Python 101

Functions

Announcements

quiz: `quiz05` due Thurs 09/26

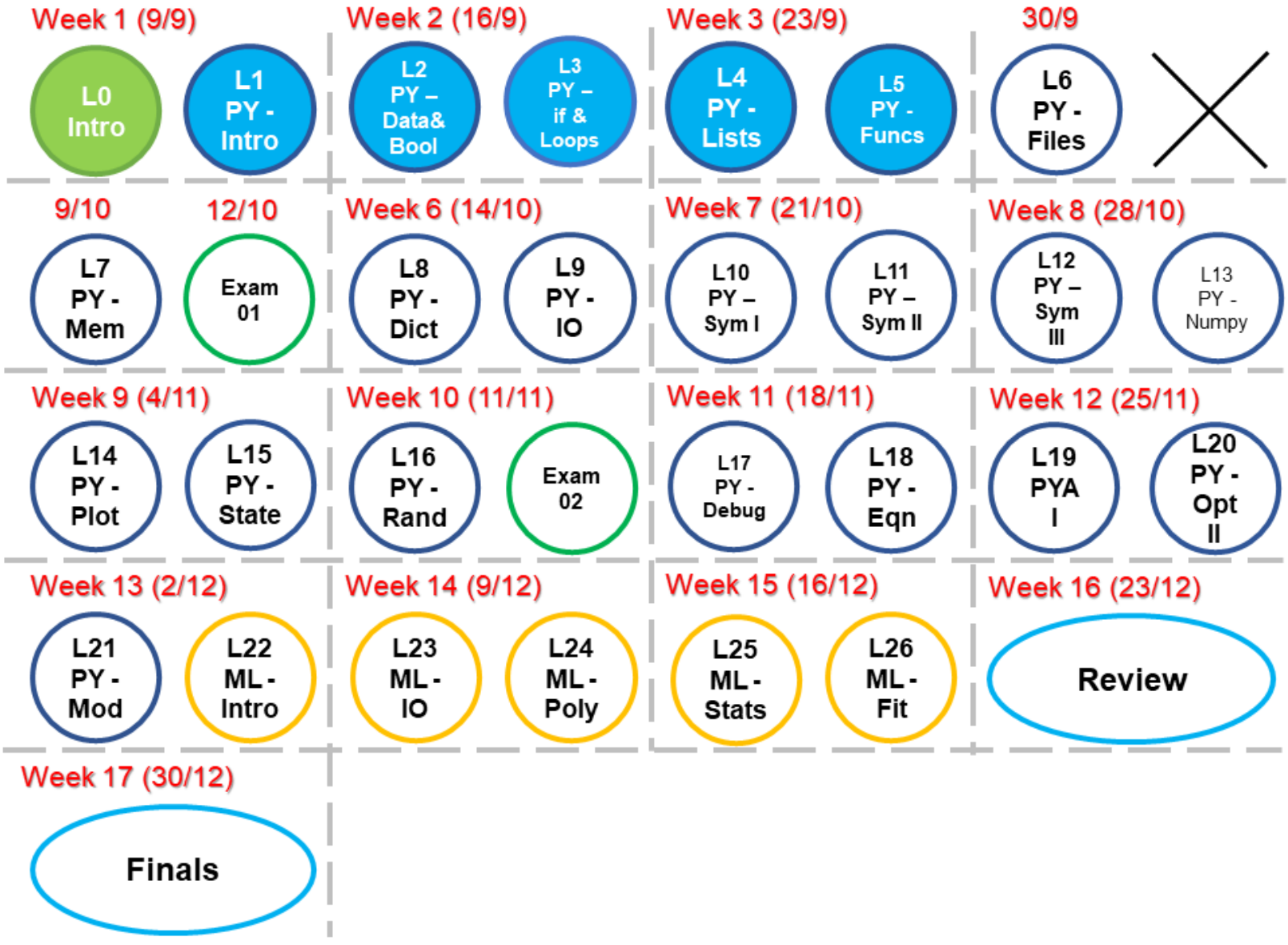
lab: `lab02` on Fri 09/27

lab: `lab03` on Sun 09/29

hw: `hw03` due Mon 09/30

exam01: `lec01 - lec05, hw01 - hw03, quiz01 - quiz05, lab01 - lab03`

Roadmap


Week 1 (9/9)
L0 Intro
L1 PY - Intro
Week 2 (16/9)
L2 PY – Data& Bool
L3 PY – if & Loops
Week 3 (23/9)
L4 PY - Lists
L5 PY - Funcs
30/9
L6 PY - Files
9/10
L7 PY - Mem
12/10
Exam 01
Week 6 (14/10)
L8 PY - Dict
L9 PY - IO
Week 7 (21/10)
L10 PY – Sym I
L11 PY – Sym II
Week 8 (28/10)
L12 PY – Sym III
L13 PY - Numpy
Week 9 (4/11)
L14 PY - Plot
L15 PY - State
Week 10 (11/11)
L16 PY - Rand
Exam 02
Week 11 (18/11)
L17 PY - Debug
L18 PY - Eqn
Week 12 (25/11)
L19 PYA I
L20 PY - Opt II
Week 13 (2/12)
L21 PY - Mod
L22 ML - Intro
Week 14 (9/12)
L23 ML - IO
L24 ML - Poly
Week 15 (16/12)
L25 ML - Stats
L26 ML - Fit
Week 16 (23/12)
Review
Week 17 (30/12)
Finals

Objectives

A. Write a simple `function` to implement a mathematical formula.
B. Use `function` to modularize code.
C. Explain how `variable scope` impacts what the program ”sees”.
D. Understand the difference between *returning a value* and *printing a value*.
E. Use default values of `(keyword arguments)` in functions.

Recap on List

Question 1

```
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] )
```

Question 1

```
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'
```

Question 2

```
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] )
```

Question 2

```
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'
```

String Methods

```
tryI = input( ”Give me a password:  ” )
```

String Methods

```
tryI = input( ”Give me a password:  ” )

if len( tryI ) < 8:
    # must be 8 characters at a minimum
    print( ’Invalid password’ )
elif tryI.isupper() or tryI.islower():
    # must have both upper- and lower-case letters
    print( ’Invalid password’ )
elif tryI.isalpha() or tryI.isdigit():
    # must have letters and numbers
    print( ’Invalid password’ )
else:
    print( ’Password OK’ )
```

String Methods

```
