#Class Exercises - Records

These tasks are designed to help you practice the knowledge and skills you have developed. There are exercises to help you revise and develop your understanding and also to challenge you further.

##Understanding records

A record is a data structure that groups together a number of variables which do not need to be of the same type and have no associated ordering. The variables of a record are called attributes or fields.

To use records in Python, we define the data structure that we want to create as a Class, and then declare the attributes of the class to use in the program.

To work with the individual attributes of a class we use dot notation:

`ClassVariableName.AttributeName`

Generally you will need to store more than a single record so it makes sense to use lists to keep them all together.

##Defining a record

Record data types can be created in Python using a class. Class definitions begin with the keyword class followed by the class name a colon.

`class className:`

A class needs to be initialised using the built-in method `\_\_init\_\_()`.

Any parameters that the function needs are named in the parentheses, along with the parameter self which it uses to refer to itself:

`def \_\_init\_\_(self, parameterName):`

The initial values for the record’s attributes should be set in the `init` function.

```python

class Contact:

def \_\_init\_\_(self,name,number):

self.name = name

self.number = number

```

###List of Records

To add a record to a list, use:

`list\_name.append(new\_record)`

You need to create an empty list and then append records to the list:

`list\_name = []`

##Task 1 - Dry Run Activity

The following program has been developed to create a record that will hold the name of a country, the name of its capital city and its currency.

```python

class Country:

def \_\_init\_\_(self):

self.name = '-'

self.capital = '-'

self.currency = '-'

# main program

country\_list = []

for count in range(2):

new\_country = Country()

new\_country.name = input("Name of country: ")

new\_country.capital = input("Capital city: ")

new\_country.currency = input("Currency used: ")

country\_list.append(new\_country)

```

Dry run this program when the user enters the following details at the prompts:

|Country|Capital|Currency|

|-------|-------|--------|

|U.K.|London|Pound|

|Japan|Tokyo|Yen|

|France|Paris|Euro|

Complete the following table:

|count|`new\_country.name`|`new\_country.capital`|`new\_country.currency`|`country\_list[0]`|`country\_list[1]`|`country\_list[2]`|

|-----|--------------|-----------------|------------------|-----------------|------------------|-----------------|

| | | | |`.name = `, `.capital = `, `.currency = `| | |

|1 |U.K. |London |Pound |`.name = U.K.`, `.capital = London`, `.currency = Pound` | | |

|2 |Japan |Tokyo |Yen |`.name = Japan`, `.capital = Tokyo`, `.currency = Yen` | | |

| | | | | | | |

##Task 2 - Code correction

The code below contains some errors in the function `get\_meaning`. Annotate the code \*with comments\* to show where you think the errors may be.

```python

def get\_meaning(term):

index = 1 #The index should start at 0

wanted\_index = -1

while index < len(tech\_list)-1: #shouldn't have the "-1"

if tech\_list[index].term == term:

wanted\_index = index

index = index + 1

if wanted\_index != -1:

return tech\_list[index].meaning #should be returning the wanted\_index

else:

return 'not listed'

```

Discuss your ideas with another student and list the errors that you have found together.

Now load the program `technicalDictionary.py` and run it to check where the errors occur. You may need to correct the errors one at a time and test the program in between to find all of the errors.

##Class Exercises

In this section you may choose the exercises you attempt. There are three types of exercises to consider:

1. \*\*Revision Exercises\*\* - choose these exercises if you are not confident in your understanding of selection statements

2. \*\*Development Exercises\*\* - choose these exercises if you are confident in your understanding but want some more practice

3. \*\*Stretch and Challenge Exercises\*\* - choose these exercises if you feel you have mastered selection and want to tackle some tougher problems

Once you feel more confident attempt some of the more difficult exercises. The more practice you get now the more comfortable you will be using selection in more complex programs later in the course.

Remember, practice makes perfect!

###Attempting These Exercises

You should do the following for each exercise you attempt:

1. Create \*\*flowcharts\*\* first - plan your solution on paper before attempting it

2. Create a set of \*\*test data\*\* - select values you will enter to test the program and know beforehand what you expect the answers to be

3. Write the \*\*program\*\* - write the program in Python using the pseudocode to assist you and the test data to ensure the program functions correctly

###Revision Exercises

Attempt these tasks if you need to build your confidence and understanding of using records.

1. Declare a record type to store the name of a student and the marks that they got in most recent test.

Write and test a program StudentMarks, which takes two parameters: a string studentName and an integer testMark.

The program should use a record type to display the student’s name and test mark, formatted in a user-friendly way.

2. Extend your program from exercise 1 to read in 3 students’ details and present them in a tabulated format.

You may find it useful to use the built-in string function .format().

3. Declare a record type to store the following details of employees:

- Employee name

- Employee number

- Number of hours worked this week

- Hourly rate of pay

Write a program that allows a user to enter the data for an employee and it displays a simple pay slip with their details and their total pay for that week.

The interface for this program should look similar to the one below:

![](https://www.dropbox.com/s/de9xtfbtf7ksskg/pay\_slip.jpg?dl=1)

###Development Exercises

Attempt these tasks if you are confident in your understanding of records but feel you need more practice.

1. Declare a record data type to store the names of 5 students and the marks that they each gained for their 4 A level modules.

It should display the students’ names, modules marks and total mark in a tabulated format.

2. Extend your program for exercise 4 so that it calculates the mean total mark of the students and calculates and displays how many marks each student’s total was above or below the mean mark (to one decimal place).

3. Declare a record type to store 6 numbers for the National Lottery draw and the date of the draw.

Write a program that asks the user which date they want to select lottery numbers for, and how many sets of numbers they want the program to select.

The program selects the required number of sets of 6 different random numbers between 1 and 49 inclusive, and then displays the date of the draw and each set of numbers in ascending order.

###Stretch and challenge exercises

Attempt these tasks if you feel you have mastered records and want to tackle tougher problems.

1. As in exercise 3, declare a record type to store the following details of employees:

- Employee name

- Employee number

- Number of hours worked this week

- Hourly rate of pay

Write a program that stores the data for 15 employees (for this exercise, this data may be hard-coded initially).

The program should allow the user to search for an employee by name and display a simple pay slip for that employee with their details and their total pay for that week. If the employee’s name cannot be found it should display an appropriate message.

2. Extend your program from exercise 7 so that it gives the user the choice of whether to edit one employee’s details, or to display their pay slip. If the user selects to edit the employee’s details, the program should search for the employee by name, display the stored details, and ask the user for the new data and store the updated record.

9. Write a program that reads in an unspecified number of friends’ first names and dates of birth into records. It should then display a birthday planner with the names and dates of birth (in date order) for all those friends with birthdays in the next 30 days. (Hint: the `datetime` module includes `timedelta` objects for time differences.)