ALL ARE WELCOME!
NO EXPERIENCE NECESSARY

4-5 & 5-6 PM
Informational Meetings
Tuesday, October 22
@ the Price Science Commons Room B040

CIS 210

code review
– drop in
– lab help hours (B004/A)
– good faith effort grade
– 2 per term (wks 3-6/7-10)
– discuss submitted code

CIS 210

code demo
– scheduled appointment
– 139 Deschutes
– grade per project rubric
– 1 per term (wks 4-10)
– computational problem solving: spec -> writing code (whiteboard)

CIS 210

Exploring

>>> def = 123
a) assignment
??

>>> abs = 456
b) expression is evaluated
??

>>> abs(-7)
c) error
??

CIS 210

while loop

most general type of loop

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

• for v. while loops
• namespaces/what happens when a function is executed
  → variable scope
• Boolean data type/conditionals
• Monte Carlo algorithm for approximating pi
• strings – sequences, immutable data types
### CIS 210
#### while loop

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

---

### CIS 210
#### while loop

```python
p = 10
d = 1
cnt = 1  # loop var
for cnt in range(p):
    d = d * 2  # check
    cnt += 1  # advance
    print(d)  # the end condition
```

---

### CIS 210
#### while loop

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

---

### CIS 210
#### VARIABLE SCOPE

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

---

### 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 *global* (`__main__`) namespace.
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 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 can be directly accessed, i.e., without using a namespace prefix.

Local variables have local scope - for example, function variables - function parameters or variables defined in the body of the function.

Global variables have global scope - for example, variables defined outside of all functions, function name

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

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

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

>>> y = 5
>>> y  #back in global scope
5
>>> twice(y)
5
['result', 'x', 'y']
>>> 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
    '''

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

In most cases where you are tempted to use a global variable, it is better to use a parameter for getting a value into a function or return a value to get it out.

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
    c) 15
    >>> thrice(x)
    ??
    >>> m
    e) error
    ??
    >>> a = 6
    >>> a = test1(a)
    >>> a
    e) error
    ??
    def test1(a):
        '''
        a += 5
        return a
        a) 5
        b) 6
        c) 10
        d) 11
        a) 5
        b) 6
        c) 10
        d) 11
        e) error
        >>> a = test1(a)
        >>> a
        e) error
        ??
        >>> a
        e) error
        ??
        def twice(x):
            '''
            result = y * x  # no local y ->
            return result  # find global y
            # do this sparingly!!
            -- function independence/reusability
            -- readability of code
            >>> y = 5
            >>> twice(y)
            ??
            >>> y
            >>> x
            >>> twice
            # global again
```python
def test1(a):
    def test2(b):
        a += 5
        print(b)
        print(a)
        return None

>>> a = 6
>>> test1(14)
>>> test2(14)
>>> a
>>> b

>>> a = 6
>>> test1(14)
>>> test2(14)
>>> a
>>> b

>>> 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

✓ for v. while loops
✓ namespaces ➔ variable scope
• Boolean data type/conditionals
• Monte Carlo algorithm for approximating pi
• strings – sequences, immutable data types

Boolean Expressions/Conditional Statements
if <boolean expression>:
    <block of code> may execute or not
<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>
```

25

26

27

28

29

30
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 (only)
a != b  #to generate Boolean values

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')

Boolean expressions

short circuit evaluation ... can cause hard-to-find bugs

a = -1
b = 88

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

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')

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')
CIS 210
Exploring

```python
>>> s = 'abc'
>>> if len(s) == 0 or 1:
    print('yes')
??
```

```python
>>> s = ''
>>> if len(s) == 1 or 0:
    print('yes')
??
```

CIS 210
Exploring

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

```python
if temp >= 80:
    return 'very warm'
```
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

# Example usage
temp_alert(90)

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 ')

