Assignment 2A functions, arguments and parameters

Lillebaelt Academy of
Professional Higher Education

Author
Martin Grønholdt
mart80c7@edu.eal.dk

Friday 18 November 2016

Table of Contents

Introduction

The programs in this hand-in are structured using functions. Every program has at least one function, besides the main function, to perform the processing. One program uses classes, and recursiveness, it may not be necessary but the task fit the bill.

Error handling

All programs handle bad input by asking the user, to use only numbers, where after it exits.

```
Enter the amount of a purchase: 2hjjhg

Please use only numbers.
```

Example output of a program when the user enters an incorrect value.

1 Kilometre Converter

This program converts distance in kilometres to distance in miles.

prog1.py

```python
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# The above lines tell the shell to use python as interpreter when the
# script is called directly, and that this file uses utf-8 encoding,
# because of the country specific letter in my surname.
'''
Name: Program 1
Author: Martin Bo Kristensen Grønholdt.
Version: 1.0 (13/11-2016)

Convert kilometres to miles.
'''
def km2miles(kilometres = 0):
    '''
    Convert kilometres to miles.

    :param kilometres: Distance in kilometres.
    :return: Distance in miles.
    '''
    #Code that converts km to miles
    return(kilometres * 0.6214)


def main():
    '''
    Main entry point.
    '''
    # Get the amount of kilometres from the user.
    try:
        kilometres = float(input('Input kilometres: '))
    except ValueError:
        # Complain when something unexpected was entered.
        print('\nPlease use only numbers.')
        exit(1)

    print('{:0.2f} kilometres is {:0.2f} miles'.format(kilometres, km2miles(kilometres)))


# Run this when invoked directly
if __name__ == '__main__':
    main()
```

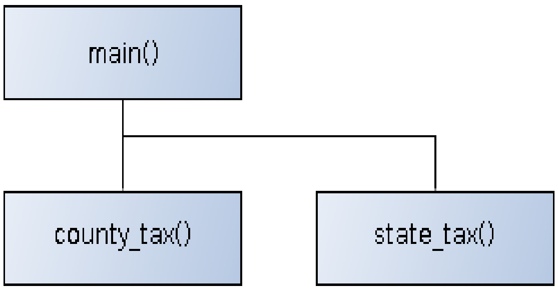
Result

```
/usr/bin/python3.5 "/home/oblivion/Dokumenter/Skole/It-
et/2016/programming/Assignment A2/prog1.py"
Input kilometres: 10
10.00 kilometres is 6.21 miles
```

Output of the program when run from the command line.

2 Sale Tax Program Refactoring

This program uses a function for each of the calculations it performs on the input data.



Hierarchy diagram of the program.

The program starts out in the main() function calling both county_tax() and state_tax() several times with the value entered by the user, the return values are printed as part of the result.

prog2.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# The above lines tell the shell to use python as interpreter when the
# script is called directly, and that this file uses utf-8 encoding,
# because of the country specific letter in my surname.
'''
Name: Program 2
Author: Martin Bo Kristensen Grønholdt.
Version: 1.0 (13/11-2016)

Calculate the county and state sales tax, and the final total of a purchase.
'''
def county_tax(total_purchase = 0):
    '''
    Calculate the county tax of 2% of the total purchase.

    :param total_purchase: Amount spend on the purchase in total.
    :return: Amount of county tax.
    '''
    # Calculate the county tax.
    return(total_purchase * 0.02)


def state_tax(total_purchase=0):
    '''
    Calculate the state tax of 4% of the total purchase.

    :param total_purchase: Amount spend on the purchase in total.
    :return: Amount of state tax.
    '''
    # Calculate the state tax
```

```python
    return(total_purchase * 0.04)


def main():
    '''
    Program main entry point.
    '''
    # Get the amount of purchase from the user.
    try:
        total_purchase = float(input('Enter the amount of a purchase: '))
    except ValueError:
        # Complain when something unexpected was entered.
        print('\nPlease use only numbers.')
        exit(1)

