Python functions, finish

conditional/selection statements

Boolean data types

relational operators

doctest module
When to write a function?

• When code repeats \hspace{1cm} \text{REUSE/}
  (or almost repeats – parameters) \hspace{1cm} \text{FLEXIBILITY}

• Useful bit of code (for future “repeats”)

• When you want to hide the details \hspace{1cm} \text{INFO-HIDING/}
  CLARITY

• When you want to break up a large problem \hspace{1cm} \text{ABSTRACTION}

Thinking – Understanding the problem and solution:

For each function –
  – give example(s) of use
  – determine type contract (parameters and return value)
  – write a brief description
  – create/revise/understand algorithm to solve the problem

Turn it into Python code:
  – write docstring (+ pass and return None)
  – review Python toolkit
  – write Python code

Test code using examples of use

Revise and retest
Functions can call functions:

```python
def combine(s, t):
    '''
    res1 = len(s)
    res2 = len(t)
    ttl_len = res1 + res2
    return ttl_len

    Which? Why?
```
Functions can call functions:

```python
def sketch():
    '''
    fd(100)
    ...
    return None
    '''
    result = fd(100)
    ...
    return None
```

Which? Why?
(0) Which of the following is a function call?

(A) >>> type(4.5)
(B) >>> def add_one(x):
    return x + 1
(C) >>> add_one(7)

a) (A) and (B)  b) (A) and (C)  c) all of the above  d) none of the above

(1) What value is returned after the following code is entered into the Python Shell?
>>> def square(x):
    return x * x
>>> square

a) <function square at 0x10274c560>  b) 9  c) none of the above
(2) What value is returned after the following code is entered into the Python Shell?

```python
>>> def square(x):
    return x * x

>>> square(3)
```

a) <function square at 0x10274c560>
b) 9
c) none of the above
def square(x):
    result = x * x
    return result

>>> x = 5
>>> x = 6
>>> square(x)
>>> x = square(x)
??
>>> x
??

def add1(x):
    x += 1
    return None

def main():
    >>> main()
    add1(4)
    square(5)
    return None
def square(x):
    result = x * x
    return result

def add1(x):
    x += 1
    print(x)
    return None

def main():
    add1(4)
    result = square(5)
    return None

>>> main()

result = add1(4)
result = square(5)
return result
0) b
1) a
2) b

3) 9  nothing is printed after assignment
   3  16
   NameError  NameError

4) 25  nothing is printed after assignment
   5  36

5) 5
6) 5
7) 5

How Much? Write a function, shipping, which determines the shipping cost of an order. Shipping is $.05 per pound but for shipments 100 pounds or more, subtract $1.50.
How Much? Write a function, shipping, which determines the shipping cost of an order. Shipping is $.05 per pound but for shipments 100 pounds or more, subtract $1.50.

Examples of use:
- 10 pounds -> $.50
- 50 pounds -> $2.50
- 100 pounds -> $5.00? $5.00 - $1.50 = $3.50?
- 200 pounds -> $10.00 - $1.50 = $8.50
- 0 pounds -> $0.00 ??
Examples of use:

```python
def shipping(w):
    
    """ (number) -> float
    Given weight w, a positive number, return shipping cost
    ($0.05/pound, $1.50 discount for weight 100 pounds or more).
    ""

cost = w * .05

    # what about when w is 100 pounds or more?
```

For code that executes only under certain conditions we need another tool:

```python
conditionals/if statements

    if is a Python keyword
```
Python if statements

if is a Python keyword

if statements affect the program’s flow of control

one path (or another) is selected for execution,

depending on a condition(s)

def shipping(weight):
    '''(number) -> float

    Return cost of shipping freight of weight.
    Cost is $.05 per unit, with $1.50 subtracted if weight is 100 units or more.

    >>> shipping(10)
    0.5
    >>> shipping(100)
    3.5
    >>> shipping(101)
    3.55
    '''
    cost = weight * .05
    if weight > 100:
        cost = cost - 1.50  # when condition "weight > 100" is True
        return cost  # then this block of code is executed.
    # (otherwise, it is not executed)
def shipping(weight):
    '''(number) -> float

    Return cost of shipping freight of weight.
    Cost is $.05 per unit, with $1.50 subtracted
    if weight is 100 units or more.

    >>> shipping(10)
    0.5
    >>> shipping(100)
    3.5
    >>> shipping(101)
    3.55
    '''
    cost = weight * .05
    if weight >= 100:  # when condition “weight >= 100” is True
        cost = cost - 1.50  # then this block of code is executed.
    return cost  # (otherwise, it is not executed)
CIS 122 Introduction to Programming/Problem Solving      Spring 2017

Python if statements

```python
>>> weight = 100
>>> cost = weight * .05

>>> weight >= 100   # a Boolean expression
True               # returns a Boolean value

>>> if weight >= 100:
    cost = cost - 1.50

>>> cost
??
```

```python
>>> weight = 50
>>> cost = weight * .05

>>> weight >= 100   # a Boolean expression
False              # returns a Boolean value

>>> if weight >= 100:
    cost = cost - 1.50

>>> cost
??
```
Boolean is another of Python’s built in data types:

- Integer
- Floating point
- String
- Function
- None
- **Boolean**
  - values – True, False
  - operators – not, and, or

