PYTHON

LANGUAGE SYNTAX AND PROGRAMMING CONSTRUCTS

Gene Olafsen

JUPYTER NOTEBOOK

- This slide deck is accompanied by a Jupyter notebook file containing examples of language concepts.
- Download and install Anaconda for Python 3 and Jupyter notebook support.
  - https://www.anaconda.com/download/
- The accompanying notebook can be found at:
  - https://github.com/geneostrat/Metrowest-Developers-Machine-Learning-Group

NOTEBOOK

jupyter Introduction to Python Last Checkpoint: a minute ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Code

Introduction to Python

This notebook provides an overview of Python language syntax and program structures. If you can duplicate these examples on your own, it's probably safe to say that you know how to program in the Python language. Of course there are many libraries that extend Python, but the language fundamentals remain the same.

Each of the samples below can be run standalone- there are no dependencies on any previous code fragments. This provides you a self-contained playground to experiment with each concept. The examples strive to convey each concept using a minimal lines of code. (a local minimum- not the global minimum)

In [171]:
```
# PLS1-COM
# comments
# single line comment or
"""
    multi-
    line
    comment
"""
# another single line comment
```

Out[171]:
```
'\n    multi-\n    line \n    comment\n'
```

In [3]:
```
# variables do not have to be declared before use
A = 1 + 2
C = 4.0 + A
print(A, C)
```

```
hello 7.0
```

NOTEBOOK CELL IDENTIFIER PLS1-COM

- An example identifier comment is placed at the top of each referenced cell. (highlighted below)
- Slide titles will also have the cell identifier (see above)

```
In [171]: # PLS1-COM
          # comments
          # single line comment or
          """
              multi-
              line
              comment
          """
          # another single line comment
```

COMMENTS

PLS1-COM

- Single and multi-line comments are supported

```
In [171]: # PLS1-COM
          # comments
          # single line comment or
          """
              multi-
              line
              comment
          """
          # another single line comment
```

VARIABLES

PLS1-VAR1-3

- Variable type is established at initialization.
- A variable may be assigned to a different type at any time.
- Multiple variables may be initialized at once.

DELETE

PLS1-DEL

- The *DEL* command is used to delete variables or any other object. Even though Python has had a garbage collector since version 2, this command provides a mechanism to manage memory directly.

```
# PLS1-DEL
# delete a variable (or any Python object, such as a list)
A = "hello world"
print(A)
del A
print(A)
```

PRINT

PLS1-PRINT1,2

- The *PRINT* command outputs to the Python console.
- Multiple values may be printed on the same line separated by a comma.
- The newline character \n forces subsequent print statements to the next line.
- A print statement with an end="" argument allows the next print statement to appear on the same line.

SPANNING LINES PLS1-CONT1,2

- Use the backslash character '\' to span both Python expression and *PRINT* statements.

```
# PLS1-CONT1
# line continuation character is the backslash
string_one    \
 = "separate \
lines"
print(string_one)
```

INDENTATION

PLS1-IND

- Python requires all statements within a block to be indented the same amount.
- Python indentation rules equate to the use of curly brackets in C/C++/C#/Java, etc.
- It can be argued that indentation is less ambiguous than using curly brackets.

INPUT

PLS1-INP

- The *INPUT* command is used to solicit text from the console.
- The command accepts a prompt string argument.

```
# PLS1-INP
# solicit input
my_input = input("Enter some text: ")
print("You entered: ", my_input)
```

IMMUTABLE TYPES

PLS1-IMT

- Boolean, integer, long, float, tuple, string, complex-number and frozenset.
- A type is immutable if it's value is fixed upon initialization.
- Changing the assignment of a variable causes Python to re-evaluate the type associated with that variable.

