CIS 2348 UNIVERSITY OF HOUSTON INFORMATION SYSTEM APPLICATION DEVELOPMENT

FALL 2021

CHAPTER 10

CLASSES

WHAT IS A CLASS?

- A CLASS IS AN OBJECT THAT CONTAINS RELATED VARIABLES AND FUNCTIONS
- IT'S A WAY TO NATURALLY GROUP ITEMS TOGETHER
- FOUNDATION OF MODERN OBJECT-ORIENTED PROGRAMMING
- PROVIDES FOR A STRAIGHTFORWARD WAY TO EXTEND FUNCTIONALITY EASIER

GENERAL CLASS SYNTAX

```
CLASS CLASSNAME:
        CLASS_VARIABLE_BLOCK
        DEF __INIT__(SELF,…):
                INIT_BLOCK
        DEF OTHER_CLASS_METHOD1(SELF,…):
                OTHER_CLASS_METHOD1_BLOCK
        DEF OTHER_CLASS_METHOD2(SELF,…):
                OTHER_CLASS_METHOD2_BLOCK
```

CLASS CONSTRUCTOR

```
DEF __INIT__(SELF,...):
        SELF.VARIABLE1=VALUE1
        SELF.VARIABLE2=VALUE2
        SELF.VARIABLE3=FUNCTION1(PARAMETERS)
        SELF.VARIABLE4=SELF.VARIABLE1+SELF.FUNCTION2(VALUE2)
```

CLASS CONSTRUCTOR EXAMPLE

```
CLASS MY_DICTIONARY:
        DEF __INIT__(SELF):
                SELF.NUM_READ=0
                SELF.DICT={}
MY_ENGLISH_SPANISH=MY_DICTIONARY()
```

ACCESSING MEMBER VARIABLES AND METHODS

- USE CLASSNAME.VARIABLENAME SYNTAX OR CLASSNAME.FUNCTIONNAME() SYNTAX
- I.E.

```
PRINT(MY_SPANISH_ENGLIST.NUM_READ)
```

OR

```
MY_SPANISH_ENGLISH.NUM_READ = X   #THOUGH TYPICALLY YOU WANT TO SET CLASS VARIABLES WITH CLASS METHODS
```

CLASS METHODS/MEMBER FUNCTIONS EXAMPLE

```
CLASS MY_DICTIONARY:
        DEF __INIT__(SELF):
                SELF.NUM_READ=0
                SELF.DICT={}
        DEF SET(SELF, WORD, TRANSLATION):
                SELF.DICT[WORD]=TRANSLATION
        DEF READ(SELF, WORD):
                SELF.NUM_READ+=1
                RETURN SELF.DICT[WORD]
```

CLASS METHOD EXAMPLE

```
CLASS MY_DICTIONARY:

……

MY_ENGLISH_SPANISH=MY_DICTIONARY()

MY_ENGLISH_SPANISH.SET('RED','ROJO')

PRINT(MY_ENGLISH_SPANISH.NUM_READ)

PRINT(MY_ENGLISH_SPANISH.READ('RED'), MY_ENGLISH_SPANISH.NUM_READ)
```

0

ROJO 1

CLASS INSTANCES

- CLASS DEFINES AN OBJECT “TYPE”
- AN INSTANCE IS A SPECIFIC CASE OF A CLASS TYPE
- ANALOGOUS TO LIST TYPE AND A GIVEN LIST

INSTANCE EXAMPLE

```
CLASS MY_DICTIONARY:

……

MY_ENGLISH_SPANISH=MY_DICTIONARY()

MY_ENGLISH_SPANISH.SET('RED','ROJO')

MY_ENGLISH_GERMAN=MY_DICTIONARY()

MY_ENGLISH_GERMAN.SET('RED','ROT')

PRINT(MY_ENGLISH_SPANISH.READ('RED'), MY_ENGLISH_SPANISH.NUM_READ,
MY_ENGLISH_GERMAN.NUM_READ)

ROJO 1 0
```

CLASS DESIGN

IN MY_DICTIONARY CLASS EXAMPLE SHOULD I ADD THE NUM_SET VARIABLE?