Relational operators return Boolean values

<
>
<=
>=
==
!=

Relational operators return Boolean values

```python
>>> 6 < 10
True
>>> 6 != 10
True
>>> 6 == 10
False
>>> d = 99
>>> d < 100
True
>>> d <= 99
True
```
**Boolean operators**

```python
>>> not (d == 99)
>>> (90 < d) and (d < 100)
>>> 90 < d < 100

>>> x = 99
>>> y = 100
>>> (x > 99) or (y > 99)
>>> (x > 100) or (y > 100)
```

---

**Truth Tables**

<table>
<thead>
<tr>
<th></th>
<th>not</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
</tr>
</tbody>
</table>
Truth Tables

<table>
<thead>
<tr>
<th>and</th>
<th>or</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>

Truth Tables

<table>
<thead>
<tr>
<th>cold</th>
<th>windy</th>
<th>not cold and windy</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>
CIS 122 Introduction to Programming/Problem Solving  Spring 2017

Truth Tables

cold  windy  (not cold) and windy  not(c and w)
T    T          T
T    F          F
F    T          T
F    F          F

CIS 122 Introduction to Programming/Problem Solving  Spring 2017
Truth Tables

cold  windy  (not cold) and windy  not(c and w)
T    T          (not T) and T  not(T and T)  not(T)
      F          F          F
T    F          (not T) and F  not(T and F)  not(F)
      F          F          T
Order of Operations

• parentheses!!

• arithmetic – **, unary + -, * / // %, + -
• relational – all at same level of precedence -
  < > <= >= == !=
• boolean – not and or

for same level of precedence – left to right

Python if (selection/conditional) statements

```python
>>> a = 5
>>> b = 3

>>> if a > b:
    c = 10
    d = c * 2

>>> c
??
>>> d
??
```
Python if statements

>>> a = 3
>>> b = 5

>>> if a > b:
    c = 10
    d = c * 2

>>> c
??
>>> d
??

Python if statements

>>> a = 5
>>> b = 3

>>> if a > b:
    c = 10  # fork-in-the-road 1
else:
    c = 15  # fork-in-the-road 2

>>> c  # can take only 1 of these paths!
Python if statements

```python
>>> a = 3
>>> b = 5

>>> if a > b:
    c = 10  # fork-in-the-road 1
else:
    c = 15  # fork-in-the-road 2

>>> c  # can take only 1 of these paths!
```

Python if statements

```python
>>> a = 5
>>> a = 3
>>> a = 3

>>> b = 3
>>> b = 5
>>> b = 3

>>> if a > b:
    c = 10  # fork-in-the-road 1
elif a < b:
    c = 15  # fork-in-the-road 2
else:
    c = 20  # fork-in-the-road 3

>>> c  # can take only 1 of these paths!
```
Be careful! What is the result of executing >>> q1(90)?

def q1(score):
    gradepoint = 0
    if score >= 90:
        gradepoint = 4
    if score >= 80:
        gradepoint = 3
    if score >= 70:
        gradepoint = 2
    if score >= 60:
        gradepoint = 1
    return gradepoint
**Payday:** Write a function, `payday`, with parameters `hourly_wage` and `hours`. The function should compute and return the pay. For the first 40 hours worked, pay is at the input hourly wage. Any hours over forty but less than or equal to 60 are paid at 1½ times the regular hourly wage. Any hours over sixty are paid at 2 times the regular hourly wage.

```python
import doctest
def foo(x):
    '''(int) -> int

    add 1 to x and return the new value
    >>> foo(4)
    5
    >>> foo(10)
    12
    '''
    return x + 1
doctest.testmod()
```
CIS 122 Introduction to Programming and Problem Solving Spring 2017

>>> RESTART:
/Users/kfreeman/Documents/cis122spr17/project-solutionsSpr17/check.py
**************************************
File"/Users/kfreeman/Documents/cis122spr17/project-solutionsSpr17/check.py", line 12, in __main__.foo
Failed example:
foo(10)
Expected:
12
Got:
11
**************************************
1 items had failures: 1 of 2 in __main__.foo
***Test Failed*** 1 failures.

CIS 122 Introduction to Programming Spring 2017 Week 4, 1

Python toolkit so far

Data types:
numbers (int and float)
strings
functions (executable data type)

Built in functions and operators:
+ - * / // % ** [ ] [:] int, float, str for
pow, round, abs
print, input
min, max
type, id
help
range

Errors:
SyntaxError  TypeError  NameError  IndexError
string methods – count, find, index

import
modules – math, turtle

function definition (def)

conditionals (if elif else)
relational operators
Boolean data type

automated testing – doctest module

Bug Collection So Far

>>> x === 5
SyntaxError: invalid syntax

>>> y
NameError: name 'y' is not defined

>>> 9 + 'planets'
TypeError: unsupported operand type(s) for +: 'int' and 'str'

>>> greeting[len(greeting)]
IndexError: string index out of range
CIS 122 Introduction to Programming Spring 2017

Bug Collection So Far

>>> abs()
Traceback (most recent call last): File "<pyshell#0>", line 1, in <module>
abs() TypeError: abs() takes exactly one argument (0 given)
>>>