Software Development 1
Conditional Statements(ifs)

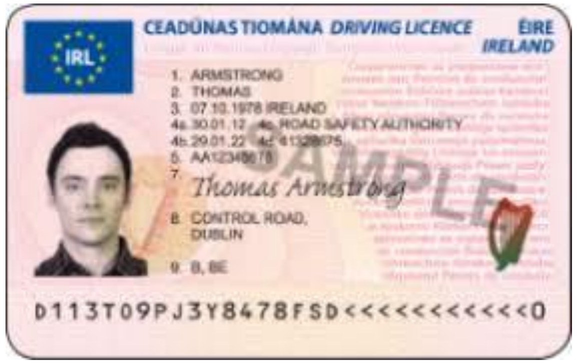
TUDublin –Tallaght Campus

E.Costelloe

Conditional Statements

- We can now take in data and perform mathematical operations, or perform operations on strings.
- This is very important in programming, but there is the next step.
- How to ask a question, and make a decision, and depending on the answer perform specific tasks.
- Lets look at a basic example.

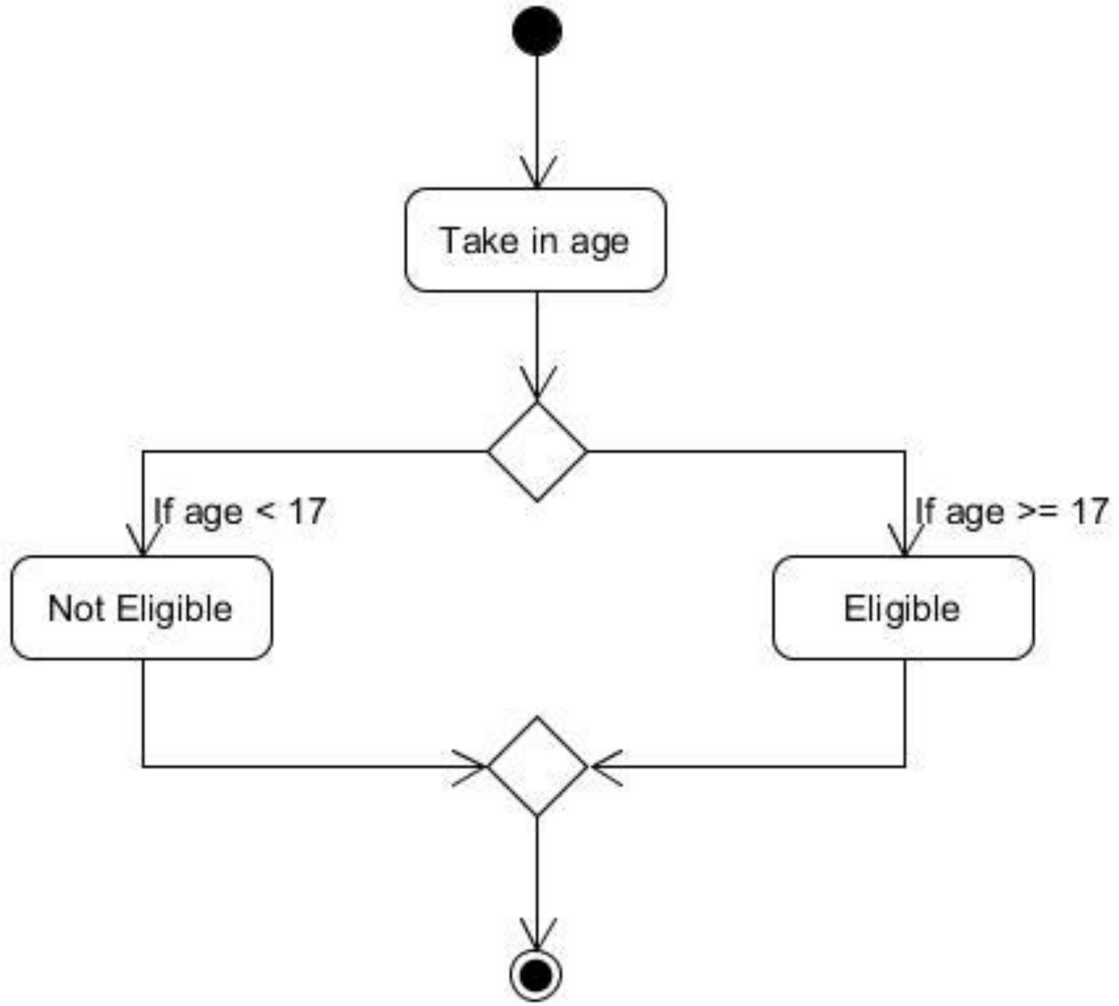
Conditional Statements

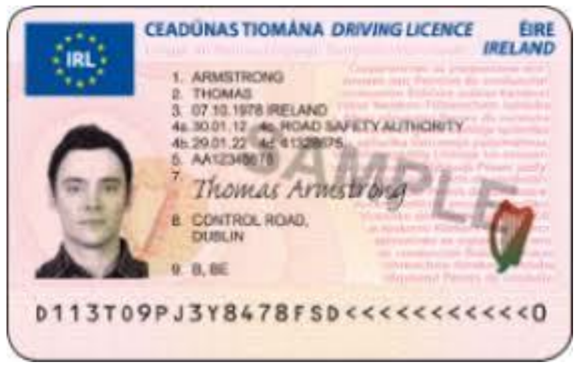


- Driving licence (car)
- Lets build a program that asks for a persons age