CLASS VS INSTANCE VARIABLE

- CLASS VARIABLE HAVE SAME VALUE FOR ALL INSTANCES
- INSTANCE VARIABLES HAVE INDEPENDENT VALUES FOR EACH INSTANCE

- SYNTAX:

```
DEF CLASSNAME():
        CLASSVARIABLE1=VALUE1
        DEF __INIT__(SELF):
                ....
```

CLASS VARIABLE EXAMPLE

```
CLASS MY_DICTIONARY:
        VERSION=3
        DEF __INIT__(SELF):
                SELF.NUM_READ=0
                SELF.DICT={}
        DEF SET(SELF, WORD, TRANSLATION):
                SELF.DICT[WORD]=TRANSLATION
        DEF READ(SELF, WORD):
                SELF.NUM_READ+=1
                RETURN SELF.DICT[WORD]
```

CLASS CONSTRUCTOR WITH PARAMETERS

- A CLASS CONSTRUCTOR CAN TAKE PARAMETERS TO INITIALIZE A CLASS INSTANCE

```
CLASS CLASS_NAME:
        DEF __INIT__(SELF,PARAMETER1,PARAMETER2,…)
            ….
MY_CLASS_INSTANCE=CLASS_NAME(ARGUMENT1,ARGUMENT2,…) # MUST MATCH
CONSTRUCTOR
```

CONSTRUCTOR WITH PARAMETER EXAMPLE

```
CLASS MY_DICTIONARY:
        VERSION=3
        DEF __INIT__(SELF, FIRST_WORD, FIRST_TRANSLATION):
                SELF.NUM_READ=0
                SELF.DICT={}
                SELF.DICT[FIRST_WORD]=FIRST_TRANSLATION
        DEF SET(SELF, WORD, TRANSLATION):
                SELF.DICT[WORD]=TRANSLATION
        DEF READ(SELF, WORD):
                SELF.NUM_READ+=1
                RETURN SELF.DICT[WORD]
```

USAGE

```
CLASS MY_DICTIONARY:

……

MY_ENGLISH_SPANISH=MY_DICTIONARY('RED','ROJO')

MY_ENGLISH_GERMAN=MY_DICTIONARY('RED','ROT')

PRINT(MY_ENGLISH_SPANISH.READ('RED'), MY_ENGLISH_SPANISH.NUM_READ,
MY_ENGLISH_GERMAN.NUM_READ)

ROJO 1 0
```

CONSTRUCTOR WITH DEFAULT PARAMETERS

```
CLASS CLASS_NAME:

        DEF __INIT__(SELF,PARAMETER1=DEFAULT1,PARAMETER2=DEFAULT2,...)


MY_CLASS_INSTANCE1=CLASS_NAME(ARGUMENT1,ARGUMET2)

MY_CLASS_INSTANCE2=CLASS_NAME()
```

CONSTRUCTOR WITH DEFAULT PARAMETERS EXAMPLE

```
CLASS MY_DICTIONARY:
        VERSION=3
        DEF __INIT__(SELF, FIRST_WORD='NONE', FIRST_TRANSLATION='NONE'):
                SELF.NUM_READ=0
                SELF.DICT={}
                IF (FIRST_WORD!='NONE') AND (FIRST_TRANSLATION!='NONE):
                        SELF.DICT[FIRST_WORD]=FIRST_TRANSLATION
        DEF SET(SELF, WORD, TRANSLATION):
                SELF.DICT[WORD]=TRANSLATION
        DEF READ(SELF, WORD):
                SELF.NUM_READ+=1
                RETURN SELF.DICT[WORD]
```

USAGE EXAMPLE

```
CLASS MY_DICTIONARY:

……

MY_ENGLISH_SPANISH=MY_DICTIONARY('RED','ROJO')

MY_ENGLISH_GERMAN=MY_DICTIONARY()

MY_ENGLISH_GERMAN.SET('RED','ROT')

PRINT(MY_ENGLISH_SPANISH.READ('RED'), MY_ENGLISH_SPANISH.NUM_READ,
MY_ENGLISH_GERMAN.NUM_READ)

ROJO 1 0
```

PRIVATE VS PUBLIC METHODS

- PYTHON MAKES NO INTERNAL DISTINCTION BETWEEN PUBLIC AND PRIVATE METHODS
- BY CONVENTION ANY FUNCTION THAT IS SUPPOSED BE ACCESSED ONLY BY OTHER CLASS METHODS SHOULD PREPEND AN UNDERSCORE TO THE NAME I.E.

```
CLASS CLASSNAME:
        CLASS_VARIABLE_BLOCK
        DEF __INIT__(SELF,…):
                INIT_BLOCK
        DEF OTHER_CLASS_METHOD1(SELF,…):
                OTHER_CLASS_METHOD1_BLOCK
        DEF _OTHER_PRIVATE_CLASS_METHOD2(SELF,…):
                OTHER_PRIVATE_CLASS_METHOD2_BLOCK
```

