Python overview, continued

- Boolean data type/conditional statements, revisited
- executing a function, revisited
- variable scope
- midterm review
- midterm (Friday)

```python
def trackscore(score):
    '''convert score to grade point; return grade point'''
    gradepoint = 0
    if score >= 90:
        gradepoint = 4
    elif score >= 80:
        gradepoint = 3
    elif score >= 70:
        gradepoint = 2
    elif score >= 60:
        gradepoint = 1
    return gradepoint
```

What is the result of executing the following code:

```python
gradepoint = 4
>>> gp = trackscore(90)
>>> gp
```

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>a) <code>class 'bool'</code></td>
</tr>
<tr>
<td>'True'</td>
<td>b) <code>class 'str'</code></td>
</tr>
<tr>
<td>true</td>
<td>c) NameError: name is not defined</td>
</tr>
<tr>
<td>'false'</td>
<td></td>
</tr>
</tbody>
</table>
Boolean expressions

**lazy evaluation**

| a = 99 |
| b = 88 |
| c = -1 |

if (a < 0) and (b/c < 0):

if (a > 0) or (b/c > 0):

**lazy evaluation - consequences**

| a = -99 |
| b = 88 |
| c = 0 |

if (a < 0) and (b/c < 0):

if (a > 0) or (b/c > 0):

**Boolean expressions - style**

if (a < 0) == True:

if a < 0:
Boolean expressions - *style*

```python
if (a < 0) == True:
    if a < 0:
        if isInCircle(.5, .5, 1) == True:
            if isInCircle(.5, .5, 1)
```

---

```python
def montePi(numDarts):
    '''
    
    inCircleCt = 0
    for i in range(numDarts):
        x = random.random()
        y = random.random()
        result = isInCircle(x, y, 1)
        if result == True:
            inCircleCt += 1
    approxPi = inCircleCt / numDarts * 4
    return approxPi
    '''
```

---

Boolean data type *(is trickier than you might think for a data type that has only two values)*

- Boolean operations on Boolean values only
- Order of operations (use parens for clarity)
- Booleans are not strings
- Boolean short circuit ("lazy") evaluation can lead to hard-to-find errors
- Good style for Boolean expressions
When a function is called/executed, Python:

1. evaluates each argument one at a time, working from left to right
2. assigns the resulting values to the function parameters
3. creates a space (activation record/stack frame) to keep track of function execution, including local namespace; add to call stack
4. executes the function until return statement
5. stops function execution and returns value specified in return statement
6. the activation record is (eventually) destroyed
7. processing resumes where the function was called

```python
def twice(x):
    
    y = 2
    result = y * x
    return result
```

```bash
>>> y = 5
>>> twice(y)
>>> y  >>> x  >>> twice
```

```python
# global namespace - func def
```

```python
# global namespace - y
```

```python
# local namespace - func exec
```

```bash
>>> y  >>> x  >>> twice  # global again
```
def twice(x):
    y = 2; print(y)  # y is local to twice
    result = y * x  # when twice is executing
    return result

>>> y = 5  # global namespace
>>> twice(y)  # local namespace – func exec
>>> y  >>> x  >>> twice  # global again

def twice(x):
    x += 1  # x is local to twice
    m = 2
    result = m * x
    return result

>>> x = 5  # x in global namespace
>>> twice(x)
>>> x  >>> m

def test1(a):
    a += 5
    return a

>>> a = 6
>>> test1(a)  >>> a
??  ??
### Namespaces:

- **local** (function execution)
- **global** (module)
- **builtin** (Python)

### Code Snippet

```python
def test1(a):
    def test2(b):
        a += 5
        print(a)
        return None

>>> a = 6
>>> test2(14)
???
```

---

### Programming/Computer Science concepts

- Computational Problem Solving: designing, implementing, checking, revising algorithms/programs.
- Good programming style: function docstrings (type contract; description including parameters, returned value, and side effects if any; examples of function use), well-named variables, use of whitespace between operators and sections of code, judicious use of inline comments (why not what).
- Python primitive elements: Objects - value/attributes, operations, memory location (id).
- Combining primitive elements: Expressions - expressions evaluate to a value.
- Naming values: Variables/assignment - assignment statements are not expressions and do not return a value; namespace, variable scope.
- User-defined functions – functions always return a value (sometimes None); functions sometimes have side effects.

### What happens when a function is called?

- Activation record/frame added to call stack; local namespace
- Call-by-assignment parameter passing
- Side effects/returning a value
- Iterative algorithms; accumulator pattern
- Monte Carlo algorithms
- Cipher encrypting and decrypting