Boolean data type/conditionals
indefinite iteration/while loops
Python types (objects)
strings – sequential, immutable type
Monte Carlo algorithm for approximating pi

CIS 210 code demos
• solve a problem using a structured approach to computational problem-solving:
  • understand problem
  • solve problem
  • plan how to implement solution as a Python program
  • implement working solution
  • code conforms to CIS 210 style guidelines
• problems from projects, labs, class notes
• grade per project rubric
• scheduled appointments (Canvas calendar)
• 1 per term (wks 5-10)
• code demo demo video available week 4
• some optional practice code demo times available (Canvas calendar)

CIS 210 Introduction Computer Science
>>> myvar = 123
>>> myvar
123
>>> abs = 456
>>> abs
456
>>> abs(-7)
7
>>> for = 4
??

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

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

>>> if 4 > 5:
    print('surprise')
>>>  

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>  

Boolean expressions

logical and 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 Boolean values (only)
a != b         # with logical operators

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

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

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')

CIS 210

Exploring

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

CIS 210

Exploring

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

CIS 210

Exploring

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

CIS 210

def temp_alert(temp):
    ""
    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

What is the result of executing the following code:

>>> temp_alert(90)
??
def temp_alert(temp):
    '''
    return information about the temperature
    '''
    if temp >= 90:
        return 'hot'
    elif temp >= 80:
        return 'very warm'
    elif temp >= 70:
        return 'warm'
    elif temp >= 60:
        return 'cool'
    return 19

CIS 210

def temp_alert(temp):
    '''
    return information about the temperature
    '''
    if temp >= 90:
        return 'hot'
    elif temp >= 80:
        return 'very warm'
    elif temp >= 70:
        return 'warm'
    elif temp >= 60:
        return 'cool'
    return

CIS 210

CIS 210

Nifty Python - style

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

x = 44

if isEven(x) == 'True':
    <will this code be executed?>

if isEven(x) == True:
    <will this code be executed?>

if isEven(x):
    <will this code be executed?>
Boolean expressions - style

```python
if isEven(x) == 'True':
    if isEven(x) == True:
        if isEven(x):
            if isEven:
                <will this code be executed?>
```

Boolean data type (is trickier than you might think for a data type that has only two values)

- logical 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
- use good style for Boolean expressions
- double (triple) check Boolean expressions

Boolean data type/conditionals

- indefinite iteration/while loops
- Python types (objects)
- strings – sequential, immutable type
- Monte Carlo algorithm for approximating pi

while loop

```python
now that we have Boolean expressions ...
most general type of loop

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

for example,

```python
start with a random card from a deck of cards.
choose cards until the sum of all the cards in the hand is >= 21.

myhand = randint(1,10)
??
```
for example, start with a random card from a deck of cards.
choose cards until the sum of all the cards in the hand is $\geq 21$.

```
myhand = randint(1, 10)

while myhand < 21:
    draw = randint(1, 10)
    myhand += draw

print(myhand)
```

write a function, mypow, that returns the value of 2 raised to the $p$th power

```
def mypow(p):
    r = 1
    for ctr in range(p):
        r *= 2
    print(r)
    return
```

for (repeat) loops are preferred for definite iteration
A while loop is needed when we can’t specify ahead of time how many times the loop will run (indefinite iteration).

```python
myhand = randint(1, 10)  # 1) initialize loop variable

while myhand < 21:  # 2) check end condition
    draw = randint(1, 10)  # 3) move loop var toward end condition
    myhand += draw

