Introduction to Python

MRE/EME 5983 Robot Operating Systems

Overview

- Why Python?
- Getting started with Python
- Python overview
  - High level overview
  - Python data types, flow control and interacting with the user
- Python essentials
  - Deep drive into key concepts
  - Keywords, built-in functions, coding conventions, comments, variables and assignments, lists, tuples, classes and object-oriented programming
- This overview follows “The Quick Python Book” by Naomi Ceder

Why Python?

- There are many computer languages available: C/C++, C# and Java
- Python is a good choice to solve many problems and it is good choice to for individuals learning to program
- Truly cross-platform: Windows, Linux, Macintosh, supercomputers and cell phones
- It is free! And, many libraries are freely available: graphical, scientific and more

Source: The Quick Python Book by Naomi Ceder

Python Strengths

- Python is easy to learn and use
  - Supports all the typical constructs: loops, conditional statements, arrays, etc.
  - Python typically operates at a much higher level of abstraction
  - Syntax rules are very simple
- Python is expressive
  - A single line of Python code can do more than a single line of code in most other languages
- Python is readable
- Python is complete – “batteries included”
- Python is cross-platform
- Python is free

Source: The Quick Python Book by Naomi Ceder

Python Weaknesses

- Python is not the fastest language
  - This is typically the single greatest drawback of Python
- Python doesn't have the most libraries
  - Although Python has many libraries it does not have as many libraries of more well established languages such C and C++
- Python doesn't check variable types at compile time
  - This can be viewed as a drawback for traditional programmers

Source: The Quick Python Book by Naomi Ceder

Python Versions

- Python has three main versions: 1.x, 2.x and 3.x

- Python 3.x
  - First version of Python in the history of the language to break backward compatibility (Python 1.x runs in 2.x; Python 2.x may or may not run in 3.x)
  - Language is more consistent, more readable, and less ambiguous

- We will be learning and using Python 3.x in this course

Getting Started With Python

- Python is an interpreted language, what does this mean?
  - Some computer languages require you to compile the code before running the code. For example, C and C++
  - Python is compiled at run time = Syntax errors are caught at run time

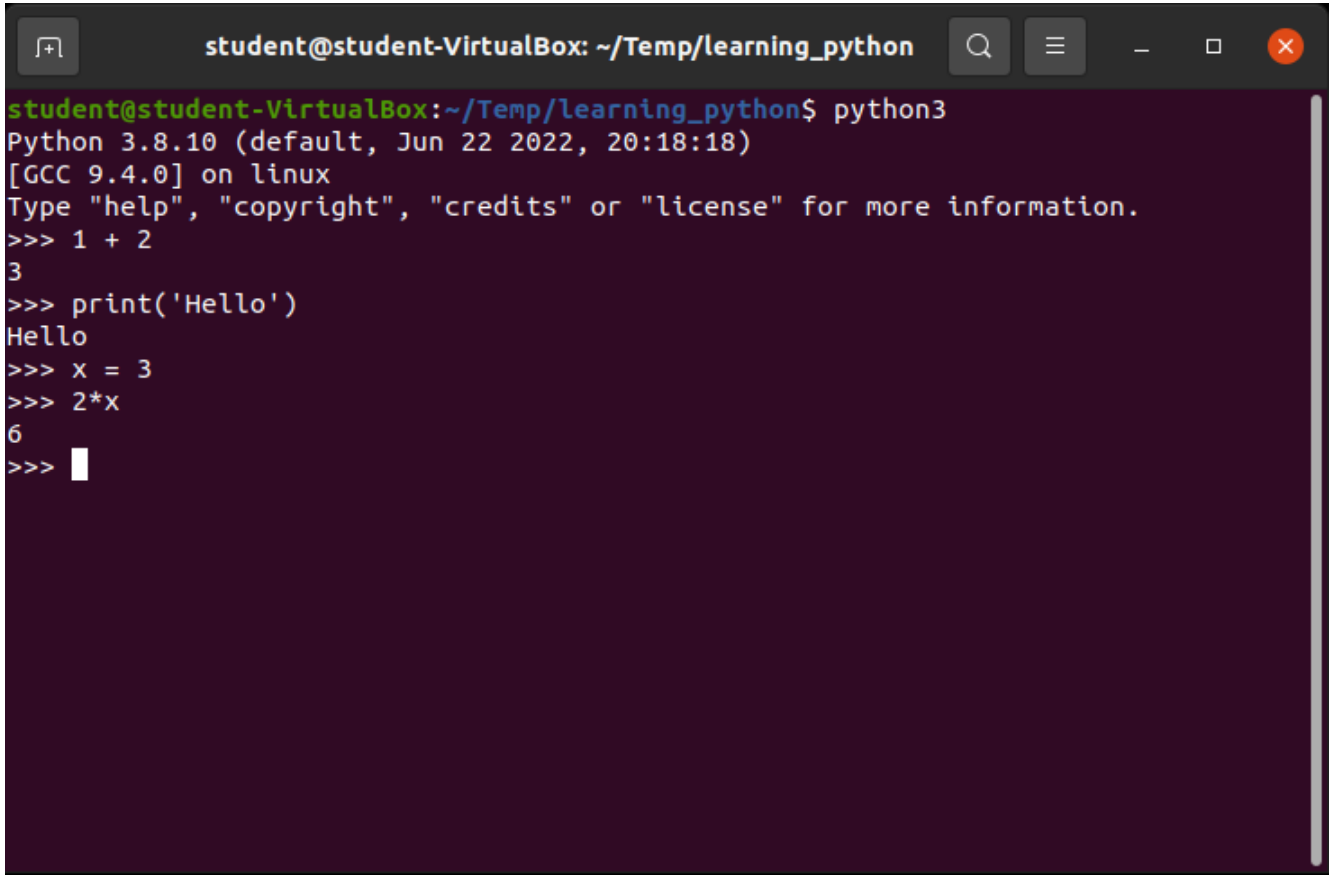
- Python has two methods of developing and executing code
  - Basic interactive mode
  - Execute a Python file as a script

