**Pearson Edexcel Level 1/Level 2 GCSE (9–1)**

**Tuesday 20 May 2025**

Afternoon (Time: 2 hours)

**Paper reference** **1CP2/02**

# Computer Science
## PAPER 2: Application of Computational Thinking

**You must have:**
- a computer workstation with appropriate programming language code editing software and tools, including an IDE that you are familiar with that shows line numbers
- a 'STUDENT CODING' folder containing code and data files
- printed and electronic copies of the Programming Language Subset (PLS) document (enclosed).

## Instructions

- Answer **all** questions on your computer.
- Save the new or amended code in the 'COMPLETED CODING' folder using the name given in the question.
- Do **not** overwrite the original code and data files provided to you.
- You must **not** use the internet at any time during the examination.

## Information

- The total mark for this paper is 75.
- The marks for **each** question are shown in brackets
  – use this as a guide as to how much time to spend on each question.
- The 'STUDENT CODING' folder in your user area includes all the code and data files you need.

## Advice

- Read each question carefully before you start to answer it.
- Save your work regularly.
- Check your answers and work if you have time at the end.

*Turn over* ▶

**Answer ALL questions.**

**Suggested time: 10 minutes**

1   A program calculates a discount based on the number of two-course meals, three-course meals and drinks ordered by the user.

When the total number of meals exceeds eight, a 15% discount is applied.

When the number of three-course meals exceeds four, a 10% discount is applied.

When the number of two-course meals exceeds two, a 5% discount is applied.

Only one discount can be applied.

When the number of drinks does not match the total number of meals, an error message is displayed. The total must not be calculated or displayed.

Open file **Q01.py**

Amend the code to:

- fix the syntax error on original line 4
  ```
  cost2Courses = 15.00        Cost of meals
  ```

- fix the syntax error on original line 6
  ```
  cost Drinks = 2.00
  ```

- fix the syntax error on original line 10
  ```
  0 = numDrinks
  ```

- fix the TypeError on original line 28
  ```
  total = "num2Courses" * cost2Courses
  ```

- fix the NameError on original line 29
  ```
  total = total + num3Courses * cost33Courses
  ```

- fix the NameError on original line 38
  ```
  total = 0.95 * tatol
  ```

- fix the logic error on original line 33
  ```
  if (num2Courses - num3Courses > 8): # Get 15% discount
  ```

- fix the logic error on original line 34
  ```
  total = 0.85 + total
  ```

- fix the logic error on original line 40
  ```
  print ("Total is:", total)
  ```

Do **not** change the functionality of the given lines of code.

Do **not** add any additional functionality.

Save your amended code file as **Q01FINISHED.py**

**(Total for Question 1 = 9 marks)**

**Suggested time: 10 minutes**

**2**  A program is required to simulate the countdown to a rocket launch.

The user inputs a number between 1 and 10.

The program counts down from that number.

The wait time between each count is 1.5 seconds and is stored in the program code.

Here is the output of a functioning program, when the user enters the number 3

```
What is the countdown? 3
3
2
1
Ignition - lift off!
```

Open file **Q02.py**

Amend the code to:

- import the time library

- create a variable named `countDown` and set it to 0

- take the input from the user and convert it to an integer

- ensure the number inputted is valid by using two relational operators and one logical operator

- ensure that the loop stops when it reaches 0

- display `countDown` for the user

- call a function in the time library to pause the program for the wait time

- reduce `countDown` by 1

- display "Ignition – lift off!" for the user.

Do **not** add any additional functionality.

Save your amended code as **Q02FINISHED.py**

**(Total for Question 2 = 10 marks)**

**Suggested time: 20 minutes**

**3**   A program allows the user to enter a string.

If the string is fewer than three characters long, an error message is displayed and the string is not processed.

Otherwise, the program adds up the ASCII values for each character in the string.

When the total is divisible by four, a four is added to the string.

When the total is divisible by five, a five is added to the string.

When the total is divisible by both four and five, a four is added to the string.

When the total is not divisible by either four or five, a zero is added to the string.

The program displays the total value and the changed string.

