count for zara : ", aList.count('zara'))
- Count for zara : 1

- The method **extend()** appends the contents of *seq* to list.

- >>> aList = [123, 'xyz', 'zara', 'abc', 123];
- >>> bList = [2009, 'manni'];
- >>> aList.extend(bList)
- >>> print ("Extended List : ", aList)

Extended List : [123, 'xyz', 'zara', 'abc', 123, 2009, 'manni']

- The method **index()** returns the lowest index in list that *obj* appears.

- >>> aList = [123, 'xyz', 'zara', 'abc'];
- >>> print ("Index for xyz : ", aList.index('xyz'))

- Index for xyz : 1

- The method **insert()** inserts object *obj* into list at offset *index.*

- >>> aList = [123, 'xyz', 'zara', 'abc']
- >>> aList.insert(3, 2009)
- >>> print ("Final List : ", aList)

- Final List : [123, 'xyz', 'zara', 2009, 'abc']

- **remove()**
- aList = [123, 'xyz', 'zara', 'abc', 'xyz'];
- aList.remove('xyz');
- print ("List : ", aList)
- aList.remove('abc');
- print ("List : ", aList)

- List : [123, 'zara', 'abc', 'xyz']
- List : [123, 'zara', 'xyz']

- The method **reverse()** reverses objects of list in place.

- aList = [123, 'xyz', 'zara', 'abc', 'xyz']; aList.reverse();
- print ("List : ", aList)

- List : ['xyz', 'abc', 'zara', 'xyz', 123]

Python Dictionary

- Each key is separated from its value by a colon (:), the items are separated by commas.
- the whole thing is enclosed in curly braces.
- Keys are unique within a dictionary.

- >>> dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}
- >>> print (dict['Name'])

- **dict['Name']: Zara**

Updating Dictionary

- dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}

- dict['Age'] = 8; # update existing entry
- dict['School'] = "DPS School"; # Add new entry
- print ("dict['Age']: ", dict['Age'])
- print ("dict['School']: ", dict['School'])

- dict['Age']: 8
- dict['School']: DPS School

Delete Dictionary Elements

- dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}

- del dict['Name']; # remove entry with key 'Name'
- dict.clear(); # remove all entries in dict
- del dict ; # delete entire dictionary

Built-in Dictionary Functions & Methods

- **len() Method:**

- >>> dict = {'Name': 'Zara', 'Age': 7};
- >>> print ("Length : %d" % len (dict))

- Length : 2

- dict.clear():The method **clear()** removes all items from the dictionary.

- >>> dict = {'Name': 'Zara', 'Age': 7};
- >>> dict.clear()
- >>> print(dict)
- {}

- The method **copy()** returns a shallow copy of the dictionary.

- >>> dict1 = {'Name': 'Zara', 'Age': 7};
- >>> dict2 = dict1.copy()
- >>> print(dict2)

- {'Age': 7, 'Name': 'Zara'}

- The method **items()** returns a list of dict's (key, value) tuple pairs.

- >>> print (dict1.items())

- dict_items([('Age', 7), ('Name', 'Zara')])

- The method **values()** returns a list of all the values available in a given dictionary.

- >>> print (dict1.values())

- **dict_values([7, 'Zara'])**

- The method **update()** adds dictionary dict2's key-values pairs in to dict.

- >>> dict = {'Name': 'Zara', 'Age': 7}
- >>> dict2 = {'Gender': 'female' }
- >>> dict.update(dict2)
- >>> print(dict)

- **{'Age': 7, 'Name': 'Zara', 'Gender': 'female'}**

- **keys() Method:** The method **keys()** returns a list of all the available keys in the dictionary.

- >>> dict = {'Name': 'Zara', 'Age': 7}
- >>> print(dict.keys())

- **dict_keys(['Age', 'Name', 'Gender'])**

Python Strings

- var1 = 'Hello World!'
- var2 = "Python Programming"

- print ("var1[0]: ", var1[0])
- Print ("var2[1:5]: ", var2[1:5])

- **var1[0]: H**
- **var2[1:5]: ytho**

String Formatting Operator

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

- My name is **Zara** and **weight** is 21 kg!

Built-in String Methods

- capitalize()
- It returns a copy of the string with only its first character capitalized.

- >>> str = "this is string example....wow!!!";
- >>> print (str.capitalize())
- This is string example....wow!!!

count()

- The method **count()** returns the number of occurrences of substring sub in the range [start, end].

- str = "this is string example....wow!!!";
- sub = "i";
- print ("str.count(sub, 4, 40) : ", str.count(sub, 4, 40))

- sub = "wow";
- Print ("str.count(sub) : ", str.count(sub))

- **str.count(sub, 4, 40) : 2**
- **str.count(sub) : 1**

find()

- It determines if string *str* occurs in string, or in a substring of string if starting index *beg* and ending index *end* are given.

- str1 = "this is string example....wow!!!";
- str2 = "exam";

- print str1.find(str2)
- print str1.find(str2, 10)
- print str1.find(str2, 40)

- **15**
- **15**
- **-1**

isdigit()

- The method **isdigit()** checks whether the string consists of digits only.