Source: The Quick Python Book by Naomi Ceder

Getting Started With Python

- Basic interactive mode (run from Ubuntu terminal)

  $ python3

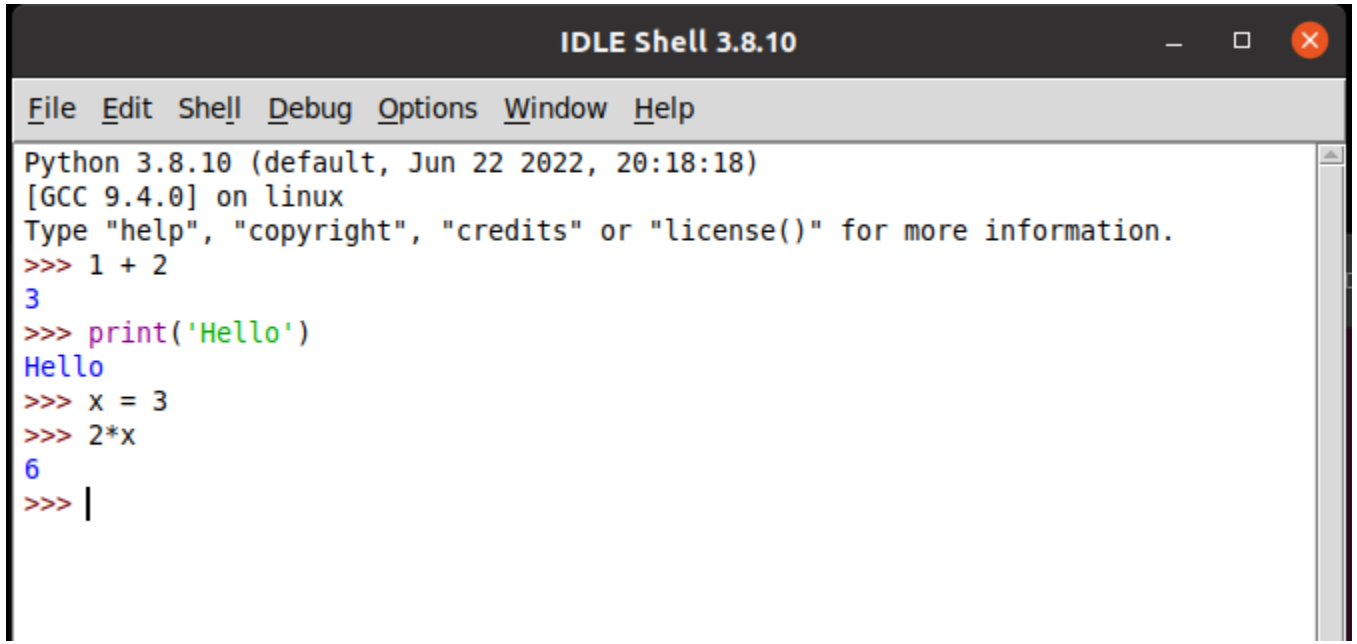


Note: To exit the Python interactive mode, use exit() or Ctrl-D

Getting Started With Python

- Basic interactive mode (using IDLE)
  - IDLE = **I**ntegrated **D**eve**L**opment **E**nvironment