    # Print the totals
    # Use new style Python 3 format strings.
    # {:12.2f} means align for a total of 12 digits with 2 digits
    # after the decimal point.
    print('\nTotal purchase:\t\t\t{:12.2f}'.format(total_purchase))
    print('State sales tax (4%):\t{:12.2f}'.format(state_tax(total_purchase)))
    print('County sales tax (2%):\t{:12.2f}'.format(county_tax(total_purchase)))
    print('Final total:\t\t\t{:12.2f}'.format(total_purchase +
                                          state_tax(total_purchase) +
                                          county_tax(total_purchase)))


# Run this when invoked directly
if __name__ == '__main__':
    main()
```

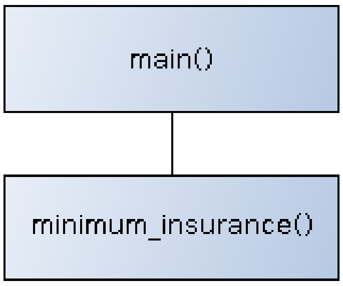
Result

```
/usr/bin/python3.5 "/home/oblivion/Dokumenter/Skole/It-
et/2016/programming/Assignment A2/prog2.py"
Enter the amount of a purchase: 123

Total purchase:                 123.00
State sales tax (4%):             4.92
County sales tax (2%):            2.46
Final total:                    130.38
```

Output of the program when run from the command line.

3 How Much Insurance?

This program calculates the minimum amount of insurance to cover the replacement cost of a building.



Hierarchy diagram of the program.

The program starts out in the main() function calling minimum_insurance() with the value entered by the user, the return value is printed as part of the result.

prog3.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# The above lines tell the shell to use python as interpreter when the
# script is called directly, and that this file uses utf-8 encoding,
# because of the country specific letter in my surname.
'''
Name: Program 3
Author: Martin Bo Kristensen Grønholdt.
Version: 1.0 (13/11-2016)

Calculate the minimum amount of insurance, given the replacement cost of a
building.
'''
def minimum_insurance(replacement_cost = 0):
    '''
    Calculate the county tax of 2% of the total purchase.

    :param replacement_cost: The replacement cost of the building.
    :return: Minimum insurance sum.
    '''
    # Calculate minimum insurance sum.
    return(replacement_cost * 0.80)


def main():
    '''
    Program main entry point.
    '''
    # Get the amount of purchase from the user.
    try:
        replacement_cost = float(input('Enter the replacement cost of the ' +
                                       'building: '))
```

```python
    except ValueError:
        # Complain when something unexpected was entered.
        print('\nPlease use only numbers.')
        exit(1)

    # Print the totals
    # Use new style Python 3 format strings.
    # {:12.2f} means align for a total of 12 digits with 2 digits
    # after the decimal point.
    print('\nBuilding replacement cost:\t\t' +
          '{:12.2f}'.format(replacement_cost))
    print('Minimum insurance sum (80%):\t' +
          '{:12.2f}'.format(minimum_insurance(replacement_cost)))


# Run this when invoked directly
if __name__ == '__main__':
    main()
```

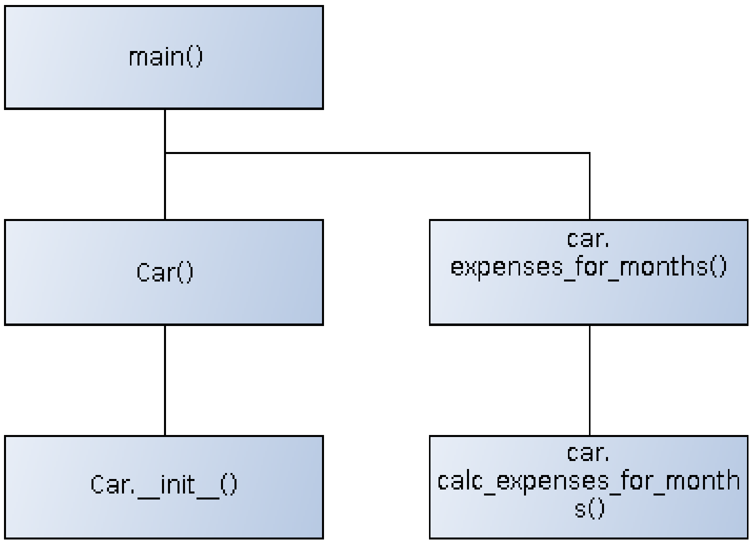
Result

```
/usr/bin/python3.5 "/home/oblivion/Dokumenter/Skole/It-
et/2016/programming/Assignment A2/prog3.py"
Enter the replacement cost of the building: 1230000

Building replacement cost:       1230000.00
Minimum insurance sum (80%):      984000.00
```

Output of the program when run from the command line.