The table shows accurate test results for six inputs.

| Input | Output |
|---|---|
| X | String must be three or more characters long |
| XY | String must be three or more characters long |
| cup | 328 cup4 |
| snow | 455 snow5 |
| Ian | 280 Ian4 |
| HAT | 221 HAT0 |
| <empty> | No output, but exits program |

The lines of code in the program are mixed up. The indentation has been done for you.

Open file **Q03.py**

Amend the code to make the program work and produce the correct output. You will need to rearrange the lines.

Do **not** change the functionality of the given lines of code.

Do **not** add any additional functionality.

Save your amended code as **Q03FINISHED.py**

**(Total for Question 3 = 15 marks)**

**Suggested time: 25 minutes**

**4** A program is required to calculate the values in a multiplication square.

Here is an example of a multiplication square.

| × | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 1 | 2 | 3 | 4 | 5 |
| 2 | 2 | 4 | 6 | 8 | 10 |
| 3 | 3 | 6 | 9 | 12 | 15 |
| 4 | 4 | 8 | 12 | 16 | 20 |
| 5 | 5 | 10 | 15 | 20 | 25 |

$2 \times 5 = 10$

The number 10, at the intersection of row 2 and column 5, is the result of $2 \times 5$

The user enters a number for the sides of the square into the program.

The program has these requirements:

- calculate the values for the multiplication square

- write the values to a new file

- separate values with commas

- do not include the row and column labels.

Here are the contents of the output file, when the user enters the number 5

```
1,2,3,4,5
2,4,6,8,10
3,6,9,12,15
4,8,12,16,20
5,10,15,20,25
```

The program is partially complete.

Open file **Q04.py**

Amend the code to meet the requirements.

Use the constants, variables and existing code provided.

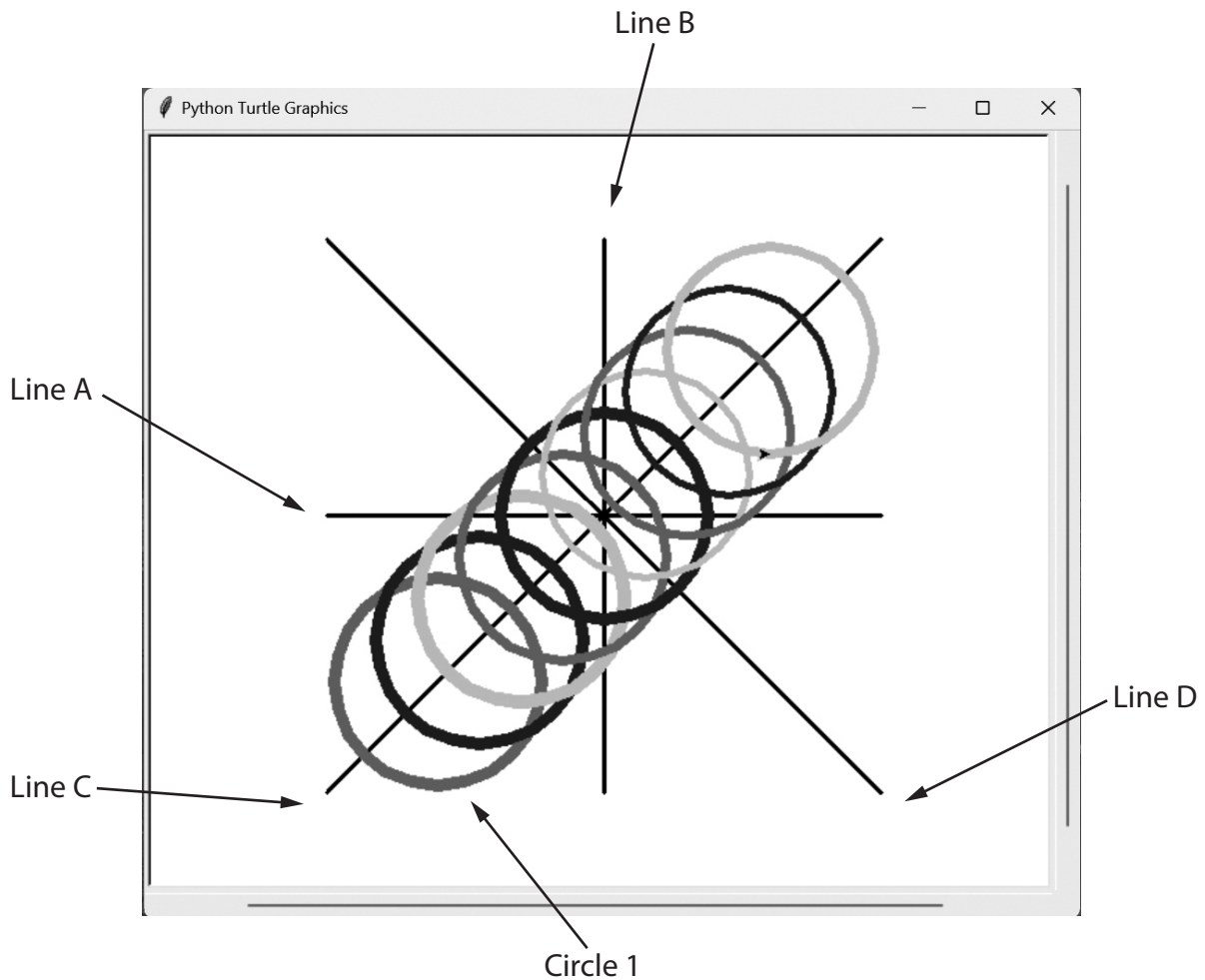Do **not** add any additional functionality.