TYPE CONVERSION

PLS1-TC

- Python supports type conversion.

```
# PLS1-TC
# type conversion
i = 4
f = 4.8

int_as_float = float(i)
float_as_int = int(f)
print("int as float: ", int_as_float)
print("float as int: ", float_as_int)  # NOTE: no rounding up
```

TUPLES AND LISTS

PLS1-TL1,2

- Tuples and Lists are collection objects.
- Tuples are immutable and defined with the collection contained in parenthesis.
- Lists are dynamically defined and initialized with a collection contained in square-brackets.
- Tuples and Lists collections may contain a mix of types.
- Lists support more operations (insert, delete, etc.) however they are slower than *tuples*.

REFERENCES

PLS1-REF

```
# immutable and reference example
tuple1 = (1,2,3)
tuple2 = tuple1     # a reference to tuple1 is established UNTIL either tuple1 or tuple2 changes
print(tuple2 is tuple1) # vefify that tuple1 and tuple2 reference the same object
tuple1 = (1,2,3,4) # you should NOT expect t2 to change, a new instance of tuple1 is established

list1 = [1,2,3]
list2 = list1    # a reference to list1 is established
list2.append(4) # you can expect a1 to change

print ("tuple1: ", tuple1, " tuple2: ", tuple2)
print ("list1: ", list1, " list2: ", list2)
```

SLICING

PLS1-SLC1,2

- Square brackets and a colon are used to contain and separate the range for both Tuples and Lists.
  - [start-index:length]
- New objects are returned by slice operations.

COMMON TUPLE AND LIST OPERATORS PLS1-OP1,2

- Operators include:
  - + add two collections together
  - * multiple the collections
- Some operations occur in-place
  - Sort, Reverse, Random

DICTIONARY

PLS1-DCT

- Dictionary is a collection of key-value pairs.
- Dictionary keys and values can be of any type.
- KEY:VALUE pairs are contained in {} and separated by a :
  - my_dictionary = {1:'one', 2:'two', 3:'three'}
- Items may be added, removed or changed.

FUNCTIONS

PLS1-FN1,2

- Python functions are objects.
- Functions may return multiple values/objects.

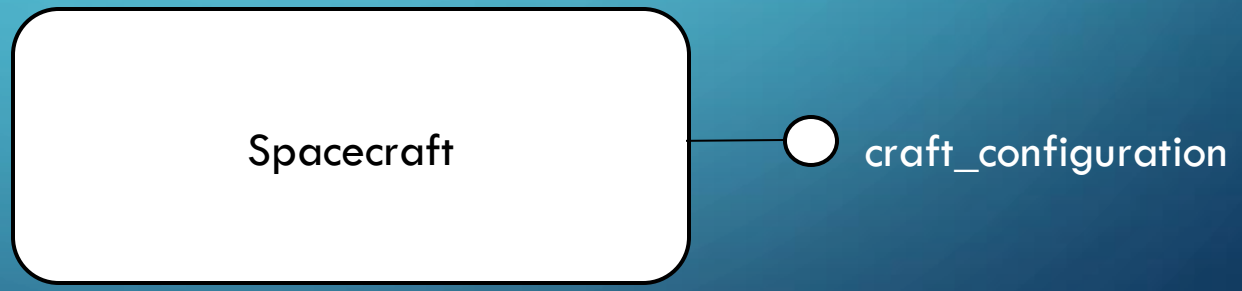
```
def my_first_function(my_arg1, my_arg2):
    print("my_arg1: ", my_arg1)
    print("my_arg2: ", my_arg2)
    return "hello world"

my_first_function(3, 'code')
```

FUNCTION ARGUMENTS PLS1-ARG1,2,3

- Function arguments that are value types will not be changed for the caller if changed within a function.
- Function arguments that are passed by reference (lists, etc.) may be changed within the function and changed for the caller as well.
- Python supports default argument values.