print(myhand)
```

The programming language we use provides a (relatively) high level method for communicating with the computer.

Theoretical basis of computer science: any “Turing complete” language has the same functionality as any other language.

read/write to memory (e.g., variables), (sequential) flow of control, conditionals (branching, repetition)

Recall: Python objects (elements)
- type
- value(s)
- memory location

<table>
<thead>
<tr>
<th>Python objects</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
</tr>
<tr>
<td>function</td>
</tr>
</tbody>
</table>

- Boolean data type/conditionals
- indefinite iteration/while loops
  - Python types (objects)
  - strings – sequential, immutable type
  - Monte Carlo algorithm for approximating pi
Python objects
• type
• value(s)
• memory location

>>> 4
# objects have values
4

>>> 'hello'
'hello'

>>> round
# values may be complex
<built-in function round>

>>> type(4)
# objects have types
<class 'int'>

>>> type('hello')
<class 'str'>

>>> type(round)
<class 'builtin_function_or_method'>

>>> isinstance(4, int)
# checking object type
True

>>> isinstance('hello', str)
True

>>> isinstance(round, str)
False

>>> isinstance(True, str)
False

>>> 4 + 5
9

>>> 'hello'[0]
'b'

>>> True and True
True

>>> 4 + None
TypeError: unsupported operand type(s) for +: 'int' and 'NoneType'

>>> 100 + '1'
TypeError: unsupported operand type(s) for +: 'int' and 'str'

>>> True[0]
TypeError: 'bool' object is not subscriptable

def isEven(n):
    """
    (n: int) → bool
    return True if n is even,
    False otherwise.
    ""
    >>> isEven(100)
    True
    >>> isEven(1)
    False
    >>> return n % 2 == 0
```python
def example(s, n):
    '''(s: ??, n: ??) → ??
    TYPE CONTRACT SPECIFIES EXPECTED TYPE FOR EACH PARAMETER AND THE RETURNED VALUE
    >>> example('abc', 2)
    False
    >>> example('banana', -1)
    True
    >>>
    return s[n] == 'a':
```

**Strings**

```
>>> 'PYTHON ROCKS'
# string literal evaluates to itself
>>> x = 'PYTHON ROCKS'
>>> x
'PYTHON ROCKS'
>>> type(x)  # type literal evaluates to <class 'str'>
<class 'str'>
>>> isinstance(x, str)
True
```

Strings are **sequences** of characters.
Operators
Concatenation +
Repetition *
Indexing []
Slicing [:]

Note: "overloaded operators"

```python
>>> 99 + 100
??
>>> 'hello' + 'goodbye'
??
```

```
>>> len(x)
??
>>> x[4]
??
>>> x + 'yes'
??
```

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

```
operators: +, -, *, /, //, %, <, !=, [], [:], ...
"syntactic sugar" for function calls
functions: isinstance, min, max, print, ...
use with multiple data types
methods: str.find, str.index, ...
associated with a specific data type
```

```
for is a sequential operator
x = "We can’t stop for gas, we’re already late."
>>> for ch in x:
    print(ch)
W
e
```

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

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

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

>>>

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 it is 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-c)

• the area of the circle is π/4 and area of square is 1
• the fraction of darts that lands in the circle is $rac{\pi/4}{1} = \frac{\pi}{4}$

Monte Carlo Simulation (Problem 3-c)

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

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 $\frac{\pi/4}{1} = \frac{\pi}{4}$
• 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-c)

• the area of the circle is \( \pi r^2 \) and area of square is \( r^2 \)
• the fraction of darts that land in the circle is \( \frac{\pi}{4} \)
• the fraction of darts that land in the circle is \( \frac{\text{inCircleCt}}{\text{numDarts}} \)
• \( \pi = 4 \times \frac{\text{inCircleCt}}{\text{numDarts}} \)

• throw darts - generate \( x \) and \( y \) using \( \text{random()} \)
  >>> from random import random
  >>> help(random)
  Help on built-in function random:
  random()
  random() - x in the interval [0, 1).

Monte Carlo Simulation (Problem 3-c)

to generate an approximate value for \( \pi \):
• throw a dart (random to get \( x, y \) coordinates)
• test whether it is in the circle (distance formula)
• keep track of number of darts that land in the circle (accumulator pattern again) and total darts thrown
• approximate \( \pi \) using
  \[ \pi = 4 \times \frac{\text{inCircleCt}}{\text{numDarts}} \]

73

74

Monte Carlo Simulation (Problem 3-c)

