Python overview, continued

- Designing Computational Solutions to Problems
- CIS210 Programming Style Guidelines (Python docstring)

- string data type
- Boolean data type/conditional statements, revisited
- variable assignment, revisited
- executing a function, revisited
- indefinite iteration (while loop)

Computers do not solve problems - computers carry out solutions, specified by people, to problems.

The art of computational problem solving:
Designing computational solutions to problems

A step by step introduction

Computational Problem Solving

Once you understand the problem, developing computational solutions:

1) it’s already done, e.g., math.sqrt
2) implement an algorithm, e.g., functions to approximate square root, pi, convert temp, cricket chirp
3) reuse or refactor a function, e.g., montePi
4) develop algorithm and function from scratch, e.g., art_show
Develop thorough understanding of the algorithm:

**Iterative method:** a first approximation is produced, then a method which improves the accuracy of the solution accuracy of the solution is used for a certain number of iterations or until two successive approximations agree to the accuracy required.

\[ x_{k+1} = \frac{1}{2} \times (x_k + \frac{n}{x_k}), \text{where } x_0 = 1 \]
Review and understand project specifications (requirements):

`mysqrt` will have two parameters, `n`, the number to find the square root for, and `k`, the number of times the iterative square root approximation process should run. The function should return the approximate square root value for `n`.

```python
def mysqrt(n, k):
    '''
    DESIGNING AND DOCUMENTING CODE
    ACCORDING TO CIS 210 STYLE GUIDELINES
    '''
    pass
    return x
```

Write a template for the function – header, docstring, return

What is the input? What is the result? How is it reported? -- number to find square root for -- number of times to iterate -- approximate square root -- value is returned

```python
def mysqrt(n, k):
    '''
    (integer, integer) -> float
    '''
    pass
    return x
```
def mysqrt(n, k):
  '''
  (integer, integer) -> float  type contract
  brief description of function that mentions each parameter by name
  Generates an approximate square root for n, a positive integer, via an iterative process that runs k times.
  
  pass
  return x
  
  describe returned value
  The approximate square root is returned.
  and side effects (if any)
  
  pass
  return x

Every docstring of every function should include:

• type contract
• brief description that mentions each parameter by name
• side effects (e.g., print), if any
• returned value
• simple examples of usage
• calls/called by (if any, if helpful)
Every docstring of every function should include:

- type contract
- calls/called by, if any, if helpful
- brief description that mentions each parameter by name
- side effects (e.g., print), if any
- returned value
- simple examples of usage

→ reflects thoughtful design
→ contributes to usability/maintainability of code
→ integrated with Python help function

“One of the characteristics of a well-written function is the ability to read the code [including documentation] and see the underlying algorithm.”

Designing computational solutions to problems:

- make sure you understand the problem
- review, adapt, and/or design an algorithm for solving the problem
- review and make sure you understand the project specifications
- begin designing a computer solution (overall program, functions)
- start writing individual functions — header, docstring, return
- review the tools in your Python language toolkit thus far
- use Python tools to fill in code for individual functions
- test individual functions using docstring examples and other test cases given in the project specification and/or developed by you;
- fill out a bug log: 1) steps to reproduce the bug; 2) description of what you expect to happen; 3) description of what does happen; 4) steps taken to try to fix the bug.

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)
- expressions
- Python modules — math, turtle, random
- assignment statement
- Python repetition — for, while
- Python conditionals — if
- variable assignment
- user-defined functions
Translate the algorithm into Python code using tools from your Python toolkit.

Test code using your simple examples and/or examples given in the projects specifications.

Debug and revise your code (maybe even algorithm) until it works.

Debugging code:

Create a **bug log**:
1) steps to reproduce the bug
2) description of what you expect to happen
3) description of what does happen
4) steps taken to try to fix the bug.

*Keep a backup copy of your code!*
Operators | Methods
---|---
Concatenation + | upper
Repetition * | lower
Indexing [ ] | center
Slicing [: ] | count

Function
len

**STRINGS**

```python
>>> x = 'Python rocks'

find method, for example:

```python
>>> str.find(x, 'o')
4
```nn
```python
>>> x.find('o') >>> 'Python rocks'.find('s')
4 11
```nn

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

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

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

they are all methods
for is a sequential operator

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

for is a sequential operator

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

while loop

```python
while loop

indefinite iteration

most general type of loop

while <condition>:
    statement1
    statement2
...
    statement
```
while loop

\[
\begin{align*}
p &= 10 \\
i &= 1 \\
ctr &= 1 \quad \# initialize loop variable \\
while \ ctr \leq p: & \quad \# check end condition \\
i &= i \times 2 \\
ctr &= ctr + 1 \quad \# move loop var toward \\
print(i) & \quad \# the end condition
\end{align*}
\]

for loop

\[
\begin{align*}
num &= 12345678 \quad \# loop var \\
while \ len(num) > 0: & \quad \# check end condition \\
rdrigs &= num \% 100 \\
num &= num // 100 \quad \# advance loop var \\
\end{align*}
\]

loop var

\[
\begin{align*}
while \ len(num) > 0: & \quad \# check end condition \\
rdrigs &= num \% 100 \\
num &= num // 100 \quad \# advance loop var \\
\end{align*}
\]