02-5 Conditionals and Recursion

CSI 500

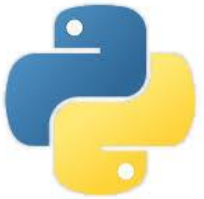
Spring 2018

Course material derived from:
Downey, Allen B. 2012. "Think Python, 2nd Edition". O'Reilly Media Inc., Sebastopol CA.

"How to Think Like a Computer Scientist" by Peter Wentworth, Jeffrey Elkner, Allen B. Downey, and Chris Meyers. Oct 2012
http://openbookproject.net/thinkcs/python/english3e/index.html

The Modulus operator

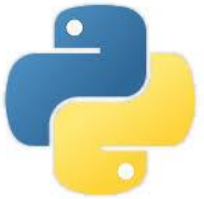
- Python provides the '%' modulus operator
  - Returns the remainder from integer division
- Useful for other tasks
  - determine if one number is evenly divisible by another (modulus will be 0)
  - truncate rightmost digits from a number



```
quotient = 7 / 3       # integer division
print( quotient )
2
remainder = 7 % 3
print( remainder )
1

13411601779454228 % 135792468
0

123 % 10
3
123 % 100
23
```

Boolean Expressions

- A Boolean expression evaluates to either true or false
  - Python True and False (note case sensitivity) are special 'bool' types, not ints
- **COMMON ERROR**
  - the equality operator is ==
  - the assignment operator is =
  - new Python programmers often confuse these



```
5 == 5
True
5 == 6
False

type( True )
<type 'bool'>
```

Python Relational Operators

- Used to compare two variables or expressions
- Can be grouped using parenthesis

Expression	Meaning
x == y	x is equal to y
x != y	x is not equal to y
x < y	x is less than y
x > y	x is greater than y
x >= y	x is greater than or equal to y
x <= y	x is less than or equal to y
x <> y	INVALID SYNTAX
x => y	INVALID SYNTAX
x <= y	INVALID SYNTAX
x <- y	INVALID SYNTAX

Logical Operators

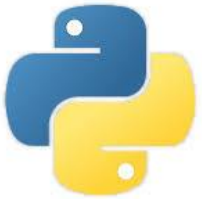
- Python supports three logical operators **and**, **or**, **not**
- Chained operators evaluated from left to right
  - True and False or True
  - True
- Python is a bit lax and will evaluate any nonzero number as True
  - True and 10 evaluates to True

Expression	Meaning
x and y	True if x is True and y is True
x or y	True if x is True and y is True True if x is True and y is False True if x is False and y is True
not x	True if x is False

Conditional Execution

- Allows decision making in program
  - format is as follows

```
if condition :
    expression(s)
```



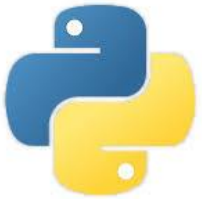
```
x = 5
if x > 4:
        print( "x is greater than 4" )
```

```
x is greater than 4
```

Alternative Execution

- Allows alternative course of action for decision making in program
  - format is as follows

```
if condition :
    expression(s)
else :
    expression(s)
```



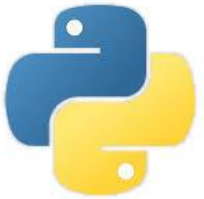
```
x = 5
if x > 4 :
        print( "x is greater than 4")
else:
        print( "x is less than or equal to 4")
```

```
x is greater than 4
```

Chained Conditionals

- Allows multiple alternative courses of action for decision making in program
  - first branch that is true is executed
  - if statement immediately ends - no need for an explicit 'break' statement like C/C++
  - format is as follows

```
if condition :
    expression(s)
elif condition :
    expression(s)
else :
    expression(s)
```



```
x = 5
y = 7
if (x > y):
        print( 'x is greater than y')
elif (x < y):
        print( 'x is less than y')
else:
        print( 'x and y are equal')
```

```
x is less than y
```

Nested Conditionals

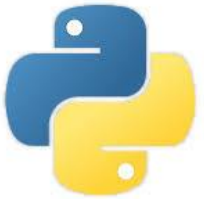
- You can write a conditional within another conditional
- Syntactically correct but can become awkward to read with deep nesting

```
if x == y:
        print('x and y are equal')
    else:
        if x < y:
            print('x is less than y')
        else:
            print('x is greater than y')
```

Recursion

- Python allows a function to call itself
  - This is called "recursion"

```
def countdown( n ):
    if n == 0:
            print(' BLAST OFF')
     else:
            print( n )
            countdown( n - 1)

countdown( 5 )
5
4
3
2
1
BLAST OFF
```



Recursion again

- It is very useful for some types of problems
- Consider the factorial of an integer, n!

  f(0) = 1

  f(1) = 1

  f(n) = n * f(n-1)
- We can implement this directly in Python using a recursive function

```
>> def fact(n):
        if (n == 0):
            return(1)
        elif (n == 1):
            return(1)
        else:
            return( n * fact(n-1))

for k in range(8):
    print( K, fact(K) )
```

```
0 1
1 1
2 2
3 6
4 24
5 120
6 720
7 5040
8 40320
```

Recursion again again

- Let's revisit our factorial function
- What happens if we call fact(1.5)
  - it will infinitely recurse (can you see why?
- Let's add some defensive code for type checking
  - what happens with fact('fred')
  - what happens with fact(1.5)
  - what happens with fact(-2)

```
# original version
def fact( n ):
        if n == 0:
                return 1
        if n == 1:
                return 1
        else:
                return n * fact( n - 1)

# add some type checking
def fact( n ):
        if not instance(n, int):
                print('fact only defined for ints')
                return None
        elif n < 0:
                print('fact not defined for negatives')
                return None
        elif n == 0:
                return 1
        else:
                return n * fact(n - 1)
```

Infinite Recursion

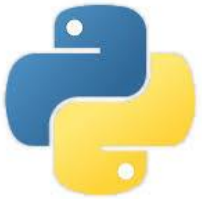
- When using recursive functions, be careful to ensure there is an end state
- Easy to miss this, causing the system to go into infinite recursion and hang
  - Actually, Python will shut down gracefully when it runs out of stack space, but still not a good idea

```
def bad_recursion():
    print('about to recurse..')
    bad_recursion()

about to recurse...
about to recurse...
about to recurse...
about to recurse...
about to recurse...

# a bunch more messages…

Traceback (most recent call last):
  File "<pyshell#4>", line 1, in <module>
    bad_recursion()
  File "<pyshell#3>", line 3, in bad_recursion
    bad_recursion()
  File "<pyshell#3>", line 3, in bad_recursion
    bad_recursion()
```

Keyboard input

- You can interact with the user to gather input
- Raw console input reads from keyboard
  - inputs are read as **strings**
  - you can do type conversions after reading the data
  - you have to handle ill formed inputs

```
val = input('type in a number \n')
type in a number
7
val
'7'
type(val)
<class 'str'>

val = input('type in a number\n')
type in a number
7
val = int(val)
val
7
type(val)
<class 'int'>
```



Summary

- Python has a wide variety of operators
  - Modulus, Boolean, relational
- Python supports conditional expressions
  - If statement
  - If Else statement
  - If Elif Else statement
- Python supports recursive functions
  - Make sure recursion eventually ends!
- Python supports direct keyboard input
  - not often used