```python
from random import random
from math import sqrt, pi

def montePi(numDarts):
    inCircle = 0  # initialize accumulator variable
    for i in range(numDarts):
        x = random()  # throw the darts
        y = random()  # check location
        d = sqrt(x**2 + y**2)  # it's in the circle
        if d <= 1:
            inCircle += 1  # increment accumulator variable
    pi = inCircle / numDarts * 4  # approximating pi
    return pi
```

75

76

Monte Carlo Simulation (Problem 3-c)

(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`, to compare the approximate value of \( \pi \) generated by the Monte Carlo method to the value of \( \pi \) from the math module, and report on any error in the approximation

77

78
CS 210

- Boolean data type/conditionals
- indefinite iteration/while loops
- Python types (objects)
- strings – sequential, immutable type
- Monte Carlo algorithm for approximating pi

Python toolkit

- numeric data types (int, float) and operations (e.g., +, %, pow, round, abs)
- string data type and operations (e.g., +, len, count, find)
- Boolean data type and operations (e.g., -, and)
- expressions
- print, input
- variables (identifiers)
- assignment statement
- Python repetition – for (repeat), while
- Python conditionals (selection) – if
- user-defined functions (def, parameter list), docstring, return (values)
- IDE/interactive development environment
- Python Standard Library – math, random, turtle, doctest modules; import

CS 210 is a community of learners where

- Everyone is welcome
- Everyone is respected
- We value intellectual challenges and deliberate practice in pursuit of new knowledge and skills
- We support and encourage each other
- We celebrate our own and each other’s accomplishments

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; use natural language, algorithm, pseudocode (not Python code)
- check algorithm using your examples – revise algorithm, re-review spec, if needed
- start writing code at the top: write the main (top-level) function (comment out lines of code as needed)
- and then move down: 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

Programming/Computer Science concepts

- Computer Science overview: support for Computational problem solving
- Computational Problem solving: designing, implementing, testing, revising algorithms/objects
- General programming style: function documentation (inform and document code, examples of function calls, and invoked behaviors) use of structured repetition and selection to code, functional and/or process-oriented, clean code
- Python as a programing language and Python as an interpreter (program) abstraction
- Programming is application based: end users, developers, maintainers, designers
- Python libraries and modules... (more later) packages, functions, operators
- Python problems... strings, lists, arrays, types, memory
- Computational problem solving: abstraction, mathematical expression
- Solving a problem... (reading, writing, revising, re-solving, using tools from the Python toolkit, start writing the body of the function)
-.function header, –function docstring, assignment statements and not expressions and do not return a value
- type contract
- named parameters
- Python function parameters: call by reference (nonlocal variables), call by value (arguments)
- For loops for constructing sets, lists, and tuples
- Functions for starting, ending, forming, deleting, and using value
- Functions can return values (parameter passing)
- Functions return values in the form of results (e.g., output)
- When loops when a function... (return)
- Functions return values to a caller (return from function)
- Functions always return a value
- Variables can have the same value
- Exceptions can be useful
- Test cases for a top-level function, Monte Carlo algorithms
- Testing and debugging

CS 210 Learning Outcomes

- understand, develop, implement algorithms for computational problem solving
- use structured design and testing methods to develop and implement programs
- read, write, review, 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 problems, data types, and algorithms

CIS 210 Learning Outcomes

Computer science overview: support for Computational problem solving

Computational Problem solving: designing, implementing, testing, revising algorithms/objects

General programming style: function documentation (inform and document code, examples of function calls, and invoked behaviors) use of structured repetition and selection to code, functional and/or process-oriented, clean code

Python as a programing language and Python as an interpreter (program) abstraction

Programming is application based: end users, developers, maintainers, designers

Python libraries and modules... (more later) packages, functions, operators

Python problems... strings, lists, arrays, types, memory

Computational problem solving: abstraction, mathematical expression

Solving a problem... (reading, writing, revising, re-solving, using tools from the Python toolkit, start writing the body of the function)

function header, –function docstring, assignment statements and not expressions and do not return a value

type contract

named parameters

Python function parameters: call by reference (nonlocal variables), call by value (arguments)

For loops for constructing sets, lists, and tuples

Functions for starting, ending, forming, deleting, and using value

Functions can return values (parameter passing)

Functions return values in the form of results (e.g., output)

When loops when a function... (return)

Functions return values to a caller (return from function)

Functions always return a value

Variables can have the same value

Exceptions can be useful