FIRST CLASS

- Python supports classes
  - Abstraction
  - Encapsulation
  - Inheritance



CLASS

PLS1-CL1

- __init__ is the constructor name for all Python classes.
- A colon ':' identifies the beginning of an indentation block.

```
class Spacecraft(object) :
    def __init__(self, name, power_source):
        self.name = name
        self.power_source = power_source
    def craft_configuration(self):
        print("name: ", self.name)
        print("power_source: ", self.power_source)
        print("power_status: operational")
        return
```

CLASS INTROSPECTION

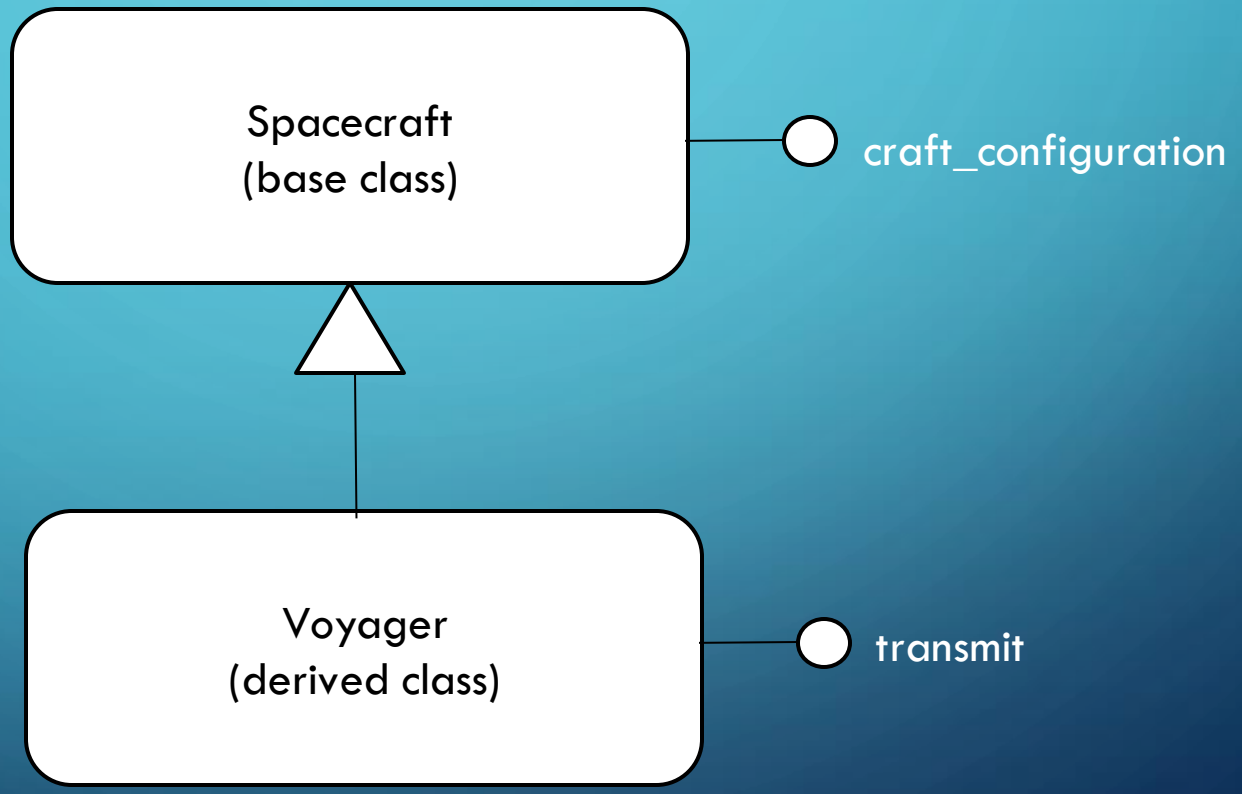
PLS1-CL2

- __dict__
  - Used to access the class attribute dictionary
- hasattr
- getattr
- setattr