- Base on their age and Irish law, the computer will tell them if they are eligible to apply for a licence
- If they are 17 or over they are eligible, if they are under 17, they are not eligible.

Conditional Statements

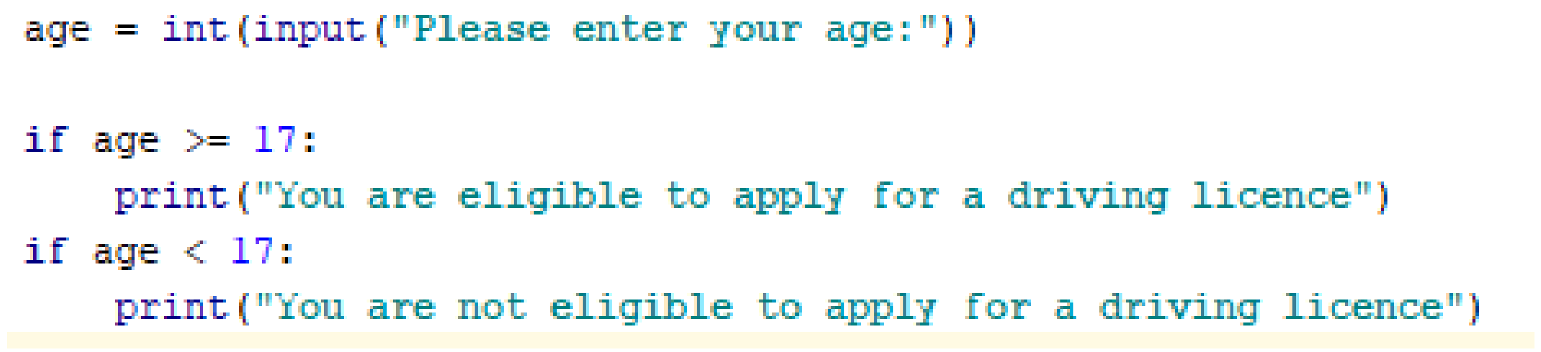
- Algorithm





Conditional Statements

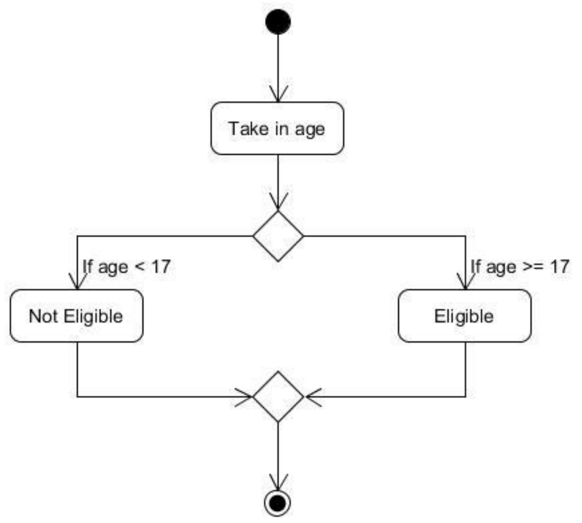
- The result would look like the following:
- Notice the indentation, this is extremely important in Python.



```
age = int(input("Please enter your age:"))

if age >= 17:
    print("You are eligible to apply for a driving licence")
if age < 17:
    print("You are not eligible to apply for a driving licence")
```

```
Input: 19

Output:
    You are eligible to apply for a driving licence
```