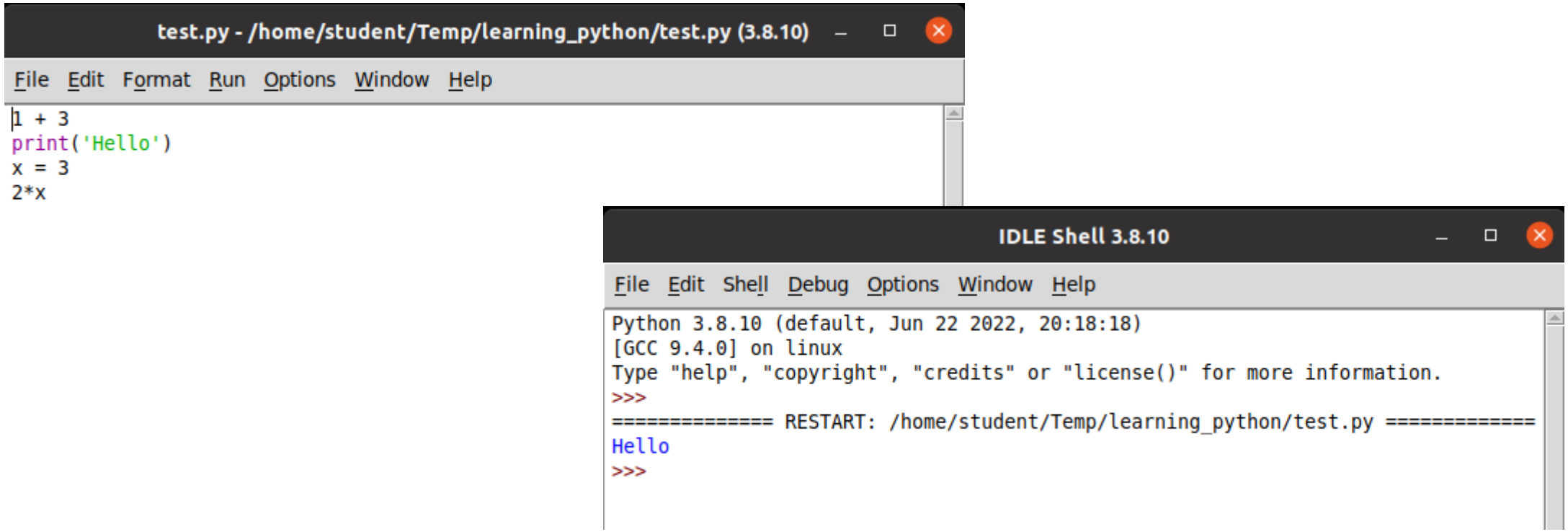
  $ idle



Note: To exit use exit() or Ctrl-D from the command prompt or File -> Exit

Getting Started With Python

- Executing a Python file as a script
  - Need to create a .py file containing Python commands
  - One method, use IDLE to create and run a script file



Python Overview: Python Data Types

- Some Python data types
  - Numeric data types: *int, float, complex*
  - Boolean type: *bool*
  - String data types: *str*
  - Sequence types: *list, tuple, range*
- Note that Python data types are associated with the object, not the variable

Python Overview: Python Data Types

- Numerical data types
  - Integers: 5, -10, 1,000,000, 0
  - Float: 2.0, 7.321, 3.1415926535, -6.022e23
  - Complex: 1 + 2j, -3.3 -4.7j

Operation	Result
`x + y`	sum of *x* and *y*
`x - y`	difference of *x* and *y*
`x * y`	product of *x* and *y*
`x / y`	quotient of *x* and *y*
`x // y`	floored quotient of *x* and *y*
`x % y`	remainder of `x / y`
`-x`	*x* negated
`+x`	*x* unchanged
`abs(x)`	absolute value or magnitude of *x*
`int(x)`	*x* converted to integer
`float(x)`	*x* converted to floating point
`complex(re, im)`	a complex number with real part *re*, imaginary part *im*. *im* defaults to zero.
`c.conjugate()`	conjugate of the complex number *c*
`divmod(x, y)`	the pair `(x // y, x % y)`
`pow(x, y)`	*x* to the power *y*
`x ** y`	*x* to the power *y*

Source: https://docs.python.org/3/library/stdtypes.html#text-sequence-type-str

Python Overview: Python Data Types

- Boolean data type
  - Booleans can be assigned **True** or **False**

Boolean Operations — `and`, `or`, `not`

These are the Boolean operations, ordered by ascending priority:

Operation	Result
`x or y`	if *x* is false, then *y*, else *x*
`x and y`	if *x* is false, then *x*, else *y*
`not x`	if *x* is false, then `True`, else `False`

Comparisons

Operation	Meaning
`<`	strictly less than
`<=`	less than or equal
`>`	strictly greater than
`>=`	greater than or equal
`==`	equal
`!=`	not equal
`is`	object identity
`is not`	negated object identity

