Recall: Python keeps track of variables using namespaces - directories of names and objects.

When we start Python, two namespaces are created – the built-in namespace and the global (__main__) namespace.

When we create names (e.g., variables, function definitions) in a Python session, they are added to __main__.

Scope refers to the visibility of variables: scope refers to a region of a program where a variable (namespace) can be directly accessed, i.e., without using a namespace prefix.

def twice(x):
    #parameters are local
    y = 2
    result = y * x
    print(dir())
    return result

>>> y = 5
>>> y
5
>>> dir()
['result', 'x', 'y']
>>> twice(y)
5
>>> x
#local scope only
10
NameError
def twice(x):
    result = y * x  # no local y ->
    return result   # find global y
    # do this sparingly!!

>>> y = 5
>>> twice(y)
??

>>> y  >>> x  >>> twice  # global again

Python searches namespaces in this order:
Local, then
Global, then
Built-in

>>> round(12.34)
??

>>> round = 567
>>> round(12.34)  >>> def = 890
??  ??

def thrice(x):
    x += 1
    m = 3
    return m * x

>>> x = 5
>>> x

>>> thrice(x)  ??
??  >>> m
??

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

>>> a = 6
>>> a = test1(a)  >>> a
??  ??

def test1(a):
    a += 5
    print(a)
    return None

def test2(b):
    a += 5
    print(b)
    return None
def test3(b):
    def test4(a):
        test4(b)
        a += 5
        print(a, b)
        return None
    return b + 1
print(a, b)
return None

>>> test3(99)
NameError: name 'b' is not defined
>>> b = 1
>>> test3(99)
local namespaces are on the same level -
104 1
100 static (lexical) scoping

Boolean Expressions/Conditional Statements

if <boolean expression>:
    <block of code>
    <next Python statement>

Flow of control

if <boolean expression>:
    <block of code>
elif <boolean expression>:
    <block of code>
elif <boolean expression>:
    <block of code>
else:
    <block of code>
<next Python statement>

Boolean expressions

logical/relational operators

return a Boolean value

True
False

Boolean expressions: relational and logical operators return Boolean values

a < b
a <= b
a > b
a >= b
a == b
a != b

not a < b
a <= b and c >= d
a <= b or c >= d
#use logical operators with
#Boolean values only!
### Boolean expressions

<table>
<thead>
<tr>
<th>Expression</th>
<th>Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>a &lt; b</code></td>
<td>not a &lt; b</td>
</tr>
<tr>
<td><code>a &lt;= b</code></td>
<td>(a &lt;= b) and (c &gt;= d)</td>
</tr>
<tr>
<td><code>a &gt; b</code></td>
<td>(a &lt;= b) or (c &gt;= d)</td>
</tr>
<tr>
<td><code>a &lt;= b</code></td>
<td>order of precedence:</td>
</tr>
<tr>
<td><code>a == b</code></td>
<td>relational operators &gt; logical</td>
</tr>
<tr>
<td><code>a != b</code></td>
<td>not &gt; and &gt; or – PARENS BEST</td>
</tr>
</tbody>
</table>

**order of precedence:**
- `not` > `and` > `or` - PARENTHESIZE BEST

#### Type Checking

```python
>>> type(True)
a) <class 'bool'>
>>> type('True')
b) <class 'str'>
>>> type(true)
c) NameError: name is not defined
```

### Short Circuit Evaluation

```python
a = 99
b = 88

if (a < 0) and (b < 0):
    print('hello')

if (a > 0) or (b > 0):
    print('hello')
```

#### Function Definition

```python
def temp_alert(temp):
    '''(number) -> None
    print information about the temperature
    '''
    if temp >= 90:
        print('hot')
    if temp >= 80:
        print('very warm')
    if temp >= 70:
        print('warm')
    if temp >= 60:
        print('cool')
    return None
```

What is the result of executing `print('hot')` the following code:

```python
>>> temp_alert(90)
??
```
```python
def temp_alert(temp):
    '''(number) -> None
    print information about the temperature'''
    if temp >= 90:
        print('hot')
        elif temp >= 80:
            print('very warm')
            elif temp >= 70:
                print('warm')
                elif temp >= 60:
                    print('cool')
    return None
```