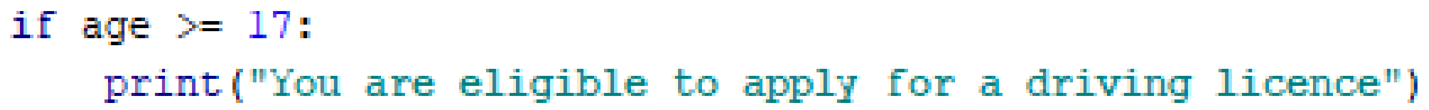
Conditional Statements



- Its good to visualize a conditional statement as a fork.
- But what really happens is the indentation is executed if and only if the conditional evaluates to true (Based on Boolean values as mentioned in the variable section).
- Conditional statements rely on Boolean values

Conditional Statements

- Lets look in detail at the previous code:



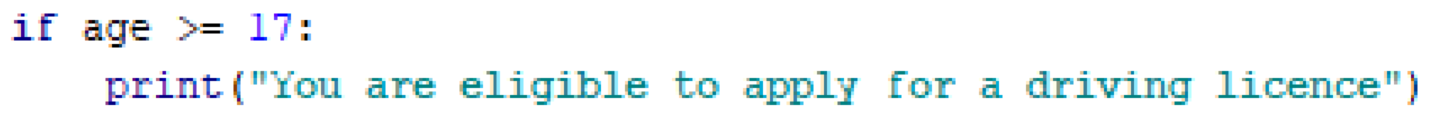
```
if age >= 17:
    print("You are eligible to apply for a driving licence")
```

Line 1, => **if** age >= 17:

- The if statement

Conditional Statements

- Lets look in detail at the previous code:



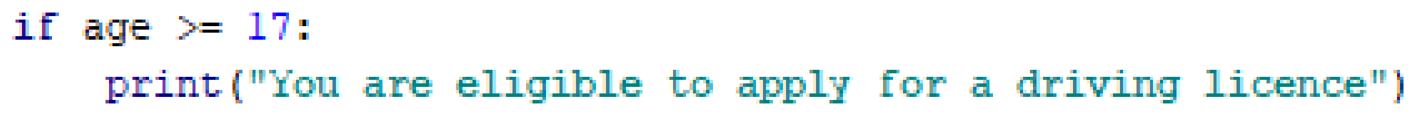
```
if age >= 17:
    print("You are eligible to apply for a driving licence")
```

Line 1, => if **age >= 17**:

- The question to evaluate – i.e. The condition

Conditional Statements

- Lets look in detail at the previous code:



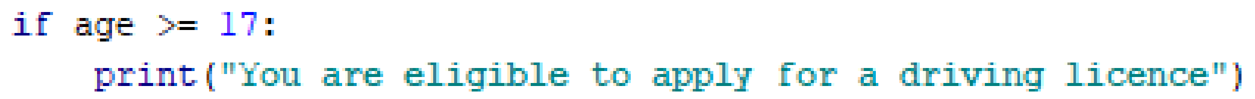
```
if age >= 17:
    print("You are eligible to apply for a driving licence")
```

Line 1, =>   if age >= 17:

- The end of the if statement

Conditional Statements

- Lets look in detail at the previous code:



```python
if age >= 17:
    print("You are eligible to apply for a driving licence")
```

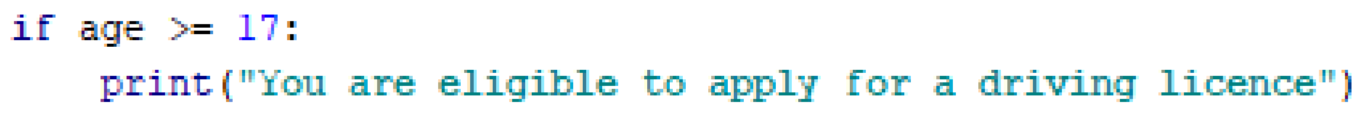
Line 2, => The code that gets executed, if the question(condition) evaluates to true. If the condition evaluates to false the indented code is skipped over.

In this case a simple print statement.

Note: only the code indented gets executed if the condition evaluates to true.