4 Auto-mobile Costs

This program calculates the cost of expenses of owning a car on a monthly basis, printing the cost after a month and after a year.



Hierarchy diagram of the program.

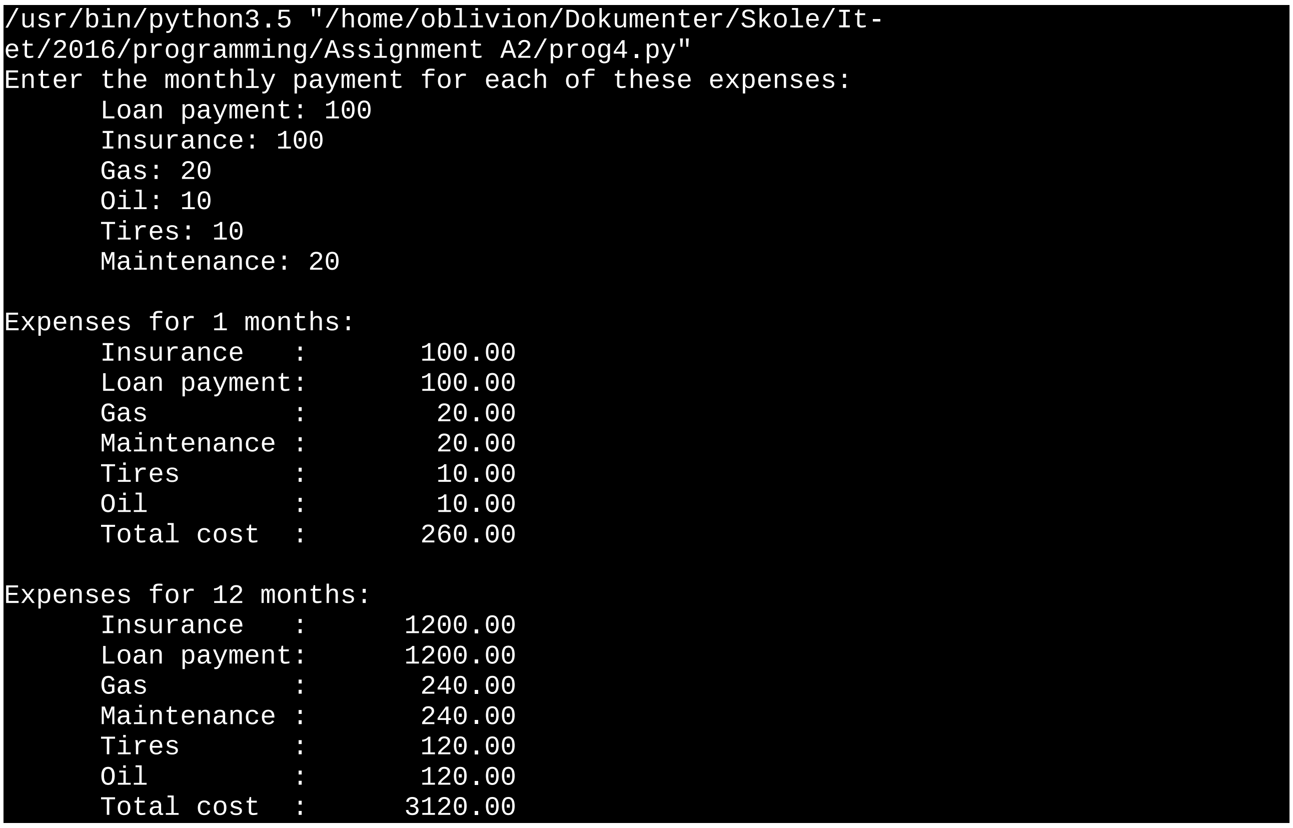
- This program uses a class, Car, to create a model of a car with the properties and methods that are needed for the current assignment. When an instance of the Car class is created, in main(), it will ask for the values of these properties by invoking the constructor Car.__init__(self).
- After creating the car instance the main() function calls the expenses_for_months() method twice to print the cost of all expenses monthly and annually.
- The Car.expenses_for_months() is a method that handles the calling and printing of the result from the method Car.calc_expenses_for_months(), which is called to handle the actual calculations.
- The Car.calc_expenses_for_months() is a recursive method, meaning it calls itself until reaching the final result, which it passes back along the calls to the original caller Car.expenses_for_months(). This is possible because the next step in the calculation is based upon the current. Car.calc_expenses_for_months() therefore uses a dictionary called current_expenses to pass along the current result, to the next call.