tryI = input( ”Give me a password:  ” )

if len( tryI ) < 8:
    # must be 8 characters at a minimum
    print( ’Invalid password’ )
elif tryI.isupper() or tryI.islower():
    # must have both upper- and lower-case letters
    print( ’Invalid password’ )
elif tryI.isalpha() or tryI.isdigit():
    # must have letters and numbers
    print( ’Invalid password’ )
else:
    print( ’Password OK’ )
```

But ’ !@#$%&* ’ will also be ok!

String Methods

So??? How to make sure that there are numbers as well as letters from the alphabet?

String Methods

So??? How to make sure that there are numbers as well as letters from the alphabet?
Add some more conditions!
M1.

```
if any(s in tryI.lower() for s
            in 'abcdefghijklmnopqrstuvwxyz'):
   # must have any of the letters from the alphabet
   if any(s in tryI for s in '0123456789'):
   # must have any numbers
      print( 'Password OK' )
```

No

String Methods

M2.

```
numE, lettE = 0, 0
for number in range(10):
    #find numbers, .find() returns -1 if not found
    if tryI.find(str(number)) != -1:
        numE = 1
        break
tryI = tryI.lower()
for letter in "abcdefghijklmnopqrstuvwxyz":
    if tryI.find(letter) != -1:
        lettE = 1
        break
if lettE == 1 and numE == 1:
    print("Password OK")
else:
     print( 'Invalid password' )
```

No

String Methods

M3

```
lettE = 0
lettCap = 0
lengthS = len(tryI)
for letter in tryI:
    if letter in 'abcdefghijklmnopqrstuvwxyz
                  ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789':
        lettE += 1
if not (tryI.isupper() or tryI.islower()):
        lettCap = 1
if lettE == lengthS and lengthS >= 8
                                    and lettCap > 0:
    print("Password OK")
else:
    print( 'Invalid password' )
```

OK

String comparison methods

These produce Boolean output.

`isdigit()`	Does a string contain only numbers?
`isalpha()`	Does a string contain only text?
`isalnum()`	what does this do?
`islower()`	Are **all the letters in a string** lower-case?
`isupper()`	Are **all the letters in a string** upper-case?

