namespaces/executing functions
→ variable scope
• Boolean data type/conditionals
• Monte Carlo algorithm for approximating pi
• strings – sequences, immutable data types
• indefinite iteration

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.

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

>>> y = 5
>>> y
5
>>> twice(y)
10
```

```python
def twice(x):
    y = 2  # y is a “local variable”
    result = y * x  # so is result
    print(dir())  # dir() returns current scope:
    return result

>>> y = 5
>>> y  # back in global scope
5
>>> twice(y)
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

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
>>> test1(a)

>>> a

>>> b
??

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

>>> test3(99)

>>> b = 1
>>> test3(99)
??

>>> b
??
def test3(b):
    def test4(a):
        test4(b)
        a += 5
        return a + b
    return None

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

Boolean Expressions/Conditional Statements
if <boolean expression>:
    <block of code>

    <next Python statement>

Boolean expressions
logical/relational operators
return a Boolean value
True
False

✓ namespaces → variable scope
- Boolean data type/conditionals
- Monte Carlo algorithm for approximating pi
- strings – sequences, immutable data types
- indefinite iteration

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>

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

- $a < b$
- $a \leq b$
- $a > b$
- $a \leq b$
- $a == b$
- $a != b$

**Order of precedence:**
- relational operators
  > logical
  > logical not
  > and
  > or

---

**Short circuit evaluation**

```python
a = 99
b = 88

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

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

---

**Exploring**

```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

>>> temp_alert(90)
hot
```

---

```python
>>> def = 123
??

>>> abs = 456
??

>>> abs(-7)
??
```
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; overloaded operators
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 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.

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