Save your amended code as **Q04FINISHED.py**

**(Total for Question 4 = 11 marks)**

**5**   A program is being developed to display nine coloured circles using turtle graphics.

Here is a greyscale image of the output produced by a fully functional program.



Line B

Line A

Line D

Line C

Circle 1

**The lines**

- The starting point for:

  – line A is (−200, 0)

  – line B is (0, 200).

- Lines A and B are 400 units long.

- The starting point for:

  – line C is (−200, −200)

  – line D is (200, −200).

- Lines C and D are at 45 degrees and 566 units long.

- The lines are drawn in black with a width of three units.

**The circles**

- There are nine coloured circles.

- All circles have a radius of 75 units.

- The position of circle 1 is stored in the variables $xPos$ and $yPos$.

- The second circle is located at $xPos + 30$ and $yPos + 30$.

- The circle colours are stored in a one-dimensional array. Each colour is used in sequence. The colours of the circles will be the same each time the program is run.

- The width of the line to draw a circle is a random integer between five and ten. The width of the line may be different for each circle and may be different each time the program is run.

**The turtle**

- The speed of the turtle must be as quick as possible.

**The graphics screen**

- The code to create the graphics screen is provided.

- The code to keep the graphics screen open is provided.


Open file **Q05.py**


Amend the code to create the image shown.

Use comments, white space and layout to make the program easier to read and understand.

Use the constants, variables and existing code provided.


Do **not** add any additional functionality.

Save your amended code file as **Q05FINISHED.py**


**(Total for Question 5 = 15 marks)**

**Suggested time: 30 minutes**

**6** A set of records is stored in a two-dimensional array.

The fields in each record are a key and a signal reading.

A valid key:

· has a minimum length of three

· is a group of letters and digits

· ends with a valid check digit.

The table shows how to calculate the check digit for any key.

| Key | Sum the digits only in the key | Check digit = right-most digit of sum |
|---|---|---|
| A2B3C5 | 2 + 3 = 5 | 5 |
| A7B30 | 7 + 3 = 10 | 0 |
| A11 | 1 | 1 |
| A274B3 | 2 + 7 + 4 = 13 | 3 |

A valid signal reading must be a real number:

· greater than or equal to 1.0

· less than or equal to 5.0

Some of the records have errors. Eight records have invalid keys. Four records have invalid signals.

Open file **Q06.py**

Write a program to meet these requirements:

· find and display each invalid record, including the key and the signal

· inform the user if a record has an invalid key (no reason required)

· inform the user if a record has an invalid signal (no reason required)

· work even if the number of records is changed

· include at least one user-devised subprogram.