Indentation

In Python, we consistently indent all the statements in a given single **nested** block the same distance to the right, and Python uses the statements’ physical indentation to determine where the block starts and stops:



```
if age >= 17:
    print("You are eligible to apply for a driving licence")
```

By *indentation*, that is the blank whitespace all the way to the left of the nested statement here, there may be more than 1 nested statement.

Python doesn’t care *how* you indent (you may use either spaces or tabs), or *how much* you indent (you may use any number of spaces or tabs). The syntax rule is only that for a given single nested block, all of its statements must be indented the same distance to the right. If this is not the case, you will get a syntax error,

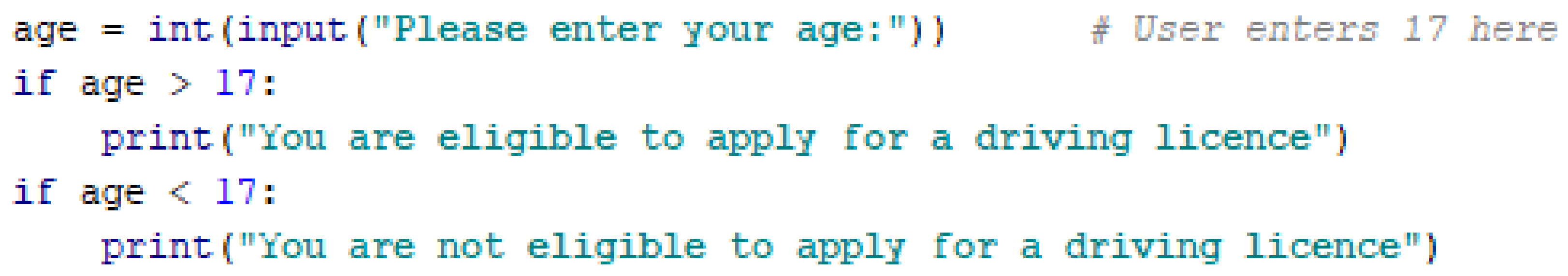
As per PEP8,Style Guide for Python Code, https://www.python.org/dev/peps/pep-0008/, Use 4 spaces per indentation level and spaces are the preferred indentation method.

Don’t mix spaces and tabs!

Indentation is a deliberate feature of Python, and it’s one of the main ways that Python almost forces programmers to produce uniform, regular, and readable code.

Conditional Statements

- There are other options, for conditional statements
- What happen here, with the small change in conditions, and the user enters 17?



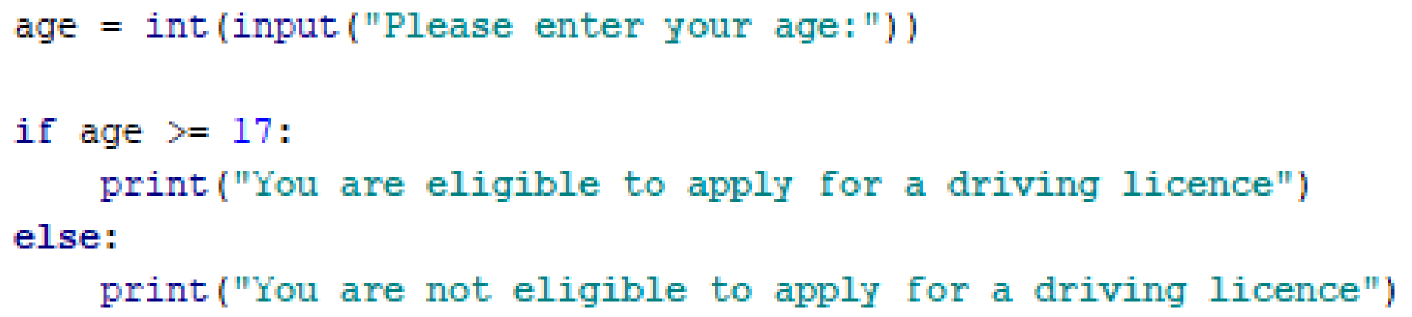
```
age = int(input("Please enter your age:"))       # User enters 17 here
if age > 17:
    print("You are eligible to apply for a driving licence")
if age < 17:
    print("You are not eligible to apply for a driving licence")
```

- Both if statements would be skipped, thus nothing would be printed to the screen!
- Not good for the user, also to note at this stage, we really want the program to execute the code in one of the two if statements.