prog4.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# The above lines tell the shell to use python as interpreter when the
# script is called directly, and that this file uses utf-8 encoding,
# because of the country specific letter in my surname.
"""
Name: Program 4
Author: Martin Bo Kristensen Grønholdt.
Version: 1.0 (13/11-2016)

Asks the user to enter the monthly costs for expenses incurred from operating
and owning an automobile. Print out the total cost of the expenses for a month
and a year.
"""
class Car(object):
    """
    Class to calculate the expenses incurred from operating and owning an
    auto-mobile.
    """
    # Loan payment per month.
    loan_payment = 0
    # Insurance payment per month.
    insurance = 0
    # Expenses on gas per month.
    gas = 0
    # Expenses on oil per month.
    oil = 0
    # Expenses on tires per month.
    tires = 0
    # Expenses on maintenance per moth.
    maintenance = 0

    def __init__(self):
        """
        Ask for values from the user, for expenses, when instantiating this
        object.
        """
        print('Enter the monthly payment for each of these expenses:')
        try:
            self.loan_payment = float(input('\tLoan payment: '))
            self.insurance = float(input('\tInsurance: '))
            self.gas = float(input('\tGas: '))
            self.oil = float(input('\tOil: '))
            self.tires = float(input('\tTires: '))
            self.maintenance = float(input('\tMaintenance: '))
        except ValueError:
            # Complain when something unexpected was entered.
            print('\nPlease use only numbers.')
            exit(1)

    def calc_expenses_for_months(self, current_expenses=None, months=1):
        """
        Calculate the expenses for an span of months.

        :param current_expenses: Tuple of the current expenses incurred up until
                                 the current call of the function.
        :param months: Span of months to calculate expenses for.
        :return: A dictionary with the total cost for each expense.
        """
        # If there are still more month to go.
        if months != 0:
            # If this is not the first call to this function.
            if current_expenses is not None:
                # Add monthly expenses to the current expenses.
                current_expenses['Loan payment'] += self.loan_payment
                current_expenses['Insurance'] += self.insurance
                current_expenses['Gas'] += self.gas
                current_expenses['Oil'] += self.oil
                current_expenses['Tires'] += self.tires
                current_expenses['Maintenance'] += self.maintenance
            else:
                # If this is the first call create a dictionary for the
                # expenses and add them for the current month.
                current_expenses = dict()
                current_expenses['Loan payment'] = self.loan_payment
                current_expenses['Insurance'] = self.insurance
                current_expenses['Gas'] = self.gas
                current_expenses['Oil'] = self.oil
                current_expenses['Tires'] = self.tires
                current_expenses['Maintenance'] = self.maintenance
            # Make the function recursive, to shoot myself in the foot when
            # I have to create the hierarchy diagram.
            # The function calls itself until month is 0.
            return(self.calc_expenses_for_months(current_expenses, months - 1))
        else:
            # If this is the final call, return a dictionary of the accumulated
            # expenses.
            return(current_expenses)

    def expenses_for_months(self, months=1):
        """
        Calculate and output the expenses for a span of months.