Use comments, white space and layout to make the program easier to read and understand.

Do **not** add any additional functionality.

Save your amended code as **Q06FINISHED.py**

<div align="right">

**(Total for Question 6 = 15 marks)**

**TOTAL FOR PAPER = 75 MARKS**

</div>

**BLANK PAGE**

**Pearson Edexcel Level 1/Level 2 GCSE (9–1)**

**Tuesday 20 May 2025**

Afternoon (Time: 2 hours)

**Paper reference** **1CP2/02**

# Computer Science
**PAPER 2:** Application of Computational Thinking
**Programming Language Subset**
**Version 6**

**PLS Booklet**
**Do not return this Booklet with the question paper.**

P78600A

Pearson

# Contents

**Introduction**

The Programming Language Subset (PLS) is a document that specifies which parts of Python 3 are required in order that the assessments can be undertaken with confidence. Students familiar with everything in this document will be able to access all parts of the Paper 2 assessment. This does not stop a teacher/student from going beyond the scope of the PLS into techniques and approaches that they may consider to be more efficient or engaging.

Pearson will **not** go beyond the scope of the PLS when setting assessment tasks. Any student successfully using more esoteric or complex constructs or approaches not included in this document will still be awarded marks in Paper 2 if the solution is valid.

The pair of <> symbols indicates where expressions or values need to be supplied. They are not part of the PLS.

**Comments**

Anything on a line after the character # is considered a comment.

**Identifiers**

Identifiers are any sequence of letters, digits and underscores, starting with a letter.

Both upper and lower case are supported.

**Data types and conversion**

**Primitive data types**

Variables may be explicitly assigned a data type during declaration.

Variables may be implicitly assigned a data type during initialisation.

Supported data types are:

| Data type | PLS |
|-----------|-----|
| integer | int |
| real | float |
| Boolean | bool |
| character | str |

**Conversion**

Conversion is used to transform the data types of the contents of a variable using int(), str(), float(), bool() or list(). Conversion between any allowable types is permitted.

**Constants**

Constants are conventionally named in all uppercase characters.

**Combining declaration and initialisation**

The data type of a variable is implied when a variable is assigned a value.

## Structured data types

A structured data type is a sequence of items, which themselves are typed. Sequences start with an index of zero.

| Data type | Explanation | PLS |
|---|---|---|
| string | A sequence of characters | str |
| array | A sequence of items with the same (homogeneous) data type | list |
| record | A sequence of items, usually of mixed (heterogenous) data types | list |

## Dimensions

The number of dimensions supported by the PLS is two.

| | |
|---|---|
| `<structure>[<first dimension>]` | Accesses an item in a one-dimensional structure |
| `<structure>[<first dimension>]`<br>`[<second dimension>]` | Accesses an item in a two-dimensional structure |

The PLS does not support ragged data structures. Therefore, in a list of records, each record will have the same number of fields.

**Operators**

**Arithmetic operators**

| Arithmetic operator | Meaning |
|:---:|:---:|
| / | division |
| * | multiplication |
| ** | exponentiation |
| + | addition |
| − | subtraction |
| // | integer division |
| % | modulus |

**Relational operators**

| Relational operator | Meaning |
|:---:|:---:|
| == | equal to |
| != | not equal to |
| > | greater than |
| >= | greater than or equal to |
| < | less than |
| <= | less than or equal to |

**Logical/Boolean operators**

| Operator | Meaning |
|:---:|:---:|
| and | both sides of the test must be true to return true |
| or | either side of the test must be true to return true |
| not | inverts |

## Programming constructs

### Assignment

Assignment is used to set or change the value of a variable.

| |
|---|
| `<variable identifier> = <value>` |
| `<variable identifier> = <expression>` |

### Sequence

Every instruction comes one after the other, from the top of the file to the bottom of the file.

### Blocking

Blocking of code segments is indicated by indentation and subprogram calls. These determine the scope and extent of variables they declare.