Conditional Statements

- We can use an if -> else statement.

- This guarantees that at least one option will be executed (use with caution)

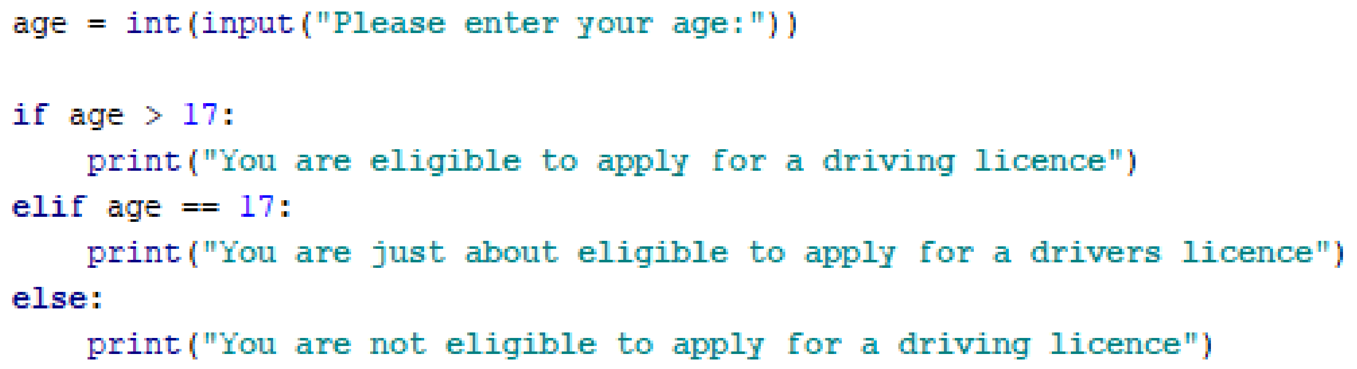


```
age = int(input("Please enter your age:"))

if age >= 17:
    print("You are eligible to apply for a driving licence")
else:
    print("You are not eligible to apply for a driving licence")
```

- If the first condition evaluated to false, the second is executed.

Conditional Statements

- This can be furthered to several statements, resulting in at least one code execution



```python
age = int(input("Please enter your age:"))

if age > 17:
    print("You are eligible to apply for a driving licence")
elif age == 17:
    print("You are just about eligible to apply for a drivers licence")
else:
    print("You are not eligible to apply for a driving licence")
```

- If the first conditional evaluated to false, the second is evaluated, if it evaluates to false, the third is executed.
- What would happen if you do not add an else?
- You can add as many elif's as you wish. Once a condition evaluates to True, the computer executes its corresponding nested block and exits the if statement. So at most only **one** block is executed.

Conditional Statements

- Lets for a minute focus on the questions:

$$\text{if} \quad \underbrace{\qquad\qquad\qquad\qquad}_{\text{The condition is located here}} \quad :$$

- We have already seen in the example, >, <, <=, >= and ==

- Why == and not =?

Conditional Statements

- Lets for a minute focus on the questions:

Greater Than	>
Less Than	<
Equal To	==
Less Than or Equal To	<=
Greater Than or Equal To	>=
Not Equal to	!=

Conditional Statements

- Lets for a minute focus on the questions:

```
x = int(input(“Enter your age”))
if x == 19:
```

- On either side of the operator, must be a variable or a hard coded value

Conditional Statements

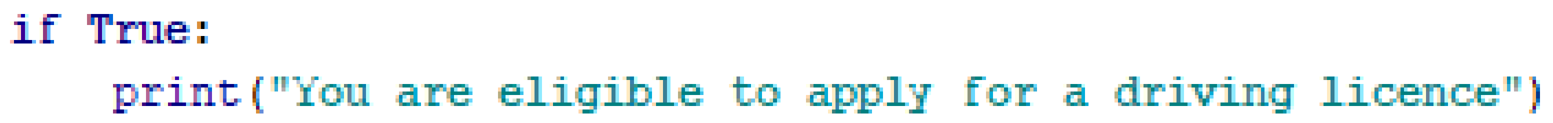
- Lets for a minute focus on the questions:

```
x = int(input(“Enter your age”))
if x == 19:
```

- Variable x is compared to 19, if they are equal, the code indented under the if statement is execute, else it is ignored.