Source: https://docs.python.org/3/library/stdtypes.html#text-sequence-type-str

Python Overview: Python Data Types

- String data
  - Strings are immutable sequences of Unicode code points
  - String literals can be written a few ways:
    - Single quotes: 'allows embedded "double" quotes'
    - Double quotes: "allows embedded 'single' quotes"
    - Triple quoted: '''Three single quotes''', """Three double quotes"""

Source: https://docs.python.org/3/library/stdtypes.html#text-sequence-type-str

Python Overview: Python Data Types

- Python strings can contain special characters, called escape characters

```
print('Single quotes in a \'single quote\' string')

print('\n')

print('Space\tOut\tData')

print('\n')

print('A in octal is 101 and hex is 41: \101 / \x41')
```

```
Single quotes in a 'single quote' string

Space	Out	Data

A in octal is 101 and hex is 41: A / A
```

Code	Description
\'	Single Quote
\"	Double Quote
\\	Backslash
\n	New Line
\r	Carriage Retrun
\t	Tab
\b	Backspace
\f	Form Feed
\ooo	Octal Value
\xhh	Hex Value

Source: https://pythonexamples.org/python-escape-characters/

Python Overview: Python Data Types

- Sequence types: *list, tuple, range*
- List construction
  - Using a pair of square brackets to denote the empty list: []
  - Using square brackets, separating items with commas: [a], [a, b, c]
  - Using a list comprehension: [x for x in iterable]
  - Using the type constructor: list() or list(iterable)
- Tuple construction
  - Using a pair of parentheses to denote the empty tuple: ()
  - Using a trailing comma for a singleton tuple: a, or (a,)
  - Separating items with commas: a, b, c or (a, b, c)
  - Using the tuple() built-in: tuple() or tuple(iterable)
- Range construction
  - range(stop)
  - range(start, stop[, step])

Source: https://docs.python.org/3/library/stdtypes.html#text-sequence-type-str

Python Overview: Python Data Types

- Sequence types: *list, tuple, range*
  - All list, tuple and range indexing begins at 0
  - The last item in the list, tuple or range is (length of the sequence – 1)
  - You may also use negative indexing (backwards and wrap)

```
x = [1,2,3,4]
y = (1,2,3,4)
z = range(1,5)

print('Results:')
print(x)
print(y)
print(z)

print('')
print('x 1st element: ', x[0])
print('y 1st element: ', y[0])
print('z 1st element: ', z[0])

print('')
print('x last element: ', x[3])
print('y last element: ', y[3])
print('z last element: ', z[3])

print('')
print('x last element: ', x[-1])
print('y last element: ', y[-1])
print('z last element: ', z[-1])
```

```
Results:
[1, 2, 3, 4]
(1, 2, 3, 4)
range(1, 5)

x 1st element:  1
y 1st element:  1
z 1st element:  1

x last element:  4
y last element:  4
z last element:  4

x last element:  4
y last element:  4
z last element:  4
```

Python Overview: Python Data Types

- Sequence types: *list, tuple, range*

Common Sequence Operations

Operation	Result
`x in s`	`True` if an item of *s* is equal to *x*, else `False`
`x not in s`	`False` if an item of *s* is equal to *x*, else `True`
`s + t`	the concatenation of *s* and *t*
`s * n` or `n * s`	equivalent to adding *s* to itself *n* times
`s[i]`	*i*th item of *s*, origin 0
`s[i:j]`	slice of *s* from *i* to *j*
`s[i:j:k]`	slice of *s* from *i* to *j* with step *k*
`len(s)`	length of *s*
`min(s)`	smallest item of *s*
`max(s)`	largest item of *s*

Source: https://docs.python.org/3/library/stdtypes.html#text-sequence-type-str

Python Overview: Python Data Types

- Sequence types: *list, tuple, range*

Mutable Sequence Types

Operation	Result
`s[i] = x`	item *i* of *s* is replaced by *x*
`s[i:j] = t`	slice of *s* from *i* to *j* is replaced by the contents of the iterable *t*
`del s[i:j]`	same as `s[i:j] = []`
`s[i:j:k] = t`	the elements of `s[i:j:k]` are replaced by those of *t*
`del s[i:j:k]`	removes the elements of `s[i:j:k]` from the list
`s.append(x)`	appends *x* to the end of the sequence (same as `s[len(s):len(s)] = [x]`)
`s.clear()`	removes all items from *s* (same as `del s[:]`)
`s.copy()`	creates a shallow copy of *s* (same as `s[:]`)
`s.extend(t)` or `s += t`	extends *s* with the contents of *t* (for the most part the same as `s[len(s):len(s)] = t`)
`s *= n`	updates *s* with its contents repeated *n* times
`s.insert(i, x)`	inserts *x* into *s* at the index given by *i* (same as `s[i:i] = [x]`)
`s.pop()` or `s.pop(i)`	retrieves the item at *i* and also removes it from *s*
`s.remove(x)`	remove the first item from *s* where `s[i]` is equal to *x*
`s.reverse()`	reverses the items of *s* in place