What is the result of executing `print('hot')` the following code:

```python
>>> temp_alert(90)
```

```python
CIS 210
Nifty Python - style
```a = -99
b = 88
c = 77
if a < b and b < c:
    if (a < b) and (b < c):
        print('most languages')
        print('better')
if a < b < c:
    print('Python allows chaining of relational operators ')
```

```python
CIS 210
Boolean expressions - style
``````python
>>> isinstance(101, int)
True
>>> isinstance(101, str)
False
```if isinstance(101, int) == True:
    if isinstance(101, int):
        if isinstance:
            <will this code be executed??>
```

```python
CIS 210
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 evaluation can lead to hard-to-find errors
• Good style for Boolean expressions
• Double (triple) check Boolean expressions

```python
CIS 210
Monte Carlo Algorithms
```• Statistical simulation methods – use sequences of random numbers to perform a simulation
• Any method which solves a problem by generating random numbers and observing that fraction of the numbers obeying some property or properties
• Example of a heuristic technique – guesstimate, approximation - useful when difficult, impossible, or inefficient to use other, more exact, methods

```python
CIS 210
```
Monte Carlo Simulation to Approximate Pi

- Simulate a game of darts
- Randomly place darts on the board
- Value of pi can be computed by keeping track of the number of darts that land on the board

Monte Carlo Simulation (Problem 3-2)

- the area of the circle is $\pi/4$ and area of square is 1
- the fraction of darts that lands in the circle is $(\pi/4)/1 = \pi/4$
- the fraction of darts that lands in the circle is $\frac{\text{inCircleCt}}{\text{numDarts}}$
- $\frac{\text{inCircleCt}}{\text{numDarts}} = \frac{\pi}{4} \Rightarrow \pi = 4 \times \left(\frac{\text{inCircleCt}}{\text{numDarts}}\right)$

- to determine whether a dart has landed in the circle – use formula for finding the distance between the point and the origin: $d = \sqrt{x^2 + y^2}$
- how do we throw darts at the board??
Monte Carlo Simulation (Problem 3-2)

Figure 2.10

Monte Carlo Simulation (Problem 3-2)

• the area of the circle is π/4 and area of square is 1
• the fraction of darts that lands in the circle is (π/4) / 1 = π/4
• the fraction of darts that lands in the circle is (inCircleCt / numDarts)
• inCircleCt / numDarts = π/4 * (inCircleCt / numDarts)
• to determine whether a dart has landed in the circle – use formula for finding the
distance between the point and the (0, 0) origin:
d = math.sqrt(x**2 + y**2)

• throw darts - generate x and y = random.random()

>>> help(random.random)
Help on built-in function random:  random(...)
random() - x in the interval [0, 1).

Monte Carlo Simulation (Problem 3-2)

import random
import math
def montePi(numDarts):
    '''
    inCircle = 0
    for i in range(numDarts):
        x = random.random()
        y = random.random()
        d = math.sqrt(x**2 + y**2)
        if d <= 1:
            inCircle += 1
    pi = inCircle / numDarts * 4
    return pi

Monte Carlo Simulation (Problem 3-2 and 3-3)

(0) type in the montePi function from the text; add docstrings per CIS 210 style
guidelines

(1) revise montePi so that it calls a new isInCircle function (exercises 2.38 and 2.39)

(2) write new function, drawBoard, to draw the “dartboard” for the graphical output

(3) add docstrings to showMontePi starter code (montePi + visualization)

(4) revise showMontePi so that it calls the new isInCircle and drawboard functions

(5) write new function, reportPi, which will be called from showMontePi, to compare
the approximate value of pi generated by the Monte Carlo method to the value of
math.pi, and report on any error in the approximation

(6) modify showMontePi to call reportPi
Strings are sequences of characters.