Conditional Statements

- What would the output be here?



```
if True:
    print("You are eligible to apply for a driving licence")
```

- Remember the Boolean values: these are the overall evaluations of conditional statements.
- If True, execute code, if False, do not.

Using indentation to create Blocks

- By indenting a line it becomes a block. A block is one is one or more consecutive lines indented by the same amount. The lines form a logical unit.
- Blocks can be used, as here, as part of an if statement. They're the statement or group of statements that gets executed if the condition is True.
- Blocks can be as many statements as you require.

PEP 8:

Indentation

Use 4 spaces per indentation level

Python 3 disallows mixing the use of tabs and spaces for indentation.

Class example:

You have been asked to write a program for a simple payroll application:

Employees are paid on the basis of an hourly rate.

However, if they work hours in excess of 40 hours per week, they are paid an extra bonus of half the hourly rate for the extra hours worked.

The program must be able to check the hours worked and execute the bonus instructions only if the hours are greater than 40.

Class example:

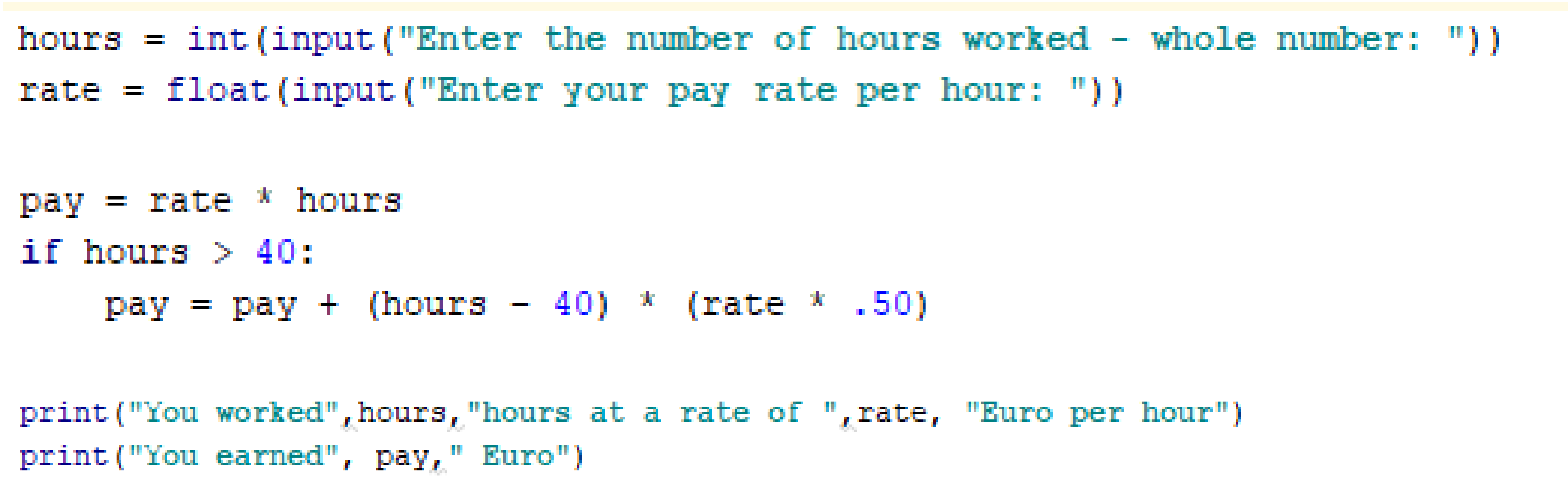
Test Data		
Hours	**Rate**	**Results Expected (Pay)**
30	5.00	150.00
40	10.00	400.00
45	5.00	237.50

Class example:

```
Begin
    1. Input hours
    2. Input rate
    3. Calculate pay at basic rate
    4. If hours > 40
            add extra at half the basic rate for overtime hours
    5. Output pay
End
```

Class example:

```
Begin
    1.  Input hours
    2.  Input rate
    3.  calculate pay at basic rate
          3.1  pay =  rate * hours
    4.  If  hours  > 40
          add extra at half the basic rate for overtime hours
          4.1 pay = pay + ((hours - 40) * (rate * 0.50))
    5.  Output pay
End
```



```python
hours = int(input("Enter the number of hours worked - whole number: "))
rate = float(input("Enter your pay rate per hour: "))

pay = rate * hours
if hours > 40:
    pay = pay + (hours - 40) * (rate * .50)

print("You worked",hours,"hours at a rate of ",rate, "Euro per hour")
print("You earned", pay," Euro")
```

Class example 2:

A program is required to compute **gross** and **net** pay

Hourly rate and hours worked are entered by the user

Gross pay is hourly rate * hours worked

A standard tax amount of €25 is deducted if the employee earns more than €100. We then have the net pay.

