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

Tuesday 21 May 2024

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 which shows line numbers
- a 'STUDENT CODING' folder containing code and data files
- printed and electronic copies of the Program 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 ▶

P75441RA

F:1/1/1/1/1/1/1/1/1/1

P 7 5 4 4 1 R A

Pearson

Answer ALL questions.

Suggested time: 10 minutes

1 A program is written to provide information about the rainbow.

Colours and wavelengths are stored in arrays. For example, the colour Violet is produced when the wavelength is from 380 to 424

The user enters an index and the colour at that array location is displayed. The user enters a wavelength and the colour for that wavelength is displayed.

Open file **Q01.py**

Amend the code to:

- fix the syntax error on original line 5
```
waveTable = [380, 425, 450, 492, 577, 597", 622]
```
- fix the NameError on original line 6
```
found = false
```
- fix the syntax error on original line 8
```
wavelength = 0123
```
- fix the syntax error on original line 15
```
index = int (input ("Enter an index: ")
```
- fix the NameError on original line 21
```
color = rainbow[index]
```
- fix the ValueError on original line 22
```
print (int (colour))
```
- fix the logic error on original line 26
```
if ((wavelength < 380) and (wavelength > 622)):
```
- fix the logic error on original line 29
```
index = 1
```
- fix the logic error on original line 35
```
elif (waveTable[index] <= wavelength):
```
- fix the logic error on original line 37
```
print (rainbow[index - 2])
```

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

Do **not** add any additional functionality.

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

(Total for Question 1 = 10 marks)

Suggested time: 15 minutes

2 A program encrypts a message using a Caesar cipher.

The letters of the alphabet are shifted a set number of places. A positive shift moves the letters to the right. A negative shift moves them to the left.

Blanks, symbols and numbers are not encrypted.

When the end of the alphabet is reached with a positive shift, shifting continues at the start of the alphabet. When the start of the alphabet is reached with a negative shift, shifting continues at the end of the alphabet.

For example, a shift of –2 encodes the plaintext letter **P** to the ciphertext letter **N**.
A shift of +4 encodes the plaintext letter **X** to the ciphertext letter **B**.

When working correctly, the program produces the encrypted ciphertext for these plaintext messages and shift inputs.

Plaintext	Shift	Ciphertext
The Rainbow	4	Xli Vemrfsa
Alphabet Soup	–5	Vgkcvwzo Njpk
123 ^&* Bye	9	123 ^&* Khn

Open file **Q02.py**

Amend the code to make the program work and produce the correct output.

You will need to:

- choose between alternative lines of code. Make a choice by removing the # at the beginning of the line you choose to execute
- run the program with the data from the table and check it meets the requirements.

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

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 is required to calculate the total cost of items purchased. Items are sold by count or by weight.

The user enters the number 1 if the purchase is by count or the number 5 if the purchase is by weight of items. The user then enters either the count or the weight in kilograms.

The program checks for invalid input. If the input is valid, the program calculates and displays the total cost.

Currency formatting with two decimal places and a symbol is not required.

The table shows test results.

Purchase category	Count of items	Weight in kilograms	Output
1	3		Total cost is 3.69
1	0		Invalid number of items
5		4.5	Total cost is 15.525
5		–6.6	Invalid weight
3			Invalid purchase type

Open file **Q03.py**

Amend the code to:

- create an integer variable named `purchaseType` and set it to 0
- complete a line with the correct logical operator and the correct constant
- complete a line with the correct constant
- complete a line to accept a real value for the weight in kilograms
- complete a line to calculate the total cost based on weight
- complete a line to check for a 0 or negative count of items
- complete a line with the correct relational operator
- add a line to display an informative message and the total cost.

Do **not** add any additional functionality.

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

(Total for Question 3 = 10 marks)

BLANK PAGE

Suggested time: 25 minutes

4 A small café provides cheese rolls and crisps for community events.

The table shows the amount of ingredients for each adult and each child attending an event.

Ingredient	Adult	Child
Crisps	0.75 of a bag	0.33 of a bag
Cheese	40 grams	30 grams
Rolls	1.5 rolls	0.5 of a roll

An algorithm is used to determine how many crisps, how many rolls, and how much cheese to order for an event.

Crisps are ordered by whole bags. A partial bag cannot be ordered.

Rolls are ordered by pack. A pack contains 24 rolls. A partial roll or a partial pack cannot be ordered.

Cheese is ordered by pack. A pack of cheese weighs 500 grams. A partial pack of cheese cannot be ordered.