        :param months: Span of months to calculate expenses for.
        """
        # Print a header for the expenses output.
        print('\nExpenses for {} months: '.format(months))
        # Get a dictionary with the total expenses as values and expense name
        # as key.
        expenses = self.calc_expenses_for_months(None, months)
        # Variable to keep the total cost of all expenses.
        total_cost = 0
        # Run through all expenses in the dictionary printing their key
        # as description and value as result.
        for key, value in expenses.items():
            # Align both the description and the result using new style string
            # formatting.
            print('\t{:<12}: {:12.2f}'.format(key, value))
            total_cost += value
        #Print the total cost
        print('\t{:<12}: {:12.2f}'.format('Total cost', total_cost))


def main():
    """
    Program main entry point.
```

```python
    """
    #Create the Car instance
    car = Car()
    # Calculate and print the expenses for one month.
    car.expenses_for_months(1)
    # Calculate and print the expenses for a year.
    car.expenses_for_months(12)


# Run this when invoked directly
if __name__ == '__main__':
    main()
```

Result

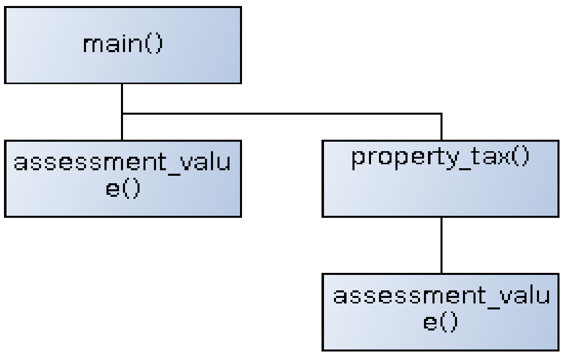


```
/usr/bin/python3.5 "/home/oblivion/Dokumenter/Skole/It-
et/2016/programming/Assignment A2/prog4.py"
Enter the monthly payment for each of these expenses:
      Loan payment: 100
      Insurance: 100
      Gas: 20
      Oil: 10
      Tires: 10
      Maintenance: 20

Expenses for 1 months:
      Insurance   :       100.00
      Loan payment:       100.00
      Gas         :        20.00
      Maintenance :        20.00
      Tires       :        10.00
      Oil         :        10.00
      Total cost  :       260.00

Expenses for 12 months:
      Insurance   :      1200.00
      Loan payment:      1200.00
      Gas         :       240.00
      Maintenance :       240.00
      Tires       :       120.00
      Oil         :       120.00
      Total cost  :      3120.00
```

Output of the program when run from the command line.

5 Property Tax

This program calculates the assessment value and property tax from the actual value of a piece of property.



Hierarchy diagram of the program.

The program starts out in the main() function calling both assessment_value(value = 0) and property_tax(value = 0) with the value entered by the user, the return values are printed as part of the result. property_tax(value = 0) in turn calls assessment_value(value = 0) since its calculations are based on the result of this function.

prog5.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# The above lines tell the shell to use python as interpreter when the
# script is called directly, and that this file uses utf-8 encoding,
# because of the country specific letter in my surname.
"""
Name: Program 5
Author: Martin Bo Kristensen Grønholdt.
Version: 1.0 (13/11-2016)

Calculate the assessment value and property tax from the actual value of a piece
of property
"""
def assessment_value(value = 0):
    """
    Calculate the assessment value (60%) from the actual value.

    :param value: The actual value of the property.
    :return: The assessment value.
    """
    # Return the assessment value.
    return(value * 0.6)
```

```python
def property_tax(value=0):
    """
    Calculate the property tax (0.64%) from the actual value.

    :param value: The actual value of the property.
    :return: The assessment value.
    """
    # Return the property tax value.
    return(assessment_value(value) * 0.0064)


def main():
    """
    Program main entry point.
    """
    # Get the value of the property from the user.
    try:
        value = float(input('Enter the value of the property: '))
    except ValueError:
        # Complain when something unexpected was entered.
        print('\nPlease use only numbers.')
        exit(1)

    print('\nProperty value:\t\t{:12.2f}'.format(value))
    print('Assessment value:\t{:12.2f}'.format(assessment_value(value)))
    print('Property tax:\t\t{:12.2f}'.format(property_tax(value)))


# Run this when invoked directly
if __name__ == '__main__':
    main()
```

Result

```
/usr/bin/python3.5 "/home/oblivion/Dokumenter/Skole/It-
et/2016/programming/Assignment A2/prog5.py"
Enter the value of the property: 10000

Property value:       10000.00
Assessment value:      6000.00
Property tax:            38.40
```

Output of the program when run from the command line.

Conclusion

Functions are the basic building blocks of larger programs. An algorithm that is divided in to smaller parts becomes it both simpler to understand, easier to maintain, and might even avoid repeated code, by being able to isolate the repeated parts of the algorithm into separate functions.