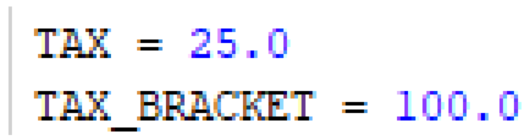
Class example 2:

Analysis:

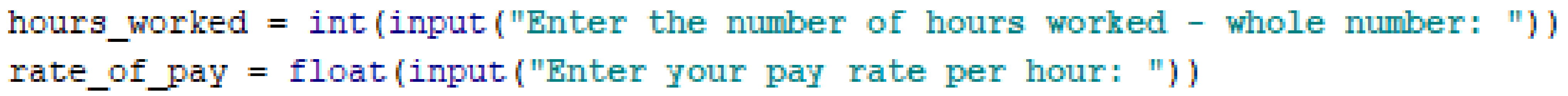
- Input:
  - hours _worked
  - rate _of_ pay
- Output:
  - gross_pay
  - net_pay
- Constants:
  - TAX = 25.0
  - TAX_BRACKET = 100.0

Class example 2:

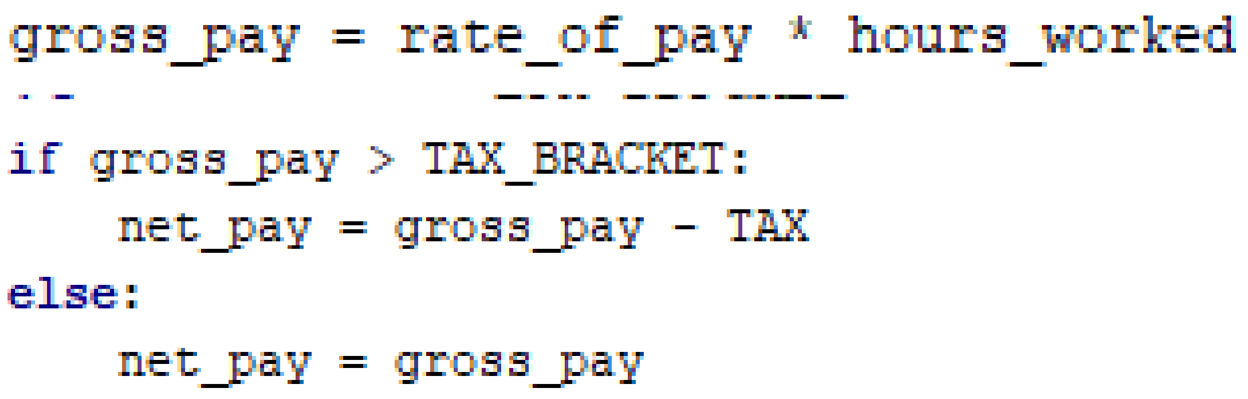
```
1.  Input hours _worked
2.  Input rate _of_ pay

3. Compute gross_pay
 3.1 gross_pay  = rate _of_ pay  *  hours _worked

4. Compute net_pay
 4.1 If gross_pay > TAX_BRACKET
           net_pay  =  gross_pay - TAX
     else
           net_pay = gross_pay

5. Output gross_pay and netPay
```



