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

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

**Midterm**: Thursday Feb. 6th

**Practice Midterm in class next Tuesday** (Jan 28th)
iClicker Question: which option from a) - d) would produce an error when used for the for-loop?

```
import math

def mysqrt(itg, iters):
    x = 1
    for...
        x = .5 * (x + itg/x)
        #print(x)

    return x
```
Exploring

```python
>>> def = 123
??
a) assignment

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

>>> abs(-7)
??
```
>>> pi = 3.14159
>>> radius = 8.0

a) assignment
b) expression is evaluated
c) error

>>> area = pi * radius ** 2
??
Today’s Agenda:

- 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
while loop

most general type of loop

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

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

```python
p = 10
i = 1

while ctr <= p:
    i = i * 2
    ctr = ctr + 1

print(i)
```

for (repeat) loops are preferred for **definite** iteration
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)

A while loop is needed when we can’t specify ahead of time how many times the loop will run (indefinite iteration)
✓ for v. while loops
  • namespaces/executing functions
    → variable scope
  • Boolean data type/conditionals
  • Monte Carlo algorithm for approximating pi
  • strings – sequences, immutable data types
Recall: Python keeps track of variables using *namespaces* - directories of names and objects.
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.
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__**.
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.
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.
def twice(x):
    y = 2
    result = y * x
    print(dir())
    return result

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

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
??

>>> dir()
??

>>> x
??
def twice(x):
    # parameters are local

    y = 2  # y is a “local variable”
    result = y * x  # so is result

    print(dir())  # dir() returns current scope:

    return result

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

10  # local scope only
NameError
def twice(x):
    result = y * x
    return result

>>> y = 5
>>> twice(y)
??
>>> y >>> x >>> twice  # global again
def twice(x):
    result = y * x  # no local y ->
    return result   # find global y
    # do this sparingly!

-- function independence/reusability
-- readability of code
def twice(x):
    result = y * x  # no local y ->
    return result   # find global y
    # do this sparingly!!

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
a) 5

>>> thrice(x)
b) 6
c) 15

??
d) 18

??
e) error
??
def test1(a):
    
    a += 5
    return a

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

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

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

a) 11  b) 6  c) 14  d) error
def test2(b):
    """
    b += 3
    return None

>>> b = 6
>>> b = test2(b)
>>> b + 2

a) 11    b)15    c) 8    d) error
def test3(b):
    test4(b)
    return b + 1

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

>>> test3(99)
??

>>> b = 1

>>> test3(99)
??

>>> b
??

>>> b
??
def test3(b):
    test4(b)
    return b + 1

def test4(a):
    a += 5
    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
CIS 210

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

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

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

Boolean expressions

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

Try a = -1 and a = 1
b = 88

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

if (a > 0) or (b / 0 > 0):
    print('hello')
Boolean Expressions/Conditional Statements

if <boolean expression>:  
    <block of code>  
    
    may execute or not

<next Python statement>

A Boolean data type is one that has TWO possible values (True or False)
BOOLEAN DATA TYPES:

Relational operators: $<$, $<=$, $!=$ etc.

Logical operators: and, or, not

Boolean expressions:

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

Compound Boolean expressions:

- $not a < b$
- $(a <= b) and (c >= d)$
- $(a <= b) or (c >= d)$

When unsure of precedence, ADD PARENTHESES! Python reads left to right, short circuits in compound expressions, see next slide.
Boolean expressions

relational expressions:

\[ 6 \leq 10 \quad \text{(True)} \]
\[ 4 \neq 4 \quad \text{(False)} \]
Compound Boolean expressions: AND

expression 1 and expression 2
(if expression 1 is False, return expression 1; otherwise return expression 2)

example: 2 != 2 and 12 < 20  (False)

example: 2 != 2 and 12 > 20  (False)

example: 3 == 3 and 4 < 3  (False)

example: 3 == 3 and 4 > 3  (True)
Compound Boolean expressions: OR

expression 1 or expression 2

(if expression 1 is False, return expression 2; otherwise return expression 1)

eample: 2 != 2 or 12 < 20  (True)

eample: 2 == 0 or 1  (1)

eample: 3 == 3 or 0  (True)

eample: 3 == 3 or 0  (True)

**Any Python expression can be used as part of a Boolean expression. Most often, expressions will be relational expressions.**
CIS 210

Common flow of control

if <boolean expression>:
    <block of code>
elif <boolean expression>:
    <block of code>
elif <boolean expression>:
    <block of code>
else:
    1 block of code will execute
    <block of code>

<next Python statement>
Exploring

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

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

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

??

