SPIS Wednesday Lecture

Python coding (including some review)

- # comment
  - Inline : justifies why the code exists (intent)
  - Often an blank line before comment
  - One comment per idea
  - Don't describing how code works
  - Describe your intent or goal

- Displaying output:
  - print ('Hello World') # can take multiple strings separated by commas
    - ex: print ('Hello', 'Goodbye') #1
    - ex: print ('Hello' + 'Goodbye') #2

  A. Hello Goodbye B. HelloGoodbye C. Other

- Scalar Object Types (holds a single item):
  - int for whole numbers
  - float for real numbers

    - ex: 0.3 + 0.3 + 0.3

    A. 1 B. 1.0 C. 0.9 D. Other E. Error

  - bool for True or False
  - type (xyz) reports the type of xyz

- Non-Scalar Object Types:
  - str for text, known as "strings"

- …more we'll get to later

- Numeric operators:
  - + addition (overloaded for strings)
  - - subtraction
  - * multiplication (overloaded for strings)
  - // integer division
  - / float division

    - 11 divided by 2 gives 5.5 #1
    - 11 divided by 2 gives 5 #2

  A. Use // B. Use / C. Use % D. Other

  - % modulus (remainder of division)
  - ** power

- Augmented Assignment statements:
  - Shorthand code when updating an existing variable

  - abc += 3 #is the same as: abc = abc + 3
    - -=, *=, %=, ….

- Comparison operators (produces a bool result)
  - == equality
  - != inequality
  - < less than

- <= less than or equal to
- > greater than
- >= greater than or equal to

not typically with numbers

- is #are two objects really the same object
- is not #are two objects not the same object

- in is an item in a sequence
- not in is an item not in a sequence

print (“abc” in [“abc”,”def”,”ghi”])

print (“abc” not in [“abc”,”def”,”ghi”])

- Bool operators
  - and
  - or
  - not

Terminology (some review):

- Identifier (or symbol) – a name of a variable (or another entity … like a function, etc)
- scope – where symbols/identifiers/names are known
- block – a delimited grouping of lines of code that execute sequentially

- Python defines blocks by indenting

- Variables
  - = assignment: associates variable names with values
    - abc = 1
    - abc, bcd = 2, 3
    - abc, bcd, cde = 4, 5, 6
    - abc = bcd = cde = 10

  - Select names well (consider purpose)
    - Bad: i, x, y, temp
    - Better: index, result, sum

  - Case sensitive

    ```
    xyz = 10
    XYZ = 20
    print (xyz)        #1
    print (Xyz)        #2
    print (XYZ)        #3
    ```

    A. 10 B. 20 C. Other D. Error

  - Can contain letters, digits, _, (can't start with digit)
  - Can't be reserved words (keywords in language)
  - Typing by context

Functions (sometimes known as methods, procedures, or subroutines)

- What: A sequence of lines of code grouped as a unit
- Why: To encapsulate a functionality or task into a unit to be performed repeatedly when needed
- Convention: Typically, functions are silent.
    - "main" is the boss…the first function that starts program
    - Catastrophic situation are exceptions
- Avoid: Code duplication
- Ideals:
    - "Single Responsibility Principle"
        - A function should be responsible for performing one and only one task
    - "Separation of Concerns"
        - The lines of code in a function should be at the same level of abstraction.
            - Lower level ideas should be implemented by calling another function.
    - Shouldn't be too long
        - Lengthy functions can be broken into smaller functions.
- More Terminology:
    - Function definition – Python syntax to define a function (**def** keyword, name, parameter list, colon, code)
        - Tells Python about your function so it can execute in the future (when called)
    - Function body – code in the function definition
    - Function call – line of code to execute function

- Caller – the code that calls your function
- Result – value returned (sent back) from function
- Parameters – inputs to your function (aka arguments)
- Literal – a value that's not a variable
- Side effect – tasks performed that have an detectable effect other than returning a value

- Docstring – First line in function with double quote triplet:
  - Ex:

```
def function ():
    """ This function adds two values """
    print (1 + 2)
```

  - function.__doc__
    - produces Docstring as output

- How to use a function:
  1. **Define** the function, then
  2. **Call** (or execute) the function when needed
  3. Execution resumes after function call completes.
- Attributes:
  - Is named for task the code accomplishes
  - Has zero or one result produced
    - No result – task performed only
    - One result – result returned to caller
      - Caller wants result
        - Typically saved in a variable
        - Example:

```
result = function ()
```

