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

Programming/Computer Science concepts
Computational Problem Solving: designing, implementing, checking, revising algorithms/pogs.
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

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

CIS 210

VARIABLE SCOPE

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 a global (__main__) namespace.

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

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

Recall: When Python executes a function, a local namespace is created to keep track of function variables.

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.
CIS 210
VARIABLE SCOPE

def twice(x):
    """
    Scope refers to a region of the program; the scope of a variable is the region of the program where the variable is visible
    """
    y = 2
    result = y * x
    #print(dir())
    return result

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

CIS 210
VARIABLE SCOPE

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

>>> y = 5
>>> twice(y)
5
[\'result\', \'x\', \'y\']
>>> x
#local scope only
10
NameError

CIS 210
SCOPE

def twice(x):
    """
    result = y * x
    return result
    """

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

CIS 210
SCOPE

Python searches namespaces in this order:

Local, then
Global, then
Built-in

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

CIS 210
SCOPE

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

>>> x = 5
>>> twice(x)
??

CIS 210
SCOPE

>>> x = 5
>>> m
??
```python
def test1(a):
    a += 5
    return a

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

```
Boolean expressions

logical/relational operators

return a Boolean value

True
False

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 logical operators with
a != b       #Boolean values only!

Boolean expressions

a < b        not a < b
a <= b       (a <= b) and (c >= d)
a > b        (a <= b) or (c >= d)
a <= b
a == b       #order of precedence:
a != b       relational operators > logical
             not > and > or – PARENS BEST

Boolean expressions

short circuit evaluation

a = 99
b = 88

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

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

>>> type(True)       a) <class 'bool'>
>>> type('True')      b) <class 'str'>
>>> type(true)        c) NameError: name
>>> type(False)       is not defined
>>> type(False)
>>> type(false)
Boolean expressions
short circuit evaluation

a = -99
b = 88

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

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

CIS 210
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:

>>> temp_alert(90)
print('very warm')
??

>>> temp_alert(90)
print('cool')
return None