Source: https://docs.python.org/3/library/stdtypes.html#text-sequence-type-str

Python Overview: Python Flow Control

- Python has a full range of structures to control code execution and program flow, including common branching and looping structure
  - Boolean values and expressions
  - if-elif-else statement
  - while loop
  - for loop
  - Function definition
  - Module creation

Source: The Quick Python Book by Naomi Ceder

Python Overview: Python Flow Control

- Boolean values and expressions
- You can create comparison expressions using
  - comparison operators (<, <=, ==, >, >=, !=, is, is not, in, not in)
  - logical operators (and, not, or)
  - All return **True** or **False**

Boolean Operations — `and`, `or`, `not`

These are the Boolean operations, ordered by ascending priority:

Operation	Result
`x or y`	if *x* is false, then *y*, else *x*
`x and y`	if *x* is false, then *x*, else *y*
`not x`	if *x* is false, then `True`, else `False`

Comparisons

Operation	Meaning
`<`	strictly less than
`<=`	less than or equal
`>`	strictly greater than
`>=`	greater than or equal
`==`	equal
`!=`	not equal
`is`	object identity
`is not`	negated object identity

Source: https://docs.python.org/3/library/stdtypes.html#text-sequence-type-str

Python Overview: Python Flow Control

- if-elif-else statement

```
x = 5
if x < 5:
    y = -1
elif x > 5:
    y = 1
else:
    y = 0
print('Results:')
print(x, y)


Results:
5 0
```

```
x = 6
if x < 5:
    y = -1
elif x > 5:
    y = 1
else:
    y = 0
print('Results:')
print(x, y)


Results:
6 1
```

```
x = 4
if x < 5:
    y = -1
elif x > 5:
    y = 1
else:
    y = 0
print('Results:')
print(x, y)


Results:
4 -1
```

- Python uses indentation to delimit blocks. No explicit delimiters such as brackets or braces are necessary. Each block consists of one or more statements separated by newlines. These statements must all be at the same level of indentation.

Source: The Quick Python Book by Naomi Ceder

Python Overview: Python Flow Control

- while loop

```
x = 0
x_sum = 0

while x < 6:
    x_sum = x_sum + x
    x = x + 1

print('Results:')
print(x, x_sum)
```

```
Results:
6 15
```

- Similar to the if-elif-else statement, the while loop uses indentation to delimit blocks

Python Overview: Python Flow Control

- for loop

```
x_sum = 0

for x in range(6):
    x_sum = x_sum + x

print('Results:')
print(x, x_sum)
```

```
Results:
5 15
```

```
x_list = [1, 2, 'three', 4]

for x in x_list:
    print(x)
```

```
1
2
three
4
```

- The for loop uses indentation to delimit blocks

Python Overview: Python Flow Control

- Function definition
  - Functions are defined using the **def** statement
  - The return statement is what a function uses to return a value
  - If no return statement is encountered, Python's **None** value is returned

```
def sample_fun( x, y, z=1 ):
    value = (x**2 + y**2)**0.5
    return z*value



print('Results')

val = sample_fun(3, 4)
print(val)

val = sample_fun(3, 4, 3)
print(val)
```

```
Results
5.0
15.0
```

- Functions use indentation to delimit function contents

Python Overview: Python Flow Control – Module Creation

quad.py

```
#!/usr/bin/python3

def quadratic(a, b, c):
    # Check input for x^2 coefficient
    if( a == 0.0 ):
        return None

    # Compute roots
    x = -b
    y = (b*b - 4.0*a*c)**0.5
    roots = [ (x-y)/(2*a), (x+y)/(2*a)]
    return roots

if __name__ == '__main__':
    print('sample_module loaded...')
```

IDLE Terminal

```
>>> from quad import quadratic
>>> quadratic(1,0,-1)
[-1.0, 1.0]
```

Python Overview: Interacting With The User

- Python provides a mechanism to display information to the user **print()** and get input from the user **input()**

- The input() function reads a line from input, converts it to a string (strips a trialing newline), and returns the input in a string

- The input function also has an optional argument for a prompt string. If the prompt argument is provided, standard output without a trailing newline

```
name = input('Enter your name: ')
print('Results:')
print(name)
```

```
Enter your name: Bob
Results:
Bob
```

Python Overview: Interacting With The User

- We have used the print() function to print strings and variables
- Now, we will learn how to format the string that print() function outputs
- There are a few methods for print formatting
  - format method
  - f strings
  - % operator method