The flowchart shows the algorithm for calculating the amount of ingredients that must be ordered.

Open file **Q04.py**

Write the code to implement the algorithm in the flowchart.

Use the library and constants provided.

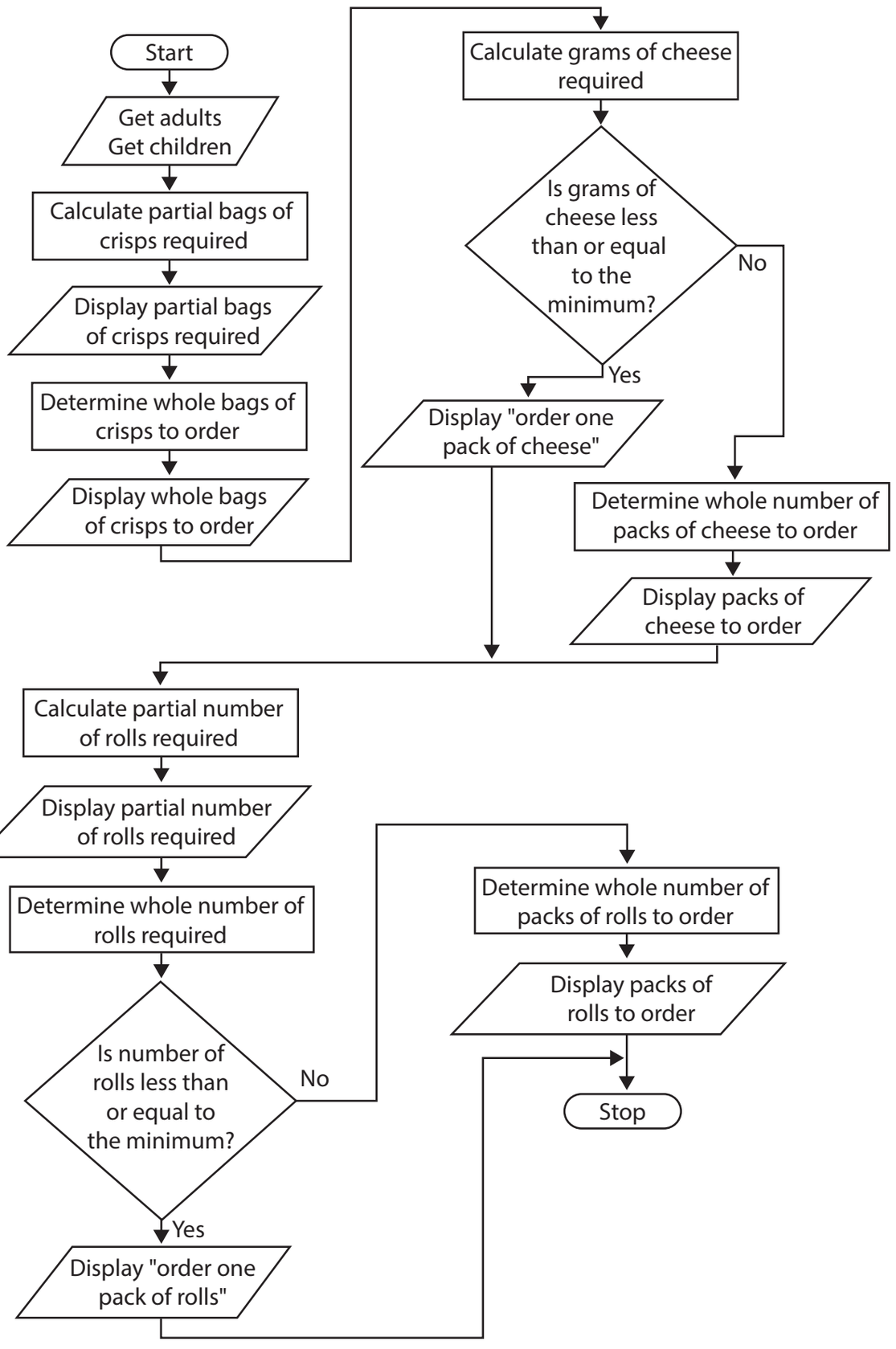
Use informative messages for the user.