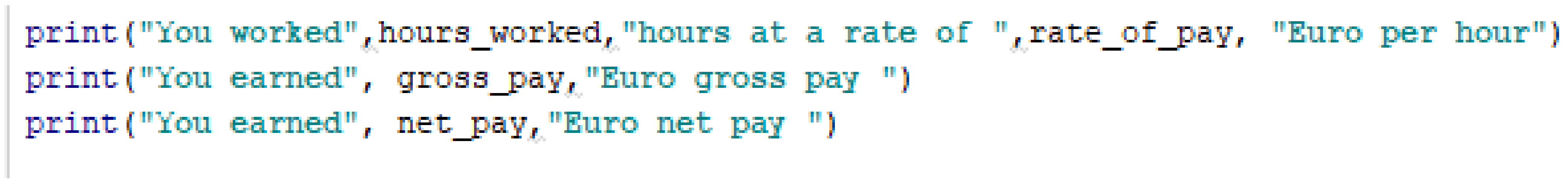
```
TAX = 25.0
TAX_BRACKET = 100.0
```



```
hours_worked = int(input("Enter the number of hours worked - whole number: "))
rate_of_pay = float(input("Enter your pay rate per hour: "))
```



```
gross_pay = rate_of_pay * hours_worked
if gross_pay > TAX_BRACKET:
    net_pay = gross_pay - TAX
else:
    net_pay = gross_pay
```



```
print("You worked",hours_worked,"hours at a rate of ",rate_of_pay, "Euro per hour")
print("You earned", gross_pay,"Euro gross pay ")
print("You earned", net_pay,"Euro net pay ")
```

Class example 3:

Write a program that allows the user to enter a GPA value and the program displays the appropriate award classification based on the following

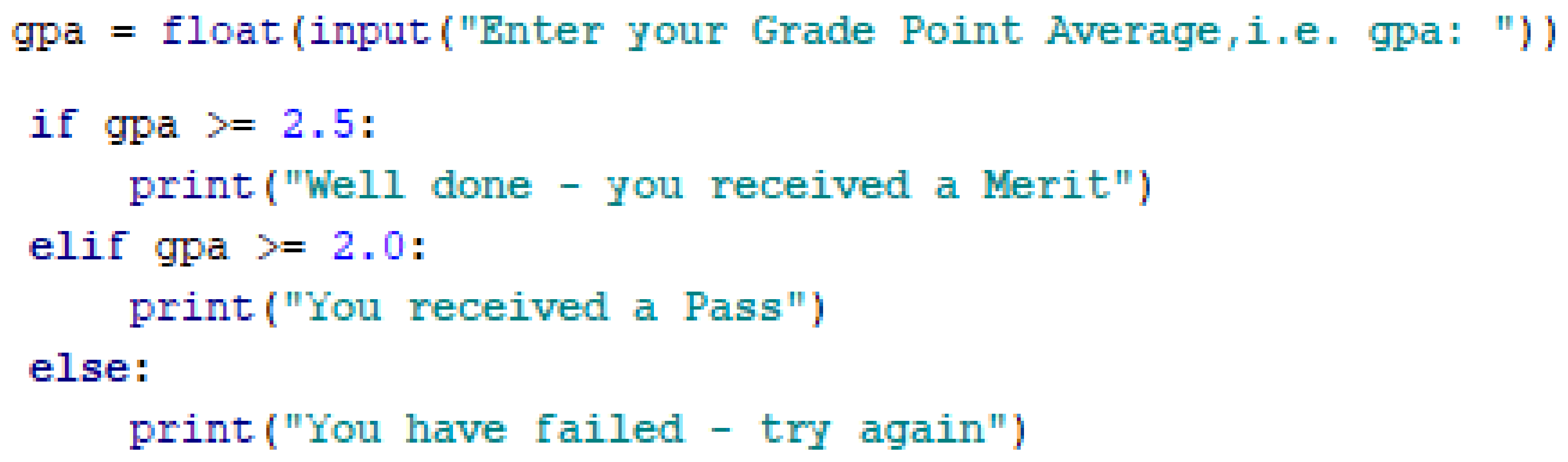
GPA	Award
0 -1.99	Fail
2.0 – 2.49	Pass
2.5 or higher	Merit

- Also calculate and display the number of fails, passes and merits

Sample Solution

```
gpa = float(input("Enter your Grade Point Average,i.e. gpa: "))

num_fails = 0
num_passes = 0
num_merits = 0
if gpa < 2.0:
    print("You have failed - try again")
    num_fails += 1
elif gpa < 2.5:
    print("You received a Pass")
    num_passes += 1
else:
    print("Well done - you received a Merit")
    num_merits += 1

print("Fails", num_fails)
print("Passes", num_passes)
print("Merits", num_merits)
```

Sample Solution



```
gpa = float(input("Enter your Grade Point Average,i.e. gpa: "))

 if gpa >= 2.5:
     print("Well done - you received a Merit")
 elif gpa >= 2.0:
     print("You received a Pass")
 else:
     print("You have failed - try again")
```