List Methods

```
Sort small to big (ascending order)
- x.sort()
Sort big to small (descending order)
- x.reverse()
```

List casting

```
list(range(5))
```

List casting

```
list(range(5))
```

= [0, 1, 2, 3, 4]

```
list('I am Happy')
```

List casting

```
list(range(5))
```

= [0, 1, 2, 3, 4]

```
list('I am Happy')
```

= ['I', ' ', 'a', 'm', ' ', 'H', 'a', 'p', 'p', 'y']

Function, Method, Attribute

Function e.g., `print()` etc

a piece of code that is called by name.
belongs to a library or your own code

Method e.g., `.isupper()` etc

also a function but special!
belongs to a data type

Attribute e.g., `.isupper(), .real` etc

property of the data type
can be a function or a value

Functions

Code blocks

```
try = input( ”Give me a password:  ” )
```

Code blocks

```
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’ )
```

Define a function

```
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 lett
        return False
    if try.isalpha() or try.isdigit():
        # must have letters and numbers
        return False
    return True  # password is OK
```

Code blocks become...

```
try = input( ”Give me a password:  ” )
ans = validate_password( try )
```

Use or Call a Function

```
output** = function ( input )
```

A *function* is a small program (block of code) we can run within Python.

Use or Call a Function

```
output** = function ( input )
```

A *function* is a small program (block of code) we can run within Python.

Saves us from rewriting code

Don’t reinvent the wheel!

Use or Call a Function

```
output** = function ( input )
```

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 ’complex’ verbs.

Use or Call a Function

```
output** = function ( input )
```

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 'complex' verbs.

Also called subroutine or procedure.

```
output** is the return value of the function
but it is optional
input is also called argument(s)
```

Function calls

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.

Function calls

When we want to execute a function, we call or invoke it.

Use name of the function with parentheses.

```
print( 'Exams are coming? Cannot wait
for me to score 100% !' )
```

Function calls

When we want to execute a function, we call or invoke it.

Use name of the function with parentheses.

```
print( 'Exams are coming? Cannot wait
for me to score 100% !' )
x = str( 10 )
```

Function calls

When we want to execute a function, we call or invoke it.

Use name of the function with parentheses.

```
print( 'Exams are coming? Cannot wait
for me to score 100% !' )
x = str( 10 )
```

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

Function calls

When we want to execute a function, we call or invoke it.

Use name of the function with parentheses.

```
print( 'Exams are coming? Cannot wait
for me to score 100% !' )
x = str( 10 )
```

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

Others we will create when needed.

Arguments

```
x = len('Exam is so exciting! I cannot breathe!')
```

Arguments are the input to functions.

Here, *argument* is `'Exam is so...'`

Functions can return a value.

Here, `x = len( )` returns a value of `38`

Return values are the output of a function. Here `x` stores the return value

Arguments

```
x = len('Exam is so exciting! I cannot breathe!')
```

Arguments are the input to functions.

Here, *argument* is `'Exam is so...'`

Functions can return a value.

Here, `x = len( )` returns a value of `38`

Return values are the output of a function. Here `x` stores the return value

Examples:

`x = len('Yong Chun')`, Which is the `return` value? Which is the argument?

Arguments

```
x = len('Exam is so exciting! I cannot breathe!')
```

Arguments are the input to functions.

Here, *argument* is `'Exam is so...'`

Functions can return a value.

Here, `x = len( )` returns a value of `38`

Return values are the output of a function. Here `x` stores the return value

Examples:

`x = len('Yong Chun')`, Which is the `return` value? Which is the argument?

`y = print('-123')`, y = ? (Any return value?)

Arguments

A function can accept **zero to many arguments**.

Arguments

A function can accept **zero to many arguments**.

```
print()
len('My String')
```

Multiple arguments are **separated by commas**:

```
min( 1,4,5 )
max( 1,4,5 )
```

Example: Type conversion

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

Example: Type conversion

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

```
float( ”0.13” )
str( 3 - 15j )
int( 0.223 )
```

Example: Type conversion

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

```
float( ”0.13” )
str( 3 - 15j )
int( 0.223 )
```

Be careful of nonsense:

```
int( ”Data” )
int( 0.37 + 2j )
```

Composing Functions

Example: Defining functions

```
def pow( a,b ):
    y = a ** b
    return y
```

Defining functions

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)

```
def pow( a,b ):
    y = a ** b
    return y
```

Parameters/Argument

Functions can accept values as arguments.