Test cases for a top-level function, Monte Carlo algorithms

Testing and debugging

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, review, 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 problems, data types, and algorithms

CIS 210 Learning Outcomes

Computer science overview: support for Computational problem solving

Computational Problem solving: designing, implementing, testing, revising algorithms/objects

General programming style: function documentation (inform and document code, examples of function calls, and invoked behaviors) use of structured repetition and selection to code, functional and/or process-oriented, clean code

Python as a programing language and Python as an interpreter (program) abstraction

Programming is application based: end users, developers, maintainers, designers

Python libraries and modules... (more later) packages, functions, operators

Python problems... strings, lists, arrays, types, memory

Computational problem solving: abstraction, mathematical expression

Solving a problem... (reading, writing, revising, re-solving, using tools from the Python toolkit, start writing the body of the function)

function header, –function docstring, assignment statements and not expressions and do not return a value

type contract

named parameters

Python function parameters: call by reference (nonlocal variables), call by value (arguments)

For loops for constructing sets, lists, and tuples

Functions for starting, ending, forming, deleting, and using value

Functions can return values (parameter passing)

Functions return values in the form of results (e.g., output)

When loops when a function... (return)

Functions return values to a caller (return from function)

Functions always return a value

Variables can have the same value

Exceptions can be useful

Test cases for a top-level function, Monte Carlo algorithms

Testing and debugging

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, review, 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 problems, data types, and algorithms

CIS 210 Learning Outcomes

Computer science overview: support for Computational problem solving

Computational Problem solving: designing, implementing, testing, revising algorithms/objects

General programming style: function documentation (inform and document code, examples of function calls, and invoked behaviors) use of structured repetition and selection to code, functional and/or process-oriented, clean code

Python as a programing language and Python as an interpreter (program) abstraction

Programming is application based: end users, developers, maintainers, designers

Python libraries and modules... (more later) packages, functions, operators

Python problems... strings, lists, arrays, types, memory

Computational problem solving: abstraction, mathematical expression

Solving a problem... (reading, writing, revising, re-solving, using tools from the Python toolkit, start writing the body of the function)

function header, –function docstring, assignment statements and not expressions and do not return a value

type contract

named parameters

Python function parameters: call by reference (nonlocal variables), call by value (arguments)

For loops for constructing sets, lists, and tuples

Functions for starting, ending, forming, deleting, and using value

Functions can return values (parameter passing)

Functions return values in the form of results (e.g., output)

When loops when a function... (return)

Functions return values to a caller (return from function)

Functions always return a value

Variables can have the same value

Exceptions can be useful

Test cases for a top-level function, Monte Carlo algorithms

Testing and debugging

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, review, 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 problems, data types, and algorithms

CIS 210 Learning Outcomes

Computer science overview: support for Computational problem solving

Computational Problem solving: designing, implementing, testing, revising algorithms/objects

General programming style: function documentation (inform and document code, examples of function calls, and invoked behaviors) use of structured repetition and selection to code, functional and/or process-oriented, clean code

Python as a programing language and Python as an interpreter (program) abstraction

Programming is application based: end users, developers, maintainers, designers

Python libraries and modules... (more later) packages, functions, operators

Python problems... strings, lists, arrays, types, memory

Computational problem solving: abstraction, mathematical expression

Solving a problem... (reading, writing, revising, re-solving, using tools from the Python toolkit, start writing the body of the function)

function header, –function docstring, assignment statements and not expressions and do not return a value

type contract

named parameters

Python function parameters: call by reference (nonlocal variables), call by value (arguments)

For loops for constructing sets, lists, and tuples

Functions for starting, ending, forming, deleting, and using value

Functions can return values (parameter passing)

Functions return values in the form of results (e.g., output)

When loops when a function... (return)

Functions return values to a caller (return from function)

Functions always return a value

Variables can have the same value

Exceptions can be useful

Test cases for a top-level function, Monte Carlo algorithms

Testing and debugging

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, review, 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 problems, data types, and algorithms