### Selection

| | |
|---|---|
| `if <expression>:`<br>`    <command>` | If <expression> is true, then <command> is executed. |
| `if <expression>:`<br>`    <command>`<br>`else:`<br>`    <command>` | If <expression> is true, then first <command> is executed, otherwise second <command> is executed. |
| `if <expression>:`<br>`    <command>`<br>`elif <expression>:`<br>`    <command>`<br>`else:`<br>`    <command>` | If <expression> is true, then first <command> is executed, otherwise the second <expression> test is checked. If true, then second <command> is executed, otherwise third <command> is executed.<br><br>Supports multiple instances of 'elif'.<br><br>The 'else' is optional with the 'elif'. |

### Repetition

| | |
|---|---|
| `while <condition>:`<br>`    <command>` | Pre-conditioned loop. This executes <command> while <condition> is true. |

### Iteration

| | |
|---|---|
| `for <id> in <structure>:`<br>`    <command>` | Executes <command> for each element of <structure>, in one dimension. |
| `for <id> in range (<start>, <stop>):`<br>`    <command>` | Count-controlled loop. Executes <command> a fixed number of times, based on the numbers generated by the range function. <stop> is required. <start> is optional. |
| `for <id> in range (<start>, <stop>,`<br>`    <step>):`<br>`    <command>` | Same as above, except that <step> influences the numbers generated by the range function. <stop> is required. <start> and <step> are optional. |

## Subprograms

| | |
|---|---|
| ```def <procname> ():```<br>    ```<command>``` | A procedure with no parameters |
| ```def <procname> (<paramA>, <paramB>):```<br>    ```<command>``` | A procedure with parameters |
| ```def <funcname> ():```<br>    ```<command>```<br>    ```return (<value>)``` | A function with no parameters |
| ```def <funcname> (<paramA>, <paramB>):```<br>    ```<command>```<br>    ```return (<value>)``` | A function with parameters |

## Inputs and outputs

### Screen and keyboard

| | |
|---|---|
| ```print (<item>)``` | Displays <item> on the screen |
| ```input (<prompt>)``` | Displays <prompt> on the screen and returns the line typed in |

## Files

The PLS supports manipulation of comma separated value text files.

File operations include open, close, read, write and append.

| | |
|---|---|
| ```<fileid> = open (<filename>, "r")``` | Opens file for reading |
| ```for <line> in <fileid>:``` | Reads every line, one at a time |
| ```<alist> = <fileid>.readlines ()``` | Returns a list where each item is a line from the file |
| ```<aline> = <fileid>.readline ()``` | Returns a line from a file. Returns an empty string on the end of the file |
| ```<fileid> = open (<filename>, "w")``` | Opens a file for writing |
| ```<fileid> = open (<filename>, "a")``` | Opens a file for appending |
| ```<fileid>.writelines (<structure>)``` | Writes <structure> to a file. <structure> is a list of strings |
| ```<fileid>.write (<aString>)``` | Writes a single string to a file |
| ```<fileid>.close ()``` | Closes file |

**Supported subprograms**

**Built-in subprograms**

The PLS supports these built-in subprograms.

| Subprogram | Description |
|---|---|
| `bool (<item>)` | Returns <item> converted to the equivalent Boolean value |
| `chr (<integer>)` | Returns the string which matches the Unicode value of <integer>. The first 128 characters of Unicode are equivalent to ASCII. |
| `float (<item>)` | Returns <item> converted to the equivalent real value |
| `input (<prompt>)` | Displays <prompt> on the screen and returns the line typed in |
| `int (<item>)` | Returns <item> converted to the equivalent integer value |
| `len (<object>)` | Returns the length of the <object>, such as a string, one-dimensional or two-dimensional data structure |
| `ord (<char>)` | Returns the integer equivalent to the Unicode string of the single character <char>. The first 128 characters of Unicode are equivalent to ASCII. |
| `print (<item>)` | Displays <item> on the screen |
| `range (<start>, <stop>, <step>)` | Generates a list of numbers using <step>, beginning with <start> and up to, but not including, <stop>.  A negative value for <step> goes backwards. <stop> is required.  <start> and <step> are optional.  The default value for <start> is zero. The default value for <step> is positive one. |
| `round (<x>, <n>)` | Rounds <x> to the number of <n> digits after the decimal (uses the 0.5 rule). The <n> is optional. If omitted, the function returns the nearest integer to <x>. |
| `str (<item>)` | Returns <item> converted to the equivalent string value |