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

??
```
Exploring

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

>>> s = ''
>>> if (len(s) == 1) or (len(s) == 0):
    print('yes')
??
```
✓ for v. while loops
✓ namespaces → variable scope
  • Boolean data type/conditionals
  • Monte Carlo algorithm for approximating pi
  • strings – sequences, immutable data types
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 the following code:

>>> temp_alert(90)
??
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')

What is the result of executing the following code:

>>> temp_alert(90)

??
def temp_alert(temp):
    
    """(number) -> None
    
    return information about the temperature
    ""

    if temp >= 90:
        msg = 'hot'

    if temp >= 80:
        msg = 'very warm'

    if temp >= 70:
        msg = 'warm'

    if temp >= 60:
        msg = 'cool'

    return msg

What is the result of executing the following code:

>>> temp_alert(90)

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

What is the result of executing the following code:

>>> temp_alert(90)
(a) cool
(b) warm
(c) very warm
(d) hot
def temp_alert(temp):
    """(number) -> None
    print information about the temperature"
    
    if temp >= 60:
        print('cool')
    elif temp >= 70:
        print('warm')
    elif temp >= 80:
        print('very warm')
    elif temp >= 90:
        print('hot')
    return None

What is the result of executing
the following code:

>>> temp_alert(90)

correct: (a) cool
(b) warm
(c) very warm
(d) hot
def temp_alert(temp):
    '''(number) -> None
    print information about the temperature'''
    
    if temp >= 90:  # What is the result of executing the following code:
        print('hot')
    elif temp >= 80:
        print('very warm')
    elif temp >= 70:
        print('warm')
    elif temp >= 60:
        print('cool')
    return None

>>> temp_alert(90)
??
Nifty Python - style

```python
a = -99
b = 88
c = 77

if a < b and b < c:
    print('most languages')
    print('better')

if a < b < c:
    print('Python allows chaining of relational operators')
```
CIS 210

Boolean expressions - **style**

```python
>>> isinstance(101, int)
True

>>> isinstance(101, str)
False

if isinstance(101, int) == True:

    if isinstance(101, int):
```
What Boolean value is returned by each of the following:

1. if isinstance(101, int) == 'True':

2. if isinstance(101, int) == True:

3. if isinstance(101, int):
Boolean expressions - style

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

if isinstance(x, int) == True:

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

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

if isinstance(x, int) == True:

if isinstance(x, int):

if isinstance:

<will this code be executed??>   (yes)
Boolean data type (is trickier than you might think for a data type that has only two values)

• 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
Figure 2.9

\[ r = 1 \]

and

\[ \text{Area of Circle} = \pi \times r^2 \]

so

\[ \text{Area of Circle} = \pi \]

\[ \text{Area of Square} = 4 \]
Area of semi-circle = $\pi/4$
Area of enclosing square = 1
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
  \[
  \frac{\pi/4}{1} = \frac{\pi}{4}
  \]
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 $inCircleCt/numDarts$

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

\[
\frac{\text{inCircleCt}}{\text{numDarts}} = \frac{\pi}{4} \Rightarrow \pi = 4 \times \frac{\text{inCircleCt}}{\text{numDarts}}
\]

- to determine whether a dart has landed in the circle – use formula for finding the distance between the point and the origin: $d = \text{math.sqrt}(x**2 + y**2)$

