PYTHON

Slides Courtesy: Prof. Paul Fodor, SBU

Classes

```
import math
class Circle:
    # Construct a circle object
    def __init__(self, radius = 1):
        self.radius = radius
    def getPerimeter(self):
        return 2 * self.radius * math.pi
    def getArea(self):
        return self.radius * self.radius *
math.pi
    def setRadius(self, radius):
        self.radius = radius
    def __str__(self):
        return "Circle: radius=" +
str(self.radius)
```

https://www.python-course.eu/python3_magic_methods.php

```
from Circle import Circle

def main():
    # Create a circle with radius 1
    circle1 = Circle()
    print("The area of the circle of radius", circle1.radius,
        "is", circle1.getArea())
    # Create a circle with radius 25
    circle2 = Circle(25)
    print("The area of the circle of radius", circle2.radius,
        "is", circle2.getArea())
    # Create a circle with radius 125
    circle3 = Circle(125)
    print("The area of the circle of radius", circle3.radius,
        "is", circle3.getArea())
    # Modify circle radius
    circle2.radius = 100
    print("The area of the circle of radius", circle2.radius,
        "is", circle2.getArea())

main() # Call the main function
```

```
crcle = Circle()
crcle.setRadius(15)
print(crcle)
```

Adding fields to Objects dynamically

```
class Employee:
    pass    #null operation

# Create an empty employee record
john = Employee()

# Add the fields of the record
john.name = 'John Doe'
john.dept = 'computer lab'
john.salary = 1000
```

Exceptions

```
import math
class Circle:
    # Construct a circle object
    def __init__(self, radius = 1):
        self.radius = radius
    def getPerimeter(self):
        return 2 * self.radius * math.pi
    def getArea(self):
        return self.radius * self.radius * math.pi
    def setRadius(self, radius):
        raise RuntimeError("Negative radius")
        self.radius = radius
    def __str__(self):
        return "Circle: radius=" + str(self.radius)

crcle = Circle()
crcle.setRadius(-15)
print(crcle)
```

Write/Read in/from File

```
def main():
    # write
    w = open("a.txt", "w")
    w.write("de")
    w.close()
    # read
    r = open("a.txt", "r")
    for line in r:
        print(line)
    r.close()
main()
```

Tuples

```
t1 = () # Create an empty tuple
t2=(1,3,5) # Create a set with three elements
# Create a tuple from a list
t3 = tuple([2*x for x in
range(1,5)])
# Create a tuple from a string
t4 = tuple("abac") # t4 is ['a', 'b', 'a', 'c']
```

- Tuples vs. lists: you cannot modify a tuple!

List Comprehensions

- List comprehensions are a concise way to create lists

```
>> squares = [x**2 for x in range(10)]
>>> squares
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

same with:

```
>>> squares = []
>>> for x in range(10):
...       squares.append(x**2)
```

but shorter

List Comprehensions

```
>>> vec = [-4, -2, 0, 2, 4]
# create a new list with the values doubled
>>> [x*2 for x in vec]
[-8, -4, 0, 4, 8]
# filter the list to exclude negative numbers
>>> [x for x in vec if x >= 0]
[0, 2, 4]
# apply a function to all the elements
>>> [abs(x) for x in vec]
[4, 2, 0, 2, 4]
```

List Comprehensions

- A list comprehension consists of brackets containing an expression followed by a **for** clause, then zero or more **for** or **if** clauses
  - the result will be a new list resulting from evaluating the expression in the context of the **for** and **if** clauses which follow it
  - example: combines the elements of two lists if they are not equal

```
>>> [(x, y) for x in [1,2,3] for y in [3,1,4] if x != y]
[(1, 3), (1, 4), (2, 3), (2, 1), (2, 4), (3, 1), (3, 4)]
```

List Comprehensions

```
# create a list of 2-tuples like (number, square)
>>> [(x, x**2) for x in range(6)]
[(0, 0), (1, 1), (2, 4), (3, 9), (4, 16), (5, 25)]