PRIVATE METHOD EXAMPLE

```
CLASS MY_DICTIONARY:
    CURSE_WORD_SET={'CURSEWORD1','CURSEWORD2'}
    DEF __INIT__(SELF):
        SELF.NUM_READ=0
        SELF.DICT={}
    DEF SET(SELF, WORD, TRANSLATION):
        IF SELF._CHECK_CURSEWORD(WORD)=='OK':
            SELF.DICT[WORD]=TRANSLATION
    DEF _CHECK_CURSEWORD(SELF,WORD):
        IF WORD NOT IN MY_DICTIONARY.CURSE_WORD_SET:
            RETURN('OK')
        ELSE:
            RETURN('NOT OK')
```

CLASSES CAN BE MEMBERS OF OTHER CLASSES

- CLASSES CAN HAVE MEMBERS THAT OTHER CLASSES
- USE AS ANY OTHER TYPE
- MEMBER CLASSES HAVE TO BE DEFINED FIRST

EXAMPLE

```
CLASS MY_DICTIONARY:

…

CLASS MY_TRIP:

        DEF __INIT__(SELF):

                SELF.DATE=0

                SELF.DICTIONARY=MY_DICTIONARY()

MY_SPAIN_TRIP=MY_TRIP()

MY_SPAIN_TRIP.DICTIONARY.SET('BLUE','AZUL')
```

IMPORTING CLASSES

- CLASSES CAN BE IMPORTED FROM OTHER MODULES

- SYNTAX :

FROM MODULE_NAME IMPORT CLASS_NAME

OR CAN BE ACCESSED USING MODULE SYNTAX

IMPORTED CLASS SYNTAX

```
IMPORT MY_TRIP_FILE

ENGLISH_SPANISH = MY_TRIP_FILE.MY_DICTIONARY()
```

OR

```
FROM MY_TRIP IMPORT MY_DICTIONARY

ENGLISH_SPANISH=MY_DICTIONARY()
```

CLASS CUSTOMIZATION

- __STR__ METHOD CAN BE PROVIDED TO HAVE AN OUTPUT THAT IS READABLE TO THE USER

- FOR THE DICTIONARY CASE WE COULD INCLUDE:

```
DEF __STR__(SELF):
        RETURN('THIS DICTIONARY HAS {} WORDS'.FORMAT(LEN(SELF.DICT)))
PRINT(ENGLISH_SPANISH)
THIS DICTIONARY HAS 100 WORDS
```

CLASS OPERATOR OVERLOADING

- ALLOWS US TO DEFINE THE BEHAVIOR OF FAMILIAR MATHEMATICAL OPERATORS PER CLASS
- SOME EXAMPLES OF THESE OPERATORS ARE: +.-,*,/,==,<,>
- ALLOWS TO EXTEND FAMILIAR NOTATION TO ABSTRACTED CLASSES
- EACH CLASS CAN HAVE ITS OWN DEFINITION OF THE OVERLOADED OPERATOR
- ACCOMPLISHED BY DEFINING SPECIAL METHOD NAMES I.E
  - + __ADD__
  - < __LT__

OPERATOR OVERLOADING EXAMPLE

```
CLASS MY_VECTOR:
        DIM=2
        DEF __INIT__(SELF,X,Y):
                SELF.X=X
                SELF.Y=Y


VECTOR1=MY_VECTOR(3,4)
```

```
DEF __LT__(SELF,OTHER):
        L1=MATH.SQRT(SELF.X**2+SELF.Y**2)
        L2=MATH.SQRT(OTHER.X**2 +OTHER.Y**2)
        RETURN( L1 < L2)
VECTOR1 = MY_VECTOR(3,4)
VECTOR2 = MY_VECTOR(0,5)
IF (VECTOR1 < VECTOR2):
        PRINT('VECTOR2 IS LONGER')
ELSE IF (VECTOR2 < VECTOR1):
        PRINT('VECTOR1 IS LONGER')
ELSE:
        PRINT('THEY ARE EQUAL IN LENGTH')
```

```
DEF __ADD__(SELF,OTHER):
        SUM_VECTOR=MY_VECTOR(0,0)
        SUM_VECTOR.X=SELF.X + OTHER.X
        SUM_VECTOR.Y=SELF.Y + OTHER.Y
        RETURN SUM_VECTOR
VECTOR1 = MY_VECTOR(3,4)
VECTOR2 = MY_VECTOR(0,5)
NEW_VECTOR = VECTOR1 + VECTOR2
```