- how do we throw darts at the board??
Figure 2.10
Monte Carlo Simulation (Problem 3-2): The random module

```python
import random

• 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) to generate an approximate value for pi:

- throw a dart (use random.random)
- test whether it is in the circle (distance formula)
- keep track of number of darts that land in the circle (accumulator pattern again)

approximate pi using

\[ \pi = 4 \times \left( \frac{\text{inCircleCt}}{\text{numDarts}} \right) \]
Monte Carlo Simulation (Problem 3-2)

import random
import math

def montePi(numDarts):
    """
    """

    inCircle = 0  # initialize accumulator variable
    for i in range(numDarts):
        x = random.random()  # throw the darts
        y = random.random()
        d = math.sqrt(x**2 + y**2)  # check location
        if d <= 1:
            # it’s in the circle
            inCircle += 1  # increment accumulator variable
    pi = inCircle / numDarts * 4  # approximating pi
    return pi
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 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
✓ 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
>>> x = 'PYTHON ROCKS'
Strings are sequences of characters.

<table>
<thead>
<tr>
<th>Positive indexes</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>P</td>
<td>Y</td>
<td>T</td>
<td>H</td>
<td>O</td>
<td>N</td>
<td>R</td>
<td>O</td>
<td>C</td>
<td>K</td>
<td>S</td>
<td></td>
</tr>
<tr>
<td>Negative indexes</td>
<td>−12</td>
<td>−11</td>
<td>−10</td>
<td>−9</td>
<td>−8</td>
<td>−7</td>
<td>−6</td>
<td>−5</td>
<td>−4</td>
<td>−3</td>
<td>−2</td>
<td>−1</td>
</tr>
</tbody>
</table>
Operators
Concatenation +
Repetition *
Indexing [ ]
Slicing [ : ]

Methods
upper
lower
center
count
index
find
replace
startswith

Function
len
CIS 210

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

Note: "overloaded operators"

```python
>>> 99 + 100
299
>>> 2 * 10
20
>>> 'hello' + 'goodbye'
'hellogoodbye'
>>> 'hello' * 4
'hhellohellohello'
```
```
>>> len(x)
??
>>> x[4]               >>> x[-1]
??                     ??
>>> x[1:4]             >>> x * 3
??                     ??
>>> x + 'yes'
??
```
CIS 210 iClicker Question

```python
>>> x = 'RIP TERRY JONES'
```

```python
>>> x[1:3]
```

(a) ‘RIP’
(b) ‘IP’
(c) ‘IP ‘
(d) ‘IP T’
CIS 210 iClicker Question

```python
>>> x = 'RIP TERRY JONES'
```

```python
>>> x[1:3]
```

(a) ‘RIP’

Correct: (b) ‘IP’

(c) ‘IP ‘

(d) ‘IP T’

*note that the apostrophes would not be printed. I put them here to differentiate possible space characters in the string*
>>> x = ‘PYTHON ROCKS'

find method, for example:

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

find method, for example:

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

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

>>> 'PYTHON ROCKS'.find('O')
4
operators:  +, -, *, /, //, %, <, !=, and, in, [], [:], ...

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

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

they are all methods
>>> str.__add__('a', 'b')
'ab'

>>> 'a'.__add__('b')
'ab'

>>> int.__add__(4, 3)    # 4 + 3
7

>>> (4).__add__(3)       >>> 4.__add__(3)
7  #why parens or space?

>>> str.__len__('abcd')
4

>>> 'abcd'.__len__()
for is a sequential operator

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

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

??
for is a sequential operator

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

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

We
c
for is a sequential operator

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

```python
>>> s = 'jello, world'
```

```python
>>> s[0] = 'h'
```

??
strings are **immutable** sequences

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

TypeError: 'str' object does not support item assignment
strings are immutable sequences

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

>>> s = 'hello, world'
>>> s = 'h' + s[1:]
```
>>> type(True)  
a) <class 'bool'>

>>> type('True')  
b) <class 'str'>

>>> type(true)  

>>> type('true')  

>>> type('false')  
c) NameError: name 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, algorithm, 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
Computational Problem Solving: designing, implementing, checking, revising algorithms/pgms.

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

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.