Boolean expressions - style

if isinstance(101, int) == True:
    if isinstance(101, int):
        if isinstance(101, int):
            if isinstance(x, int) == True:
                if isinstance(x, int):
                    if isinstance(x, int):
Boolean expressions - style

```python
if isinstance(x, int) == 'True':

if isinstance(x, int) == True:

if isinstance(x, int):

if isinstance:
    <will this code be executed??>
```

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

✓ for v. while loops
✓ namespaces → variable scope
✓ Boolean data type/conditionals
  - Monte Carlo algorithm for approximating pi
  - strings – sequences, immutable data types

Monte Carlo Algorithms

- A Monte Carlo method is a fairly simple way to get an answer to a task without having to analyze it mathematically. The solution is to simulate the task and see what happens. And this is best done with a computer program.

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

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 $\frac{\text{inCircleCt}}{\text{numDarts}}$

- $\frac{\text{inCircleCt}}{\text{numDarts}} = \pi/4 \Rightarrow \pi = 4 \times \left(\frac{\text{inCircleCt}}{\text{numDarts}}\right)$

- 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)

- 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).

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

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

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

1) revise montePi so that it calls a new isInCircle function

2) write another new function, reportPi, which will be called from montePi, 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

3) add docstring to showMontePi starter code (original montePi + visualization)

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

5) revise showMontePi so that it calls the new isInCircle, reportPi, and drawBoard functions; add visualization code

CIS 210

✓ for v. while loops
✓ namespaces → variable scope
✓ Boolean data type/conditionals
✓ Monte Carlo algorithm for approximating pi
  • strings – sequences, immutable data types, overloaded operators, methods, for
Strings are sequences of characters.

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

Methods
upper
lower
center
count
find
replace
startswith

Note: "overloaded operators"

>>> 99 + 100
>>> 2 * 10
??

>>> 'hello' + 'goodbye'
>>> 'hello' * 4
??

>>> x = 'PYTHON ROCKS'

find method, for example:

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

>>> x = 'PYTHON ROCKS'

>>> x[4]
>>> x[-1]
??
??

>>> x[1:4]
>>> x * 3
??
??

>>> x + 'yes'
??

>>> len(x)

>>> x[4:]

>>> x[-1:]

>>> x[1:4]

>>> x * 3

>>> x + 'yes'

>>> str.find(x, 'O')
4
CIS 210

STRINGS

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

find method, for example:

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

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

CIS 210

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

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

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

they are all methods

CIS 210 (FYI only)

```python
>>> str.__(a, 'b')
'ab'
>>> 'a'.__(b)
'ab'
```

```python
>>> int.__(4, 3)
# 4 + 3
7
```

```python
>>> (4).__(3)
# why parens or space?
7
```

```python
```

CIS 210

for is a sequential operator

```python
x = "We can't stop for gas, we're already late."
>>> for ch in x:
    print(ch)
```

```python
W
```

```python
e
c
```

CIS 210

for is a sequential operator

```python
>>> o_ctr = 0
>>> for letter in 'hello':
    if letter == 'o':
        o_ctr += 1
```

```python
77
```
strings are immutable sequences

```python
>>> s = 'jello, world'
>>> s[0] = 'h'
??
```

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

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

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

---

A Structured Approach to Computational Problem-Solving

- review the project specification thoroughly
- write examples of expected results for specified inputs – re-review spec, if needed
- develop, review, and/or revise a problem-solving approach, using natural language, pseudocode (not Python code)
- check algorithm using your examples – revise algorithm, re-review spec, if needed

Starting with the lowest level function -
- write the function header
- write the function docstring – type contract
- write the function docstring – brief description
- write the function docstring – examples of use (use ones developed earlier)
- write the return statement

- using tools from the Python toolkit, start writing the body of the function
- test often, revise as needed
- test using examples in the docstring, and then project spec, and then others

---

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)
- 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 (selection) – 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/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 – method – 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; Monte Carlo algorithms

Turing completeness

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.