Pearson Edexcel Level 1/Level 2 GCSE (9–1) in Computer Science (1CP2)

Programming Language Subset

Issue 1

Edexcel, BTEC and LCCI qualifications

Edexcel, BTEC and LCCI qualifications are awarded by Pearson, the UK's largest awarding body offering academic and vocational qualifications that are globally recognised and benchmarked. For further information, please visit our qualification website at qualifications.pearson.com. Alternatively, you can get in touch with us using the details on our contact us page at qualifications.pearson.com/contactus

About Pearson

Pearson is the world's leading learning company, with 35,000 employees in more than 70 countries working to help people of all ages to make measurable progress in their lives through learning. We put the learner at the centre of everything we do, because wherever learning flourishes, so do people. Find out more about how we can help you and your learners at qualifications.pearson.com

Contents

Introduction **1**

Comments **2**

Identifiers **2**

Data types and conversion **2**

Primitive data types 2

Conversion 2

Constants 2

Combining delcaration and initialisation 2

Structured data types **3**

Dimensions 3

Operators **3**

Arithemetic operators 3

Relational operators 3

Boolean operators 4

Programming constructs **4**

Assignment 4

Sequence 4

Blocking 4

Selection 4

Repetition 5

Iteration 5

Subprograms 5

Inputs and outputs **6**

Screen and keyboard 6

Files 6

Library modules **7**

Built-in functions 7

List methods 7

Random module 8

String module 8

Formatting strings 9

Math module 9

Time module 9

Turtle graphics module 10

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 indicate 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 upper-case 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	Operation
/	division
*	multiplication
**	exponentiation
+	addition
-	subtraction
//	integer division
%	modulus

Relational operators

Operator	Operator meaning
==	equal to
!=	not equal to
>	greater than
>=	greater than or equal to
<	less than
<=	less than or equal to

Boolean operators

Operator
and
or
not

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 and if true, then second <command> is executed, otherwise third <command> is executed.
	Supports multiple instances of 'elif'.
	The 'else' is not required 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.
for <id> in range (<start>, <stop>, <step>): <command>	Same as above, except that <step> influences the numbers generated by the range function.

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 end of the file
`<fileid> = open(<filename>, "w")`	Opens file for writing
`<fileid> = open(<filename>, "a")`	Opens 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

Library modules

Built-in functions

The PLS supports these built-in functions by default. No import statements are necessary.

Function	Description
`chr(integer)`	Returns the string which matches the Unicode value of integer.
`input(prompt)`	Displays the content of prompt to the screen and waits for the user to type in characters followed by a new line.
`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 a single 'char'.
`print(item)`	Prints item to the display.
`range(start, stop, step)`	The range() function is used to generate a list of numbers, beginning with the first and up to but not including the last, using step.
`round(x, n)`	Rounds x to the number of n digits after the decimal (uses the 0.5 rule)

List methods

No import necessary.

The supported functionality is as follows:

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

Random module

Import the random module.

The supported functionality is as follows:

Function	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

String module

No import necessary.

The supported functionality is as follows:

Function	Description
`len(<string>)`	Returns the length of <string>
`find(<string>)`	Returns the location of <string> in the original. Returns -1, if not found
`index(<string>)`	Returns the location of <string> in the original. Raises an exception if not found
`isalpha(<string>)`	Returns True, if all characters are alphabetic (a–z)
`isalnum(<string>)`	Returns True, if all characters are alphabetic (a–z) and digits (0–9)
`isdigit(<string>)`	Returns True, if all characters are digits (0–9), exponents are digits
`isnumeric(<string>)`	Returns True, if all characters are numeric (0–9), exponents and fractions are numeric
`replace(s1,s2)`	Returns original string with all occurrences of s1 replaced with s2
`split(<char>)`	Returns a list of all substrings in the original, using <char> as the separator
`strip(<char>)`	Returns original string with all occurrences of 'char' removed from front and back
`upper(<string>)`	Returns the original string in upper case
`lower(<string>)`	Returns the original string in lower case
`isupper(<string>)`	Returns True, if all characters are upper case
`islower(<string>)`	Returns True, if all characters are lower case

Concatenation of strings is done using the + operator.

Formatting strings

Output can be customised to suit the problem requirements and the user's needs by using the `string.format()` method, with positional parameters.

Here is an example:

```
"{:>10} string, {:^5d} integer, {:7.4f} a real".format("Fred", 358, 3.14159)
```

Category	Description
`Numbers`	• Decimal integer (d) • Fixed point (f)
`Alignment`	• Left (<) • Right (>) • Centred (^) • Repeat (*)
`Field size`	The total width of a field, regardless of how many columns are occupied

Math module

Import the math module.

The supported functionality is as follows:

Category	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 (∏)

Time module

Import the time module.

The supported functionality is as follows:

Function	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 module

Import the turtle module.

The supported functionality is as follows:

Function	Description
`turtle.forward(steps)`	Moves forward (facing direction) for number of steps
`turtle.back(steps)`	Moves backward (opposite facing direction) for number of steps
`turtle.left(degrees)`	Turns anticlockwise the number of degrees
`turtle.right(degrees)`	Turns clockwise the number of degrees
`turtle.home()`	Moves to canvas origin (0, 0)
`turtle.setpos(x, y)`	Positions the turtle at coordinates (x, y)
`turtle.reset()`	Clears the drawing canvas, sends the turtle home and resets variables to default values
`turtle.hideturtle()`	Makes the turtle invisible
`turtle.showturtle()`	Makes the turtle visible
`turtle.penup()`	Lifts the pen
`turtle.pendown()`	Puts the pen down
`turtle.pensize(width)`	Makes the pen the size of width (positive number)
`turtle.pencolor(color)`	Set the colour of the pen. The input argument can be a string or an RGB colour. For example: “red”, “#551A8B”, (0,35,102)