Python Overview: Interacting With The User

- Formatting strings with the format method

```
out_str = 'Hello, {}'.format('Bob')
print(out_str)

out_str = 'The sum of {} and {} is {}'.format(1,2,3)
print(out_str)

out_str = 'The sum of {} and {} is {}'.format(1.5,2,3.5)
print(out_str)

print('Hello, {}!  The sum of {} and {} is {}'.format('Bob',4,5,9))
```

```
Hello, Bob
The sum of 1 and 2 is 3
The sum of 1.5 and 2 is 3.5
Hello, Bob!  The sum of 4 and 5 is 9
```

Python Overview: Interacting With The User

- Formatting strings with f strings

```
a = 1
b = 2
print(f'The sum {a} and {b} is {a+b}')

a = 1.5
b = 2
print(f'The sum {a} and {b} is {a+b}')

x = [1, 'two', 3]
y = 'Bob'
z = True
print(f'You can print a list {x}, a string ({y}) and a boolean ({z})')
```

```
The sum 1 and 2 is 3
The sum 1.5 and 2 is 3.5
You can print a list [1, 'two', 3], a string (Bob) and a boolean (True)
```

Python Overview: Interacting With The User

- Formatting strings with the % operator

```
print('The sum of %d and %d is %d' % (1,2, 3))

print('The sum of %.2f and %.1f is %f' % (1.5, 2.0, 3.5))

print('Hello %s!  Here is a boolean value %r.' % ('Bob', True) )
```

```
The sum of 1 and 2 is 3
The sum of 1.50 and 2.0 is 3.500000
Hello Bob!  Here is a boolean value True.
```

Python Essentials

Python Essentials: Keywords

- The following identifiers are used as reserved words, or keywords of the language, and cannot be used as ordinary identifiers

```
False      await      else       import     pass
None       break      except     in         raise
True       class      finally    is         return
and        continue   for        lambda     try
as         def        from       nonlocal   while
assert     del        global     not        with
async      elif       if         or         yield
```

Source: https://docs.python.org/3.11/reference/lexical_analysis.html#keywords

Python Essentials: Built-in Functions

- Python built-in functions
  - So far, we have used
    - print()
    - range()
    - format()
  - We will use more moving forward

Built-in Functions

A	**E**	**L**	**R**
abs()	enumerate()	len()	range()
aiter()	eval()	list()	repr()
all()	exec()	locals()	reversed()
any()			round()
anext()	**F**	**M**	
ascii()	filter()	map()	**S**
	float()	max()	set()
B	format()	memoryview()	setattr()
bin()	frozenset()	min()	slice()
bool()			sorted()
breakpoint()	**G**	**N**	staticmethod()
bytearray()	getattr()	next()	str()
bytes()	globals()		sum()
		O	super()
C	**H**	object()	
callable()	hasattr()	oct()	**T**
chr()	hash()	open()	tuple()
classmethod()	help()	ord()	type()
compile()	hex()		
complex()		**P**	**V**
	I	pow()	vars()
D	id()	print()	
delattr()	input()	property()	**Z**
dict()	int()		zip()
dir()	isinstance()		
divmod()	issubclass()		**_**
	iter()		__import__()

Source: https://docs.python.org/3/library/functions.html

Python Essentials: Coding Conventions

- For readability, Python is typically written with “Pythonic coding conventions”
- When possible, please try to follow these guidelines

Situation	Suggestion	Example
Module/package names	short, all lowercase, underscores only if needed	`imp, sys`
Function names	all lowercase, underscores_for_readablitiy	`foo(), my_func()`
Variable names	all lowercase, underscores_for_readablitiy	`my_var`
Class names	CapitalizeEachWord	`MyClass`
Constant names	ALL_CAPS_WITH_UNDERSCORES	`PI, TAX_RATE`
Indentation	4 spaces per level, don’t use tabs	
Comparisons	Don’t compare explicitly to True or False	`if my_var:` `if not my_var:`

Source: The Quick Python Book by Naomi Ceder

Python Essentials: Comments

- Comments are an important part of writing good code
  - Helps others (and potentially yourself) understand your code
- Python offers two mechanisms of supporting comments
  - In-line comments
    - Use # to indicate a comment
    - Text following # on a given line is a comment
  - Document strings
    - Place comments between a pair of triple quotes “””

```
# Using a comment...
x = [ 1, 2, 3]    # we can a comment here too
```

```
""" This is how we can
put longer comments that
span  mutiple lines
"""
```