These variables are declared in the function header.
e.g.,

```
def print_message( msg ):
```

Multiple parameters are separated by commas.

```
def print_message( msg1, msg2, msg3 ):
    print( 'I say:', str(msg1)+str(msg2)+str(msg3))
```

return

Functions can return values with the keyword `return`.

```
def three():
    return 3
```

`return` immediately exits a function.

return

Functions can return values with the keyword `return`.

```
def three():
    return 3
```

`return` immediately exits a function.

```
def zero():
  return 0
  print('The answer is 0')
```

What happens to `print('The answer is 0')`?

return

Functions can return values with the keyword `return`.

```
def three():
    return 3
```

`return` immediately exits a function.

```
def zero():
  return 0
  print('The answer is 0')
```

What happens to `print('The answer is 0')`?
Ignored!

Scope

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.

Example: Defining functions

```
def pendulum( L ):
    import math
    g = 9.8  # m/s^2
    T = 2 * math.pi * math.sqrt( L / g )
    return T
time = pendulum( 1.0 )
print(g)
```

1. What is the value of `time`?
2. What will be shown on the screen?

Example: Defining functions

```
def pendulum( L ):
    import math
    g = 9.8  # m/s^2
    T = 2 * math.pi * math.sqrt( L / g )
    return T
time = pendulum( 1.0 )
print(g)
```

1. What is the value of `time`?
2. What will be shown on the screen?

ans:

Example: Defining functions

```
def pendulum( L ):
    import math
    g = 9.8  # m/s^2
    T = 2 * math.pi * math.sqrt( L / g )
    return T
time = pendulum( 1.0 )
print(g)
```

1. What is the value of `time`?
2. What will be shown on the screen?

ans:

1. value of `T` is approximately 2.0

Example: Defining functions

```
def pendulum( L ):
    import math
    g = 9.8  # m/s^2
    T = 2 * math.pi * math.sqrt( L / g )
    return T
time = pendulum( 1.0 )
print(g)
```

1. What is the value of `time`?
2. What will be shown on the screen?

ans:

1. value of `T` is approximately 2.0
2. Error

Example: Defining functions

```
a = 5
b = 6

def foo( c ):
    a = c
    b = c ** 2
    return a + b

print( foo( 6 ) )
print( a,b )
```

What will be printed?

Example: Defining functions

```
a = 5
b = 6

def foo( c ):
    a = c
    b = c ** 2
    return a + b

print( foo( 6 ) )
print( a,b )
```

What will be printed?

42

Example: Defining functions

```
a = 5
b = 6

def foo( c ):
    a = c
    b = c ** 2
    return a + b

print( foo( 6 ) )
print( a,b )
```

What will be printed?

42

5 6

Example

```
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.

Example

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

x = fun(2) * fun(3)
```

Example

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

x = fun(2) * fun(3)
```

```
x = 4 * fun(3)
```

Example

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

x = fun(2) * fun(3)
x = 4 * fun(3)
x = 4 * 5
```

Example

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

x = fun(2) * fun(3)
x = 4 * fun(3)
x = 4 * 5
x = 20
```

Example

```
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`)

Example

```
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)

Example

```
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)

Default Arguments

Default arguments

Functions can have default values available for certain arguments.

Default arguments

Functions can have default values available for certain arguments.

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

Default arguments

Functions can have default values available for certain arguments.

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

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

1. my_default_value()
2. my_default_value( 6 )
3. my_default_value( a=6 )  #Less used;
Note: variable used in 3. is the same
                        as defined in the function()
```

Exercise

Write a function `isclose` which assesses whether two float values `a` and `b` are sufficiently near each other to be consider ”equal”. It is equal if its relatve tolerance is less than or equal to 0.001.
(The relative tolerance is defined as $\frac{|a-b|}{\min(|a|,|b|)}$.)

Exercise

Write a function `isclose` which assesses whether two float values `a` and `b` are sufficiently near each other to be consider ”equal”. It is equal if its relatve tolerance is less than or equal to 0.001.
(The relative tolerance is defined as $\frac{|a-b|}{\min(|a|,|b|)}$.)

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

Fun time

```
def runningSum( a ):
    if a == 0:
        return 0
    f = a + runningSum(a-1)
    return f
tt = runningSum( 6 )
```

ans:

Fun time

```
def runningSum( a ):
    if a == 0:
        return 0
    f = a + runningSum(a-1)
    return f
tt = runningSum( 6 )
```

ans:
tt = 21!
This is called a recursive function

Summary

Summary

1. `function` can be `import` from library or written on your own
2. `Def .... ( arg1, arg2 ...):`
3. `return`
4. `Default value`
5. variable scope
6. `isclose()`