# flatten a list using a listcomp with two 'for'
>>> vec = [[1,2,3], [4,5,6], [7,8,9]]
>>> [num for elem in vec for num in elem]
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

List Comprehensions

```
# Nested List Comprehensions
>>> matrix = [
...     [1, 2, 3, 4],
...     [5, 6, 7, 8],
...     [9, 10, 11, 12],
... ]
>>> [ [row[i] for row in matrix]
                 for i in range(len(matrix[0]))]
[[1, 5, 9], [2, 6, 10], [3, 7, 11], [4, 8, 12]]
```

Sets

```
# Create an empty set
s1 = set()

# Create a set with three elements
s2 = {1, 3, 5}

# Create a set from a list
s3 = set([1, 3, 5])

# Create a set from a list
s4 = set([x * 2 for x in range(1, 10)])

# Create a set from a string
s5 = set("abac") # s5 is {'a', 'b', 'c'}
```

Manipulating and Accessing Sets

```
>>> s1 = {1, 2, 4}
>>> s1.add(6)
>>> s1
{1, 2, 4, 6}
>>> len(s1)
4
>>> max(s1)
6
>>> min(s1)
1
>>> sum(s1)
13
>>> 3 in s1
False
>>> s1.remove(4)
>>> s1
{1, 2, 6}
>>>
```

Equality Test, Subset and Superset

```
>>> s1 = {1, 2, 4}
>>> s2 = {1, 4, 2}
>>> s1 == s2
True
>>> s1 != s2
False

>>> s1 = {1, 2, 4}
>>> s2 = {1, 4, 5, 2, 6}
>>> s1.issubset(s2) # s1 is a subset of s2
True
>>>

>>> s2.issuperset(s1) #s2 is a superset of s1
True
>>>
```

Comparison Operators

- Note that it makes no sense to compare the sets using the conventional comparison operators (>, >=, <=, <), because the elements in a set are not ordered.
- However, these operators have special meaning when used for sets.

s1 > s2 returns true is s1 is a proper superset of s2.

s1 >= s2 returns true is s1 is a superset of s2.

s1 < s2 returns true is s1 is a proper subset of s2.

s1 <= s2 returns true is s1 is a subset of s2.

Set Operations (union, |) (intersection, &) (difference, -)

```
>>> s1 = {1, 2, 4}
>>> s2 = {1, 3, 5}
>>> s1.union(s2)
{1, 2, 3, 4, 5}

# same with:
>>> s1 | s2
{1, 2, 3, 4, 5}

>>> s1 = {1, 2, 4}
>>> s2 = {1, 3, 5}
>>> s1.intersection(s2)
{1}

# same with:
>>> s1 & s2
{1}
```

```
>>> s1 = {1, 2, 4}
>>> s2 = {1, 3, 5}
>>>
s1.difference(s2)
{2, 4}

>>> s1 - s2
{2, 4}
```

Standard Library

- Operating System Interface:

```
>>> import os

# Return the current working directory
>>> os.getcwd()
'C:\\Python35'

# Run the command mkdir
>>> os.system('mkdir today')
0
```

Standard Library

- Operating System Interface:

```
>>> import shutil

>>> shutil.copyfile('data.db', 'archive.db')
'archive.db'

>>> shutil.move('/build/executables', 'installdir')
'installdir'
```

Standard Library

- Mathematics:

```
>>> import random
>>> random.choice(['apple', 'pear', 'banana'])
'apple'

# sampling without replacement
>>> random.sample(range(100), 10)
[30, 83, 16, 4, 8, 81, 41, 50, 18, 33]

>>> random.random()    # random float
0.17970987693706186
```

Standard Library

- Mathematics:

```
>>> import statistics

>>> data = [2.75, 1.75, 1.25, 0.25, 0.5, 1.25, 3.5]
>>> statistics.mean(data)
1.6071428571428572

>>> statistics.median(data)
1.25

>>> statistics.variance(data)
1.3720238095238095
```