**Operators**
- Concatenation +
- Repetition *
- Indexing [
- Slicing [:]

**Methods**
- upper
- lower
- center
- count
- index
- find
- replace

**Note:** "overloaded operators"

```python
>>> 99 + 100
200
```

```python
>>> 'hello' + 'goodbye'
'hello' + 'goodbye'
```

```python
>>> 'hello' * 4
'hellohellohellohello'
```

```python
>>> x = 'PYTHON ROCKS'
```

```python
>>> len(x)
13
```

```python
>>> x[4]
H
```

```python
>>> x[-1]
S
```

```python
>>> x[1:4]
THON
```

```python
>>> x * 3
'PYTHON ROCKS' 'PYTHON ROCKS' 'PYTHON ROCKS'
```

```python
>>> x + 'yes'
'PYTHON ROCKSyes'
```

```python
>>> str.find(x, 'O')
4
```
>>> x = 'PYTHON ROCKS'

find method, for example:

>>> str.find(x, 'O')
4

>>> x.find('O')  >>> 'PYTHON ROCKS'.find('O')
4 4

they are all methods

---

operators:  +, -, *, /, //, %, <, !=, and, in, [], [:], ...

functions:  len, round, abs, range, ...

methods:  str.replace, str.index, ...

---

for is a sequential operator

x = "We can't stop for gas, we're already late."

>>> for ch in x:
    print(ch)

W e

c a

---

for is a sequential operator

>>> o_ctr = 0
>>> for letter in 'hello':
    if letter == 'o':
        o_ctr += 1
strings are immutable sequences

```python
>>> s = 'jello, world'
>>> s[0] = 'h'
TypeError: 'str' object does not support item assignment
```

```python
>>> s = 'hello, world'
>>> s = 'h' + s[1:]
```

while loop

indefinite iteration

most general type of loop

```python
while <boolean expression>:
    statement1
    statement2
    ...
    statement
```

```python
p = 10
i = 1
ctr = 1  # initialize loop variable
while ctr <= p:  # check end condition
    i = i * 2
    ctr += 1  # move loop var toward
print(i)  # the end condition
```
CIS 210

while loop

```python
p = 10
i = 1
ctr = 1  # loop var
for ctr in range(p):
    while ctr <= p:  # check
        i = i * 2
        ctr = ctr + 1  # advance
print(i)
```

can’t specify ahead of time how many times the loop will run

---

CIS 210

Python toolkit so far

- numeric data types (int, float) and operations (e.g., +, *, round, abs)
- string data type and operations (e.g., +, len, count, find, format)
- Boolean data type and operations (e.g., +, and)
- NoneType (None)
- print
- expressions
- Python Standard Library – math, turtle, random modules; import
- assignment statement
- Python repetition – for, while
- Python conditionals – if
- variable assignment
- user-defined functions; function design; docstrings
- IDLE interactive development environment; help function

---

CIS 210

while loop

```python
print('Enter numbers you wish to add, quit to end.')
sum = 0
next = input('next: ')
while next != 'quit':
    sum += int(next)
    next = input('next: ')
print('Sum is:', sum)
```

can’t specify ahead of time how many times the loop will run

---

CIS 210

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 is a programming language and Python is an interpreter (program)
- Python Shell is a REPL (read-evaluate-print loop)
- Python primitive elements: Objects - value/attributes, type
- Combining primitive elements: Expressions - expressions evaluate to a value; short circuit evaluation of boolean expressions
- Naming values: Variables/assignment - assignment statements are not expressions and do not return a value; namespaces – builtins and global (__main__ scope).
- Functions are an executable data type; what happens when a function is called:
  - Activation record/stack frame added to call stack for local namespace; return address
  - Call-by-assignment parameter passing
  - Functions always return a value (sometimes None)
  - Functions sometimes have side effects
- Iterative algorithms; accumulator pattern

---

CIS 210

CIS 210 Learning Outcomes

- understand, develop, implement algorithms for computational problem solving;
- use structured design and testing methods to develop and implement programs;
- read, write, revise, document, test, and debug code;
- demonstrate robust mental models of data representation and code execution;
- demonstrate good understanding of a high level programming language;
- introduce and/or implement a sampling of classic computer science problem domains and algorithms.