        - Or in a conditional statement
        - Example:

```
if function () == 10:
        print ("do something")
```

- Has zero or more parameters (aka arguments) in parenthesis, separated by commas
  - Input parameters:
    - Information needed for function to perform its job
    - Provides flexibility/variability
      - Different inputs mean different outputs
- Body (the code, itself) is indented
- Ends with line of lesser indent
- Defines a “scope”
  - Parameters and variables are known by name only within the function body

---

Get variable values from the user using “input” function

```
xyz = int (input (“Enter an integer:  “))
print (xyz)
abc = float (input (“Enter a float:  “))
print (abc)
```

Local and Global variables

- Variables are local unless declared global

```
zzz = 10
```

```
def afunction():
    global zzz
    print (zzz)
    zzz = 9
print(zzz)
afunction()
print(zzz)
# What is the output?
```

```
zzz = 10
def afunction():
    zzz = 9
print(zzz)
afunction()
print(zzz)
# What is the output?
```

```
zzz = 10
def afunction():
    print (zzz)
    zzz = 9
print(zzz)
afunction()
print(zzz)
```

What is the output?

Collections (like an array)

- Lists
  - Ordered
  - Changeable
  - Use brackets
  - Duplicates are allowed
- Tuples
  - Ordered
  - Unchangeable
  - Use parenthesis
  - Duplicates are allowed
- Sets
  - Unordered
  - Unindexed
  - Can’t change items, but you can add items
  - Use curly braces
  - Duplicates are not allowed

mylist = [“abc”, “def”, “ghi”]

mytuple = (“abc”, “def”, “ghi”)

myset = { “abc”, “def”, “ghi” }

first item is found at index 0

- A sequence of items (ints, floats, strs, bools, …)

instructors = [“Gary”, “Curt”, “Niema”]

print (len (instructors)) # displays _____

print (instructors) # displays _____

```
print (instructors[1])  # displays _____________
instructors.append (“Mohan”)
print (instructors)
instructors.remove (“Gary”)
print (instructors)
instructors.sort ()
print (instructors)
instructors.reverse ()
print (instructors)
```

```
instructors = [ “Gary”, “Curt”, “Niema” ]
instructors_again = [ “Gary”, “Curt”, “Niema” ]
good_instructors = instructors      # what is going on here?

# How many list variables are there?
# How many lists are there?
print (instructors == good_instructors)    # Result is _________
print (instructors != good_instructors)    # Result is  _________
print (instructors is good_instructors)    # Result is  _________
```

```
print (instructors is not good_instructors)    # Result is_______
print (instructors == instructors_again)    # Result is _________
print (instructors != instructors_again)    # Result is  _________
print (instructors is instructors_again)    # Result is  _________
print (instructors is not instructors_again)    # Result is_______

instructors.clear ()       # empties the list
```

Functions and Methods in Python

Functions are not called on objects (object isn't needed or used):

```
print ("Hello World")
```

Methods are called on objects (action on a specific object):

```
# remove "Gary" from specific instructors list:
instructors.remove ("Gary")
```

Calling functions or methods depends on how function or method is defined.

Use provided code found in code libraries:

print a random number between 1 and 9:

import random

print (random.randrange(1,10))

- "if" statements:
  - Allows conditional behavior
  - …take either one code path or another
  - "else" is optional
  - "elif" is optional ("else if")

- "if"statement examples:

```
abc = 2
if abc == 2:
        print ("abc is 2")
```

```
abc = 1
if abc == 2:
```

```
        print (“abc is 2”)
    else:
        print (“abc is not 2”)
```

```
    abc = 1
    if abc == 2:
        print (“abc is 2”)
    elif abc == 3:
        print (“abc is 3”)
    else:
        print (“abc is not 2 or 3”)
```

```
day = “Wed”
time = “After 10:15am”
if day == “Wed” and time == “After 10:15am”:
    print (“I am in CSE 2154”)
```

```
    if abc == 10:
        if xyz == 20:
            print (“abc is 10 and xyz is 20”)
        else
            print (“abc is not 10”)
# what happens if the else indent changes?
```