Standard Library

- Internet Access:

```
>>> from urllib.request import urlopen


>>> with urlopen('http://www.cs.stonybrook.edu') as response:
    for line in response:
        print(line)
```

Standard Library

- Dates and Times:

```
>>> from datetime import date

>>> now = date.today()
>>> now

>>> birthday = date(2000, 5, 23)

>>> age = now - birthday

>>> age.days
```

Standard Library

- Data Compression:

```
>>> import zlib
>>> s = b'data archiving and compression’
# A prefix of 'b' means that the chars are encoded in byte type
# may only contain ASCII characters

>>> t = zlib.compress(s)
>>> zlib.decompress(t)
b'data archiving and compression'
```

Standard Library

- Testing:
  - *unittest: comprehensive set of tests to be maintained in a separate file*

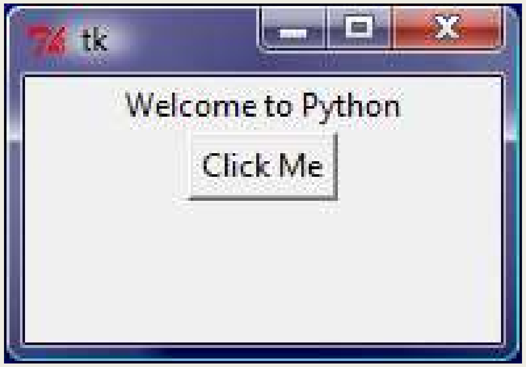
```
import unittest
class TestStatisticalFunctions(unittest.TestCase):
    def test_average(self):
        self.assertEqual(average([20, 30, 70]), 40.0)
        self.assertEqual(round(average([1, 5, 7]), 1), 4.3)
        with self.assertRaises(ZeroDivisionError):
            average([])
        with self.assertRaises(TypeError):
            average(20, 30, 70)

unittest.main() # Calling from the command line invokes all tests
```

What else?

- Lots:
  - *The Python Standard Library: built-in functions, collections, and many modules: https://docs.python.org/3/library/index.html#library-index*
  - *Installing Python Modules: pip, virtual environments https://docs.python.org/3/installing/index.html#installing-index*
  - *The Python Language Reference: the syntax and "core semantics"*

  https://docs.python.org/3/reference/index.html#reference-index

Python GUIs with tkinter



Welcome to Python

Click Me

```
from tkinter import * # Import tkinter

root = Tk() # Create a root window

# Create a label
label = Label(root, text = "Welcome to Python")

# Create a button
button = Button(root, text = "Click Me")

label.pack() # Display the label in the window
button.pack() # Display the button in the window

root.mainloop() # Create an event loop
```

Binary Search

```
# Use binary search to find the key in the
list
def binarySearch(lst, key):
    low = 0
    high = len(lst) - 1
    while high >= low:
        mid = (low + high) // 2
        if key < lst[mid]:
            high = mid - 1
        elif key == lst[mid]:
            return mid
        else:
            low = mid + 1
    # Now high < low, key not found
    return -low - 1
```

Selection Sort

```
def selectionSort(lst):
    for i in range(0, len(lst) - 1):
        # Find the minimum in the lst[i..len(lst)-1]
        currentMin = lst[i]
        currentMinIndex = i
        for j in range(i + 1, len(lst)):
            if currentMin > lst[j]:
                currentMin = lst[j]
                currentMinIndex = j
        # Swap lst[i] with lst[currentMinIndex] if necessary
        if currentMinIndex != i:
            lst[currentMinIndex] = lst[i]
            lst[i] = currentMin
    return lst
```

Standard Library

- String Pattern Matching Interface:

```
>>> import re

>>> re.findall(r’f[a-z]*',
    'which foot or hand fell fastest')

['foot', 'fell', 'fastest']
```

Lambda Expressions

- Small anonymous functions
  - a function can return a function

```
>>> def make_incrementor(n):
...     return lambda x: x + n
...
>>> f = make_incrementor(42)
>>> f(0)
42
>>> f(1)
43
```