Computational Basics

py-loops
Logic & lists Redux
s = 'ABcd'
if not s[0:2].isupper():
    if s[0] == s[2]:
        print( s[0] )
    else:
        print( s[1] )
else:
    if s[1] != s[2]:
        print( s[-1] )
    else:
        print( s[-2] )
s = 'ABcd'
if not s[0:2].isupper():
    if s[0] == s[2]:
        print( s[0] )
    else:
        print( s[1] )
else:
    if s[1] != s[2]:
        print( s[-1] )
    else:
        print( s[-2] )
'd'
s = 'abcd'
if not s.isalpha():
    print( s[0] )
elif s.isupper():
    print( s[-1] )
elif 'ab' in s:
    print( s[-2] )
else:
    print( s[1] )
s = 'abcd'
if not s.isalpha():
    print( s[0] )
elif s.isupper():
    print( s[-1] )
elif 'ab' in s:
    print( s[-2] )
else:
    print( s[1] )
'c'
Functions
try = input( "Give me a password: " )
if len( try ) < 8:
    # must be 8 characters at a minimum
    print( 'Invalid password' )
elif try.isupper() or try.islower():
    # must have both upper- and lower-case letters
    print( 'Invalid password' )
elif try.isalpha() or try.isdigit():
    # must have letters and numbers
    print( 'Invalid password' )
else:
    print( 'Password OK' )
def validate_password( try_):
    if len( try_ ) < 8: # must be 8 characters at a minimum
        return False
    if try_.isupper() or try_.islower(): # must have both upper- and lower-case letters
        return False
    if try_.isalpha() or try_.isdigit(): # must have letters and numbers
        return False
    return True # password is OK
A function is a small program (block of code) we can run within Python.
A *function* is a small program (block of code) we can run within Python.
- Saves us from rewriting code
- Don’t reinvent the wheel!
A function is a small program (block of code) we can run within Python.

- Saves us from rewriting code
- Don’t reinvent the wheel!

Analogy: If operators are verbs, functions are more verbs.
A function is a small program (block of code) we can run within Python.
- Saves us from rewriting code
- Don’t reinvent the wheel!
- Analogy: If operators are verbs, functions are more verbs.
- Also called subroutine or procedure.
When we want to execute a function, we call or invoke it.
Function calls

- When we want to execute a function, we call or invoke it.
- Use name of the function with parentheses.
When we want to execute a function, we call or invoke it.

Use name of the function with parentheses.

\[
\text{print( 'hello world' )}
\]
When we want to execute a function, we call or invoke it.

Use name of the function with parentheses.

- `print('hello world')`
- `bin(11)`
When we want to execute a function, we call or invoke it.

- Use name of the function with parentheses.
  ```python
  print( 'hello world' )
  bin( 11 )
  ```

- Many functions come built-in to Python or in the standard library.
When we want to execute a function, we call or invoke it.

- Use name of the function with parentheses.
  - `print( 'hello world' )`
  - `bin( 11 )`

- Many functions come built-in to Python or in the standard library.

- Others we will compose at need (see lec06).
Arguments

- Functions can act on data.
- *Arguments* are the input to functions.
- Functions can return a value (*fruitful*).
- Return values are the output of a function.
Functions can act on data.

*Arguments* are the input to functions.

Functions can return a value (fruitful).

Return values are the output of a function.

- `print('10')`
- `len('Rex Kwon Do')`
- `abs(-123)`
Arguments

- Arguments are values passed to a function.
- A function can accept zero to many arguments.
Arguments are values passed to a function.
A function can accept zero to many arguments.
Multiple arguments are separated by commas:
- \texttt{min( 1,4,5 )}
- \texttt{max( 1,4,5 )}
Type conversion

- A set of built-in functions to convert data from one type to another.

Functions
A set of built-in functions to convert data from one type to another.

- `float( "0.3" )`
- `str( 3 + 5j )`
- `int( 1.5 )`
A set of built-in functions to convert data from one type to another.

- `float( "0.3" )`
- `str( 3 + 5j )`
- `int( 1.5 )`

Be careful of nonsense:
- `int( "Rex" )`
- `int( 3 + 5j )`

Also called subroutine or procedure.
When we want to execute a function, we call or invoke it.