**List subprograms**

The PLS supports these list subprograms.

| Subprogram | Description |
|---|---|
| `<list>.append (<item>)` | Adds <item> to the end of the list |
| `del <list> [<index>]` | Removes the item at <index> from list |
| `<list>.insert (<index>, <item>)` | Inserts <item> just before an existing one at <index> |
| `<aList> = list ()`<br>`<aList> = []` | Two methods of creating a list structure. Both are empty. |

**String subprograms**

The PLS supports these string subprograms.

| Subprogram | Description |
|---|---|
| `len (<string>)` | Returns the length of <string> |
| `<string>.find (<substring>, <start>, <end>)` | Returns the location of the first instance of <substring> in the original <string>, reading from left to right. <start> is the index to begin the find. The default is zero. <end> is the index to stop the find. The default is the end of <string>. Returns -1, if not found. |
| `<string>.index (<substring>, <start>, <end>)` | Returns the location of the first instance of <substring> found in the original <string> as read from left to right. Raises an exception if not found. <substring> is required. <start> and <end> are optional.  The default value for <start> is zero. The default value for <end> is the end of <string>. |
| `<string>.isalpha ()` | Returns True, if all characters are alphabetic A–Z |
| `<string>.isalnum ()` | Returns True, if all characters are alphabetic A–Z or digits 0–9 |
| `<string>.isdigit ()` | Returns True, if all characters are digits 0–9, exponents are digits |
| `<string>.replace (<s1>, <s2>)` | Returns original <string> with all occurrences of <s1> replaced with <s2> |
| `<string>.split (<char>)` | Returns a list of all substrings in original <string>, using <char> as the separator |
| `<string>.strip (<char>)` | Returns original <string> with all occurrences of <char> removed from the front and back |
| `<string>.upper ()` | Returns original <string> in uppercase |
| `<string>.lower ()` | Returns original <string> in lowercase |
| `<string>.isupper ()` | Returns True, if all characters in <string> are uppercase |
| `<string>.islower ()` | Returns True, if all characters in <string> are lowercase |
| `<string>.format (<values>)` | Formats <values> and puts them into <string>. The content of <string> is described by symbols and placeholders. |

## Formatting strings

Output can be customised to suit the problem requirements and the user's needs by forming string output.

<string>.format () can be used with positional placeholders and format descriptors.

Placeholders take the form:

`{:<align><sign><width><.precision><type>}`

| Placeholder | Option | Description |
|---|---|---|
| align | < | Left aligned. Default for most items, like text. |
| | > | Right aligned. Default for numbers. |
| | ^ | Centre aligned. |
| sign | + | Use a sign for both positive and negative numbers. |
| | – | Use a sign only for negative numbers. Default for negative numbers. |
| | space | Use leading spaces for positive numbers and a minus sign for negative numbers. |
| width | whole number | The total width of the field. |
| precision | whole number | The number of digits after the decimal. |
| type | s | String. Default for strings, if not supplied. |
| | d | Numbers in base 10 (denary). Default for integers, if not supplied. |
| | f | Fixed-point notation. Formats a number with exactly the number of digits to the right of the decimal given by precision |

Here is an example:

```
layout = "{:>10} {:^5d} {:7.4f}"
print (layout.format ("Fred", 358, 3.14159))
      Fred  358    3.1416
```

The * operator can be used to generate a line of repeated characters, for example: "=" * 10 will generate "==========".

Concatenation of strings is done using the + operator.

String slicing is supported. myName[0:2] gives the first two characters in the variable myName.