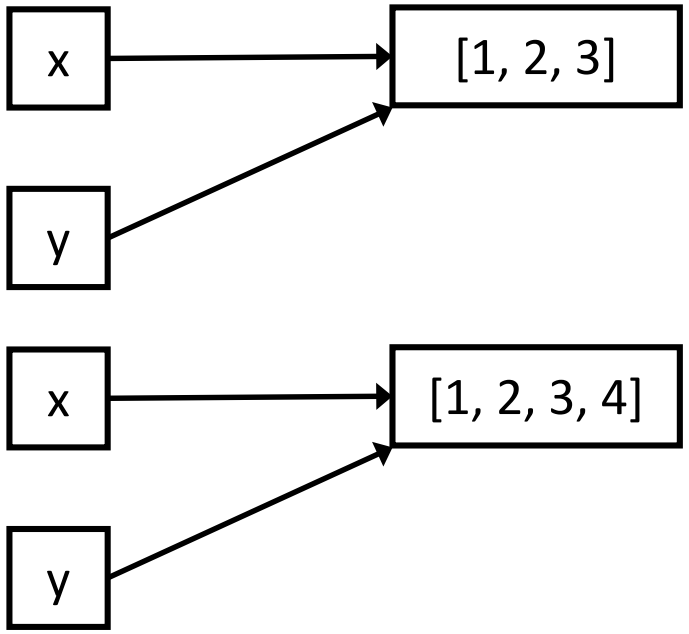
```
# FYI - we can use this method
# for longer comments that
# span mutiple lines
```

Python Essentials: Variables and Assignments

- A Python variable is a **symbolic name** that is a **reference or pointer** to an **object**

- Example

```
>>> x = [1, 2, 3]
>>> y = x
>>> x
[1, 2, 3]
>>> y
[1, 2, 3]
>>> x.append(4)
>>> x
[1, 2, 3, 4]
>>> y
[1, 2, 3, 4]
```



Python Essentials: Variables and Assignments

- The behavior observed in the previous example, should be expected for any in place operation, for example

```
>>> a = [6, 4, 1]
>>> b = a
>>> a.sort()
>>> a
[1, 4, 6]
>>> b
[1, 4, 6]
```

- This behavior will not occur for operations that are not in-place

```
>>> a = [1, 2, 3]
>>> b = a
>>> a
[1, 2, 3]
>>> b
[1, 2, 3]
>>> a = [1, 2, 3, 4]
>>> a
[1, 2, 3, 4]
>>> b
[1, 2, 3]
```

Python Essentials: Lists Overview

- Lists may be considered the widely used data structures in Python

- List are like arrays in C, C++ or Java (but much more flexible!)

```
# This assigns a three-element list to x
x = [1, 2, 3]
```

- When creating a list, you do not need to worry about:
  - Declaring a type
  - Declaring the length
  - “lists automatically grow or shrink in size as needed”

- Note, Python has an array module that provides C type arrays

Source: The Quick Python Book by Naomi Ceder

Python Essentials: Lists Overview

- Python lists can contain different types of elements; a list element can be any Python object
- Here's a list that contains a variety of elements:

```
# First element is a number, second is a string, third is another list.
x = [2, "two", [1, 2, 3]]
```

- Probably the most basic built-in list function is the **len()** function, which returns the number of elements in a list:

```
>>> x = [2, "two", [1, 2, 3]]
>>> len(x)
3
```

- Note that the **len()** function does not count the items in the inner, nested list.

Source: The Quick Python Book by Naomi Ceder

Python Essentials: List Indexing and Slicing

Python Essentials: List Modifications

list_modify.py

```
# Define list
x = [0, 1, 2, 3]
print('1) ', x)
print('')

# Append another element
x.append(4)
print('2) ', x)
print('')

# Append another list
y = [5, 6, 7]
print('3) ', x+y)
print('')
x.extend(y)
print('4) ', x)
print('')

# Remove elements
x[0:-4] = []
print('5) ', x)
print('')

# Append an element to the beginning of the list
x[:0] = [0,1]
print('6) ', x)
print('')

# Inserting in the middle
x.insert(2,3)
x.insert(2,2)
print('7) ', x)
```

IDLE Terminal

```
1)  [0, 1, 2, 3]

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

3)  [0, 1, 2, 3, 4, 5, 6, 7]

4)  [0, 1, 2, 3, 4, 5, 6, 7]

5)  [4, 5, 6, 7]

6)  [0, 1, 4, 5, 6, 7]

7)  [0, 1, 2, 3, 4, 5, 6, 7]
```

Lists are mutable data structures – we can change the contents and number of elements