Use name of the function with parentheses.
- `print( 'hello' )`

Many functions come built-in to Python or in the standard library.
- `type( 1+3.4 )` — type of an expression
- `len( 'tortoise' )` — length of str

Others we will compose at need.
Example: Defining functions

```python
def pow(a, b):
    y = a ** b
    return y
```
We define a new function with the following:

- the keyword `def`
- the name of the function
- a pair of parentheses
- a `block` of code
- a `return` statement (optional)
Example: Defining functions

```python
def greetings():
    print("Bom dia!")
    print("Bonjour!")
    print("Ni hao!")
    print("Hello!")
    print("Guten tag!")
    print("Konichiwa!")
```

header

body
Variables defined inside of a block are *independent* of variables outside of the block.

Variables inside a block *do not exist* outside of the block.

Blocks are isolated from the rest of the code!
def pendulum( L ):
    import math
    g = 9.8  # m/s^2
    T = 2 * math.pi * math.sqrt( L / g )
    return T

# out here, I shouldn’t refer to vars in the function
time = pendulum( 1.0 )
print( time )
Example: Defining functions

def pendulum( L ):
    import math
    L = 1.0  # m  Why doesn’t this work?
    g = 9.8   # m/s^2
    T = 2 * math.pi * math.sqrt( L / g )
    return T

# out here, I shouldn’t refer to vars in the function
time = pendulum( 1.0 )
print( time )
def pendulum( L ):
    import math
    L = 1.0  # m  Why doesn’t this work?
g = 9.8  # m/s^2
    T = 2 * math.pi * math.sqrt( L / g )
    return T

# out here, I shouldn’t refer to vars in the func
time = pendulum( 1.0 )
print( time )
Example: Defining functions

```python
a = 5
def fun():
    a = 3
    b = 4
    a = a + b
fun()
print(a)
```
Functions can return values with the keyword `return`.  
```python
def three():
    return 3
```
- `return` immediately exits a function.
Functions can return values with the keyword `return`.
```python
def three():
    return 3
```

`return` immediately exits a function.
```python
def zero():
    return 0
    print('0')
```
Functions can accept values as parameters (input, arguments).

These variables are declared in the function header.

Multiple parameters are separated by commas.

```python
def print_message( msg ):
    print( 'I say:', msg )
```
def fun(a):
    return a+2

x = fun(2) * fun(3)

What is the value of x?
A  6
B  8
C 24
D None of the above.
Composing Functions

```python
def fun(a):
    return a+2

x = fun(2) * fun(3)

What is the value of x?
A 6
B 8
C 24
D None of the above. ★ (20)
```
def spacer(m):
    return m + ' , '

x = spacer( "abb") + spacer( "acab" )

What is the value of x?
A ‘abb acab ’
B ‘abb acab’
C ‘abbacab’
D None of the above (error)
def spacer(m):
    return m + ' , '

x = spacer("abb") + spacer("acab")

What is the value of x?
A 'abb acab ' ✫ (note the space!)
B 'abb acab'
C 'abbacab'
D None of the above (error)
Functions can have default values available for certain arguments.

If these are present, you can use the function in several ways:

```python
def my_default_value( a=5 ):
    print( 'I have the value %i.' % a )

my_default_value()
my_default_value( 6 )
my_default_value( a=6 )
```
Write a function `isclose` which assesses whether two values $a$ and $b$ are sufficiently near each other to be considered “equal”. Have the default relative tolerance be 0.001. (The relative tolerance is defined as $\frac{|a-b|}{\min(|a|,|b|)}$.)

```python
def isclose(a, b, rtol=1e-3):
    return (abs(a - b) / min(abs(a), abs(b))) <= rtol
```
Write a function `isclose` which assesses whether two values \( a \) and \( b \) are sufficiently near each other to be considered “equal”. Have the default relative tolerance be 0.001. (The relative tolerance is defined as \( \frac{|a-b|}{\min(|a|,|b|)} \).)

```python
def isclose( a, b, rtol=1e-3 ):
    return ( abs( a-b ) / min( a, b ) ) <= rtol
```