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 **Q04FINISHED.py**

(Total for Question 4 = 15 marks)


Start
Get adults
Get children
Calculate partial bags of crisps required
Display partial bags of crisps required
Determine whole bags of crisps to order
Display whole bags of crisps to order
Calculate grams of cheese required
Is grams of cheese less than or equal to the minimum?
No
Yes
Display "order one pack of cheese"
Determine whole number of packs of cheese to order
Display packs of cheese to order
Calculate partial number of rolls required
Display partial number of rolls required
Determine whole number of rolls required
Is number of rolls less than or equal to the minimum?
No
Yes
Display "order one pack of rolls"
Determine whole number of packs of rolls to order
Display packs of rolls to order
Stop

Suggested time: 25 minutes

5 A program is being developed to work with pasta shapes. The pasta shape names are stored in a one-dimensional array.

The user is shown a menu of options. When the user chooses an option, the program performs the task.

The program loops until the user chooses the exit menu option.

Open file **Q05.py**

Amend the code to:

- complete the `getChoice()` subprogram
  - allow the user to enter a menu option number (no validation required)
  - return the choice to the main program
- complete the `getShape()` subprogram
  - use a random number as an index into the `pastaShapes` array
  - return the shape to the main program
- complete the `addShape()` subprogram
  - allow the user to enter a new pasta shape name
  - add the new name to the end of the `pastaShapes` array
- complete the main program
  - call each subprogram based on the menu option entered by the user
  - display the shape from the `getShape()` subprogram
  - tell the user when an inputted option number is not on the menu.

Use the constants and variables provided.

Do **not** add any additional functionality.

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

(Total for Question 5 = 15 marks)

Suggested time: 25 minutes

6 A program is required to process data about cows. The data is stored in a comma separated value text file named Cows.txt

The columns in the data file are:

- name
- breed
- tag number.

Open file **Q06.py**

Write a program to meet these requirements:

- create a key for each cow in the data. A valid key is a single string consisting of (in this order)
  - the first two letters of the breed name
  - the tag number integer divided by 100
  - the first two letters of the cow's name
- create a record for each cow. A valid record consists of (in this order)
  - a key, a tag number, a name and a breed
- store the record for each cow in the `cowTable`
- call the supplied subprogram, `showTable()`, to display the contents of the `cowTable`
- the program must work with any number of lines in the data file.

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**

(Total for Question 6 = 15 marks)

TOTAL FOR PAPER = 75 MARKS

BLANK PAGE

BLANK PAGE

BLANK PAGE

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

Tuesday 21 May 2024

Afternoon (Time: 2 hours)

Paper reference **1CP2/02**

Computer Science

PAPER 2: Application of Computational Thinking
Programming Language Subset
Version 5

PLS Booklet

Do not return this Booklet with the question paper.

Turn over ▶

P75441RA

F:1/1/1/1/1/1/1/1/1/1

P 7 5 4 4 1 R A

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.

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>:` `  <command>`	If <expression> is true, then command is executed.

`if <expression>:`
`  <command>`
`else:`
`  <command>`
	If <expression> is true, then first <command> is executed, otherwise second <command> is executed.

`if <expression>:`
`  <command>`
`elif <expression>:`
`  <command>`
`else:`
`  <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.
	
	Supports multiple instances of 'elif'.
	
	The 'else' is optional with the 'elif'.

Repetition

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

Iteration

`for <id> in <structure>:` `  <command>`	Executes <command> for each element of a data structure, in one dimension.
`for <id> in range (<start>, <stop>):` `  <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>,`
`  <step>):`
`  <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> ():` `  <command>`	A procedure with no parameters
`def <procname> (<paramA>, <paramB>):` `  <command>`	A procedure with parameters

`def <funcname> ():`
`  <command>`
`  return (<value>)`
	A function with no parameters

`def <funcname> (<paramA>, <paramB>):`
`  <command>`
`  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 the content of prompt to the screen and waits for the user to type in characters followed by a new line
`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>)`	Prints <item> to the display
`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 ()` `<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 the 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 the 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 the original, 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 the original string in uppercase
`<string>.lower ()`	Returns the original string in lowercase
`<string>.isupper ()`	Returns True, if all characters are uppercase
`<string>.islower ()`	Returns True, if all characters are lowercase
`<string>.format (<placeholders>)`	Formats values and puts them into the <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 (<r>)`	Returns the smallest integer not less than <r>
`math.floor (<r>)`	Returns the largest integer not greater than <r>
`math.sqrt (<x>)`	Returns the square root of <x>
`math.pi`	The constant Pi (Π)

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 anticlockwise the number of <degrees>
`<turtle>.right (<degrees>)`	Turns clockwise the number of <degrees>
`<turtle>.showturtle ()`	Makes the 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

BLANK PAGE