Python Essentials: List Operations

list_operations_1.py

```
# Sort using Python sort
x = [4, 1, 3, 2]
x.sort()
print('1) ', x, '\n')

# Sort using lexicographic order
x = ['Hello', 'Bob', 'How', 'are', 'you']
x.sort()
print('2) ', x, '\n')

# in operator to search a list
x = [1, 2, 3, 4]
found = 3 in x
print('3) ', found, '\n')
found = 5 in x
print('4) ', found, '\n')

# List construction
x = [1]*10
print('5) ', x, '\n')
x = list(range(10))
print('6) ', x, '\n')

# Find the minimum and maximum values
print('7) ', min(x), '\n')
print('8) ', max(x), '\n')
```

IDLE Terminal

```
1)  [1, 2, 3, 4]

2)  ['Bob', 'Hello', 'How', 'are', 'you']

3)  True

4)  False

5)  [1, 1, 1, 1, 1, 1, 1, 1, 1, 1]

6)  [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

7)  0

8)  9
```

Note that the sort operation is performed “in-place”
This means that the operation modifies the actual object

Python Essentials: List Operations

list_operations_2.py

```
# List search with index - index argument must be in the list!
x = [4, 3, 7, -5, 1]
idx = x.index(-5)
print('1) ', idx, '\n')

# Sort using lexicographic order
x = [1, 2, 2, 1, 5, 2, 5]
num_twos = x.count(2)
num_threes = x.count(3)
print('2) Number of twos = ', num_twos)
print('3) Number of threes = ', num_threes, '\n')

# Reversing a list
x = [1, 2, 3, 4]
print('4) ', x )
x.reverse()
print('5) ', x )
```

IDLE Terminal

```
1)  3

2) Number of twos =  3
3) Number of threes =  0

4)  [1, 2, 3, 4]
5)  [4, 3, 2, 1]
```

Note that the reverse operation is performed in-place

Python Essentials: List Operations

List operation	Explanation	Example
`[]`	Creates an empty list	`x = []`
`len`	Returns the length of a list	`len(x)`
`append`	Adds a single element to the end of a list	`x.append('y')`
`insert`	Inserts a new element at a given position in the list	`x.insert(0, 'y')`
`del`	Removes a list element or slice	`del(x[0])`
`remove`	Searches for and removes a given value from a list	`x.remove('y')`
`reverse`	Reverses a list in place	`x.reverse()`
`sort`	Sorts a list in place	`x.sort()`
`+`	Adds two lists together	`x1 + x2`
`*`	Replicates a list	`x = ['y'] * 3`
`min`	Returns the smallest element in a list	`min(x)`
`max`	Returns the largest element in a list	`max(x)`
`index`	Returns the position of a value in a list	`x.index['y']`
`count`	Counts the number of times a value occurs in a list	`x.count('y')`
`in`	Returns whether an item is in a list	`'y' in x`

Source: The Quick Python Book by Naomi Ceder

Python Essentials: Tuples

- Python tuples are very similar to lists; however tuples are immutable
  - Immutable = unchanging over time or unable to be changed

- Tuples can be consider as constant lists

tuple.py

```
# Define a tuple
x = (1, 2, 3)

# Access element of the tuple
print('x[1] = ', x[1] )

# Try to update the tuple
x[1] = -1
```

IDLE Terminal

```
x[1] =  2
Traceback (most recent call last):
  File "/usr/lib/python3.8/idlelib/run.py", line 559, in runcode
    exec(code, self.locals)
  File "/home/student/Temp/learning_python/tuple.py", line 8, in <module>
    x[1] = -1
TypeError: 'tuple' object does not support item assignment
```

Python Essentials: Classes and Object-Oriented Programming

- Object-oriented programming (OOP) is a computer programming model that organizes software design around data, or objects, rather than functions and logic
- In Python, call data types are implemented as classes
- For example, we can use classes to create a new data type that overrides mathematical operations
- In this course we will leverage classes and OOP to create containers to store data and methods (a simplified use of classes)

Python Essentials: Classes and Object-Oriented Programming

class_circle.py

```
#!/usr/bin/env python3
import math

# Circle class example
class Circle():
    # Set attributes
    unit = 'cm'

    # Instance attribute
    def __init__(self, xc, yc, radius):
        self.x_center = xc
        self.y_center = yc
        self.radius = radius

    # Methods
    def area(self):
        a = math.pi * self.radius**2.0
        return a

    def cylinder_vol(self, height):
        volume = self.area() * height
        return volume

if __name__ == '__main__':
    # Define a circle object
    my_circle = Circle(1, 2, 1)
    print('Circle area = %.2f %s^2' % (my_circle.area(), my_circle.unit) )
    print('Cylinder Volume = %.2f %s^3' % ( my_circle.cylinder_vol(2.0),
                                            my_circle.unit) )
```

IDLE Terminal

```
Circle area = 3.14 cm^2
Cylinder Volume = 6.28 cm^3
```

Summary

- We learned about Python and how to write Python programs
- Python will be the main programming language used in this course