- str = "123456"; # Only digit in this string
- print str.isdigit()

- str = "this is string example....wow!!!";
- print str.isdigit()

- **True**
- **False**

islower()

- The method **islower()** checks whether all the case-based characters (letters) of the string are lowercase.

- str = "THIS is string example....wow!!!";
- print str.islower()

- str = "this is string example....wow!!!";
- print str.islower()

- **False**
- **True**

isupper()

- The method **isupper()** checks whether all the case-based characters (letters) of the string are uppercase.

- str = "THIS IS STRING EXAMPLE....WOW!!!";
- print str.isupper()

- str = "THIS is string example....wow!!!";
- print str.isupper()

- **True**
- **False**

- The method **max()** returns the max alphabetical character from the string *str*.

- str = "this is really a string example....wow!!!";
- print "Max character: " + max(str)

- str = "this is a string example....wow!!!";
- print "Max character: " + max(str)

- **Max character: y**
- **Max character: x**

Python Numbers

- **int (signed integers)**
- **long (long integers)**
- **float (floating point real values)**
- **complex (complex numbers)**

- Type int(x) to convert x to a plain integer.

- Type long(x) to convert x to a long integer.

- Type float(x) to convert x to a floating-point number.

- Type complex(x) to convert x to a complex number with real part x and imaginary part zero.

Mathematical Functions

- abs(x)
- exp(x)
- log10(x)
- log(x)
- max(x1, x2,...)
- min(x1, x2,...)
- pow(x, y)
- sqrt(x)

- Sin(x)
- cos(x)
- tan(x)
- degrees(x)
- radians(x)

Fruitful functions

Return values

- Some of the built-in functions we have used, such as the math functions, produce results.
- Calling the function generates a value, which we usually assign to a variable or use as part of an expression.
- The first example is area, which returns the area of a circle with the given radius:

```
def area(radius):
    temp = math.pi * radius**2
    return temp
```

- **in a fruitful function the return statement includes an expression.**
- Sometimes it is useful to have multiple return statements, one in each branch of a conditional:

```
def absolute_value(x):
    if x >= 0:
        return x
    else:
        return -x
```

- **As soon as a return statement executes, the function terminates without executing any subsequent statements.**

- Code that appears after a return statement, or any other place the flow of execution can never reach, is called **dead code**.

- Write a compare function that
- returns 1 if x > y,
- 0 if x == y,
- and -1 if x < y.

Incremental development

- **The goal of incremental development is to avoid long debugging sessions by adding and testing only a small amount of code at a time.**

- As an example, suppose you want to find the distance between two points, given by the coordinates (x_1, y_1) and (x_2, y_2).

- By the Pythagorean theorem, the distance is:

- **distance = √(x2 − x1)2 + (y2 − y1)2**

- The first step is to consider what a distance function should look like in Python.

- In this case, the inputs are two points, which you can represent using four numbers. The return value is the distance, which is a floating-point value.

- Already you can write an outline of the function:

- **def distance(x1, y1, x2, y2):**
- **   return 0.0**

```
def distance(x1, y1, x2, y2):
    dx = x2 - x1
    dy = y2 - y1
    print ('dx is', dx)
    print ('dy is', dy)
    return 0.0
```

```
def distance(x1, y1, x2, y2):
    dx = x2 - x1
    dy = y2 - y1
    dsquared = dx**2 + dy**2
    result = math.sqrt(dsquared)
    return result
```

- The key aspects of the process are:

- Start with a working program and make small incremental changes. At any point, if there is an error, you should have a good idea where it is.

- Use temporary variables to hold intermediate values so you can display and check them.

- Once the program is working, you might want to remove some of the scaffolding or consolidate multiple statements into compound expressions, but only if it does not make the program difficult to read.

- *Use incremental development to write a function called **squareArea**. To calculate the area of a square.*

Composition

- We can call one function from within another. This ability is called **composition**.
- we'll write a function that takes two points, the center of the circle and a point on the perimeter, and computes the area of the circle.
- Assume that the center point is stored in the variables xc and yc, and the perimeter point is in xp and yp.
- The first step is to find the radius of the circle, which is the distance between the two points.
- We just wrote a function, distance, that does that:

- radius = distance(xc, yc, xp, yp)
- The next step is to find the area of a circle with that radius; we just wrote that, too:
- result = area(radius)

- Encapsulating these steps in a function, we get:

- def **circle_area**(xc, yc, xp, yp):
- radius = **distance**(xc, yc, xp, yp)
- result = **area**(radius)
- return result

Boolean functions

- Functions can return boolean, which is often convenient for hiding complicated tests inside functions.

```
def is_divisible(x, y):
    if x % y == 0:
        return True
    else:
        return False
```

- Write a function is_between(x, y, z) that returns True if $x \leq y \leq z$ or False otherwise.

Leap of faith

- When you come to a function call, instead of following the flow of execution, you *assume* that the function works correctly and returns the right result.

- When you call **math.cos** or **math.exp**, you **don't examine the bodies of those functions**.

- You just assume that they work because the people who wrote the built-in functions were good programmers.

- we can use the user defined function without looking at the body again if it is already tested for correctness.