**Library modules**

The functionality of a library module can only be accessed once the library module is imported into the program code.

| Statement | Description |
|---|---|
| `import <library>` | Imports the <library> module into the current program |

**Random library module**

The PLS supports these random library module subprograms.

| Subprogram | Description |
|---|---|
| `random.randint (<a>, <b>)` | Returns a random integer X so that <a> <= X <= <b> |
| `random.random ()` | Returns a float number in the range of 0.0 and 1.0 |

**Math library module**

The PLS supports these math library module subprograms and constant.

| Subprogram or constant | Description |
|---|---|
| `math.ceil (<x>)` | Returns the smallest integer greater than or equal to <x> |
| `math.floor (<x>)` | Returns the largest integer less than or equal to <x> |
| `math.sqrt (<x>)` | Returns the square root of <x> |
| `math.pi` | The constant Pi ($\Pi$) |

**Time library module**

The PLS supports this time library module subprogram.

| Subprogram | Description |
|---|---|
| `time.sleep (<sec>)` | The current process is suspended for the given number of seconds, then resumes at the next line of the program |

**Turtle graphics library module**

**Tips for using turtle**

The default mode for the PLS turtle is "standard". This means that when a turtle is created, it initially points to the right (east) and angles are counterclockwise. You can change modes using turtle.mode ().

The turtle window is one size and the turtle drawing canvas (inside the window) can be a different size. To make the turtle window bigger, a screen needs to be created and set up. Here is an example:

```
WIDTH = 800
HEIGHT = 400
screen = turtle.Screen ()
screen.setup (WIDTH, HEIGHT)
```

To make the drawing canvas bigger use <turtle>.screensize ().

In some development environments, the turtle window will close as soon as the program completes. There are two ways to keep it open:

- Add turtle.done () as the last line in the code file. This will keep the window open until closed with the exit cross in the upper right-hand corner. It also allows scrollbars on the window.

- Add a line asking for keyboard input, such as input (), as the last line. This will keep the window open until the user presses a key in the console session. The scrollbars will not work.

**Turtle window and drawing canvas**

The PLS supports these turtle library module subprograms to control the window and drawing canvas. Notice that these subprograms do not use the name of the turtle you create to the left of the dot, but the library name, "turtle" or a <window> variable.

| Subprogram | Description |
|---|---|
| `<window>.setup (<width>, <height>)` | Sets the size of the turtle window to <width> × <height> in pixels. Requires use of turtle.Screen () to create <window> first. |
| `turtle.done ()` | Use as the last line of the file to keep the turtle window open until it is closed using the exit cross in the upper right-hand corner of the window |
| `turtle.mode (<type>)` | <type> is one of the strings "standard" or "logo". A turtle in standard mode, initially points to the right (east) and angles are counterclockwise. A turtle in logo mode, initially points up (north) and angles are clockwise. |
| `turtle.Screen ()` | Returns a variable to address the turtle window. Use with <window>.setup (). |
| `turtle.screensize (<width>, <height>)` | Makes the scrollable drawing canvas size equal to <width> × <height> in pixels. Note, use with turtle.done () so scrollbars will be active. |

**Turtle creation, visibility and movement**

The PLS supports these turtle library module subprograms to control the turtle creation, visibility and movement.

| Subprogram | Description |
|---|---|
| `<turtle> = turtle.Turtle ()` | Creates a new turtle with the variable name <turtle> |
| `<turtle>.back (<steps>)` | Moves backward (opposite-facing direction) for number of <steps> |
| `<turtle>.forward (<steps>)` | Moves forward (facing direction) for number of <steps> |
| `<turtle>.hideturtle ()` | Makes the <turtle> invisible |
| `<turtle>.left (<degrees>)` | Turns counterclockwise the number of <degrees> |
| `<turtle>.right (<degrees>)` | Turns clockwise the number of <degrees> |
| `<turtle>.showturtle ()` | Makes <turtle> visible |
| `<turtle>.speed (<value>)` | The <value> can be set to "fastest", "fast", "normal", "slow", "slowest". Alternatively, use the numbers 1 to 10 to increase speed. The value of 0 is the fastest. |

**Turtle positioning and direction**

The PLS supports these turtle library module subprograms to control the positioning and direction.