SUBCLASS

PLS1-CL3

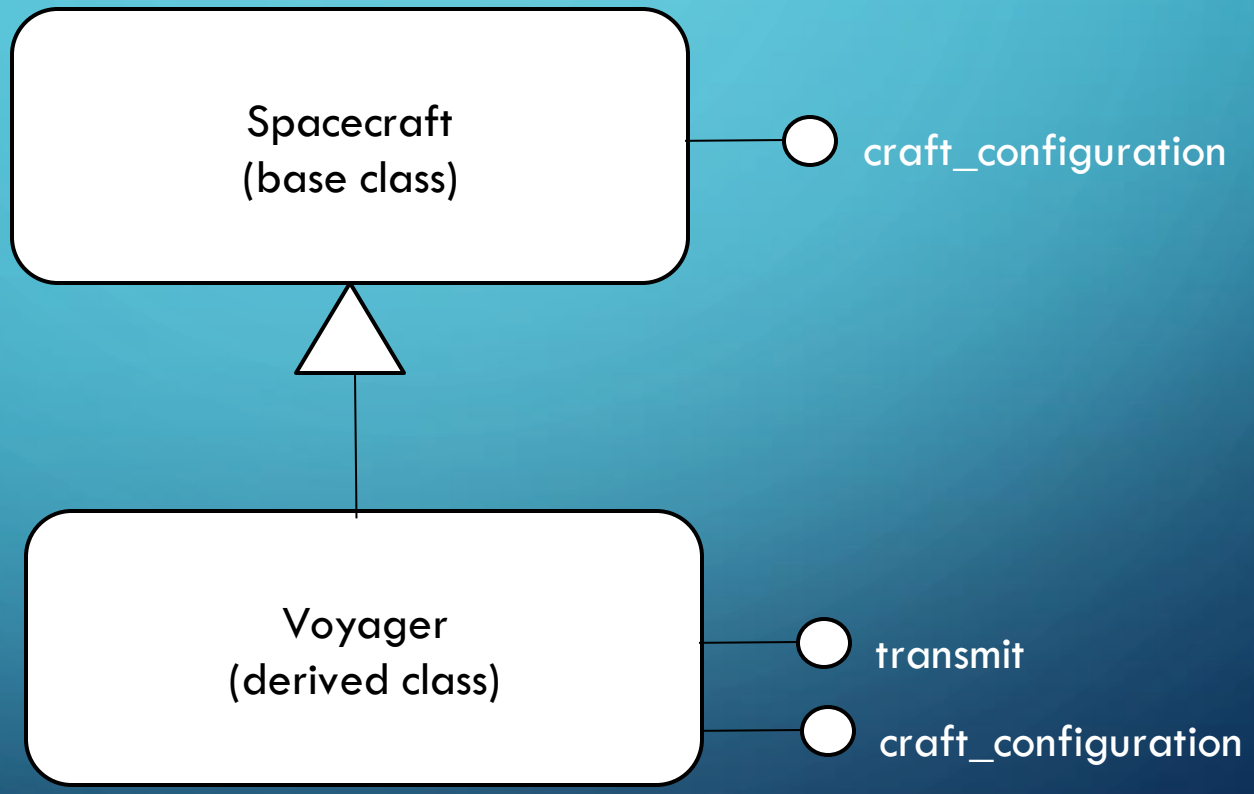
- Class inheritance



METHOD OVERRIDE

PLS1-CL4

- Class inheritance



HIDDEN CLASS ATTRIBUTE PLS1-CL5

- An attribute prefixed by double-underscores '__' can not be accessed outside the scope of the class.
  - It is not returned by __dict__

PARAMETERS

PLS1-PR1,2

- Any class method or standalone function supports the following:
  - Named parameters
  - Default parameters
  - Variable number of parameters

ASSERT

PLS1-AT

- Asserts are commonly placed at the beginning of functions to enforce the presence of parameters before getting into the function specific logic.
- Asserts are also common when writing unit tests.

EXCEPTIONS

PLS1-EXC1-3

- Python supports:
  - Exception handling with TRY-CATCH-FINALLY
  - Raising exceptions with RAISE

ANONYMOUS FUNCTIONS

PLS1-AF

- Inline functions

```
# PLS1-AF
# anonymous functions
sum = lambda arg1, arg2: arg1 + arg2          # a simple function defined inline
sum(1,2)
```

ITERATORS

PLS1-IT1,2

- An iterator is an object representing a stream of data.
- Iterators typically only provide the *next* item.
- Python allows for the creation of custom iterators.