| Subprogram | Description |
|---|---|
| `<turtle>.home ()` | Moves to canvas origin (0, 0) |
| `<turtle>.reset ()` | Clears the drawing canvas, sends the turtle home and resets variables to default values |
| `<turtle>.setheading (<degrees>)` | Sets the orientation to <degrees> |
| `<turtle>.setposition (<x>, <y>)` | Positions the turtle at coordinates (<x>, <y>) |

**Turtle filling shapes**

The PLS supports these turtle library module subprograms to control filling.

| Subprogram | Description |
|---|---|
| `<turtle>.begin_fill ()` | Call just before drawing a shape to be filled |
| `<turtle>.end_fill ()` | Call just after drawing the shape to be filled. You must call <turtle>.begin_fill () before drawing |
| `<turtle>.fillcolor (<colour>)` | Sets the colour used to fill. The input argument is a string, for example: "red". |

### Turtle controlling the pen

The PLS supports these turtle library module subprograms to control the pen.

| Subprogram | Description |
|---|---|
| `<turtle>.pencolor (<colour>)` | Sets the colour of the pen. The input argument is a string or an RGB colour, for example: "red". |
| `<turtle>.pendown ()` | Puts the pen down |
| `<turtle>.pensize (<width>)` | Makes the pen the size of <width> (positive number) |
| `<turtle>.penup ()` | Lifts the pen up |

### Turtle circles

The PLS supports this turtle library module subprogram to draw a circle.

| Subprogram | Description |
|---|---|
| `<turtle>.circle (<radius>, <extent>)` | Draws a circle with the given <radius>. The centre is the <radius> number of units to the left of the turtle. That means, the turtle is sitting on the edge of the circle. The parameter <extent> does not need to be given, but provides a way to draw an arc, if required. An extent of 180 would be half a circle. |

### Turtle colours

Python colours can be given by using a string name. There are many colours and you can find information online for lists of all the available colours.

Here are a few to get you started:

| | | | |
|---|---|---|---|
| blue | black | green | yellow |
| orange | red | pink | purple |
| indigo | olive | lime | navy |
| orchid | salmon | peru | sienna |
| white | cyan | silver | gold |

### Console session

A console session is the window or command line where the user interacts with a program. It is the default window that displays the output from `print ()` and echoes the keys typed from the keyboard.

It will appear differently in different development tools.

### Code style

Although Python does not require all arithmetic and logical/Boolean expressions to be fully bracketed, it might help the readability to bracket them. This is especially useful if the programmer or reader is not familiar with the order of operator precedence.

The same is true of spaces. The logic of a line can be more easily understood if a few extra spaces are introduced. This is especially helpful if a long line of nested subprogram calls is involved. It can be difficult to read where one ends and another begins. The syntax of Python is not affected, but it can make understanding the code much easier.

**Line continuation**

Long code lines may also be difficult to read, especially if they scroll off the edge of the display window. It's always better for the programmer to limit the amount of scrolling.

There are several ways to break long lines in Python.

Python syntax allows long lines to be broken inside brackets () and square brackets []. This works very well, but care should be taken to ensure that the next line is indented to a level that aids readability. It is even possible and recommended to add an extra set of brackets () to expressions to break long lines.

Python also has a line continuation character, the backslash \ character. It can be inserted, following strict rules, into some expressions to cause a continuation. Some editors will automatically insert the line continuation character if the enter key is pressed.

**Carriage return and line feed**

These affect the way outputs appear on the screen and in a file. Carriage return means to go back to the beginning of the current line without going down to the next line. Line feed means to go down to the next line. Each is a non-printable ASCII character, that has an equivalent string in programming languages.

| Name | Abbreviation | ASCII hexadecimal | String |
|---|---|---|---|
| Carriage return | CR | 0x0D | "\r" |
| Line feed | LF | 0x0A | "\n" |

These characters are used in some combination to control outputs. Unfortunately, not every operating system uses the same. However, editors automatically convert input and output files to make sure they work properly. In Python, `print ()` automatically adds them so that the console output appears on separate lines.

When writing code to handle files, a programmer will need to remove some of these characters when reading lines from files and add them when writing lines to files. If needed, they are added with string concatenation. If needed to be removed, they are removed using the `strip ()` subprogram.

**BLANK PAGE**

**BLANK PAGE**