Pearson Edexcel International Advanced Level

Sample assessment material

Paper reference **WCP02/01**

Computer Science

Unit 2: Practical Programming and Problem-solving Programming Language Subset (PLS)

Resource Booklet

Do not return this Booklet with the question paper.

S89028A

1/1/1/1/1/1/1

S 8 9 0 2 8 A

Pearson

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. Learners familiar with everything in this document will be able to access all parts of the Unit 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 learner successfully using more esoteric or complex constructs or approaches not included in this document will still be awarded marks in Unit 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.

Programming

Flow control constructs

Sequence

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

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.

Match-case statement

`match <variable>:`
`  case <pattern 1>:`
`  <command>`
`  case <pattern 2>:`
`  <command>`
`  . . .`
`  case other: # Default`
`  <command>`
	If <variable> matches <pattern 1>, then first <command> is executed. If <variable> matches <pattern 2>, then the second <command> is executed.
	
	Supports multiple instances of case <pattern>:
	
	The last case is the default, when no previous pattern has been matched. The last case is optional. It can be written as case _:

Count-controlled iteration

`for <id> in <structure>:` `  <command>`	Executes <command> for each element of <structure>, in one dimension.

`for <id> in range (<start>,`
`  <stop>):`
`  <command>`
	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.

Condition-controlled iteration

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

Exception handling

`try:`
`  <command>`
`except:`
`  <command>`
	The <command> under the try, is executed. If an error occurs, the <command> under the except is executed. Where there is no error, the code following the try/except block is executed.

`try:`
`  <command>`
`except <error type>:`
`  <command>`
`except:`
`  <command>`
	The <command> under the try, is executed. If an error of <error type> occurs, the <command> under the except <error type> is executed. Otherwise, the code under the except is executed. Where there is no error, the code following the try/except block is executed.

Exception	Description
`ArithmeticError`	An error occurs in numeric calculations
`AttibuteError`	An attribute reference or assignment fails
`EOFError`	The input () method hits an end-of-file condition (EOF)
`Exception`	The base exception class to catch a wide range of errors, rather than a specific error.
`FileNotFoundError`	The requested file does not exist
`IndexError`	An index of a sequence does not exist
`KeyError`	A key does not exist
`NameError`	A variable does not exist
`TypeError`	An attempt to combine two different types
`UnboundLocalError`	A local variable is referenced before assignment
`ValueError`	A wrong value in a specified data type
`ZeroDivisionError`	The second operator in a division is zero

Variables and constants

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

Both uppercase and lowercase are supported.

Constants are conventionally named in all uppercase characters.

Assignment is used to set or change the value of an identifier.

`<identifier> = <value>`	Sets the <identifier> to the <value>
`<identifier> = <expression>`	Sets the <identifier> to the result of <expression>

Operators

Arithmetic

Arithmetic operator	Description
/	Division
*	Multiplication
**	Exponentiation
+	Addition
-	Subtraction
//	Integer division
%	Modulus

Relational

Relational operator	Description
==	Equal to
!=	Not equal to
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to

Boolean

Boolean operator	Description
`not`	Inverts the argument.
`and`	The test on both sides of the operator must be true to return true.
`or`	The test on one side of the operator must be true to return true.

Bitwise

Bitwise operator	Name	Description
&	AND	For each pair of bits in two arguments, it returns a one only when both bits are one.
|	OR	For each pair of bits in two arguments, it returns a one where either one or both bits are one.
^	XOR	For each pair of bits in two arguments, it returns a one where either one but not both bits are one.
~	NOT	Inverts
<<	Logical shift left	Moves the bits left by the given number of positions, filling with zeros from the right.
>>	Arithmetic shift right	Moves the bits right by the given number of positions, filling with the most-significant bit on the left.

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.

Organising and handling data

Data types

Identifiers may be explicitly assigned a data type during declaration.

Identifiers may be implicitly assigned a data type during initialisation.

Generic data type	Implemented data type	Conversion
Integer	int	int()
Real	float	float()
Character	str	str()
Boolean	bool	bool()
String	str	str()

Type conversion

`bool (<item>)`	Returns <item> converted to the equivalent Boolean value.
`float (<item>)`	Returns <item> converted to the equivalent real value.
`int (<item>)`	Returns <item> converted to the equivalent integer value.
`str (<item>)`	Returns <item> converted to the equivalent string value.
`type (<obj>)`	Returns the type of <obj>.

Data structures

A data structure is a sequence of items, which themselves are typed.

Where a data structure supports indexing, indexes start with zero.

Structure	Description	Python data type
Array	A sequence of items with the same (homogeneous) data type. Ordered. Mutable. Duplicates allowed.	list
Dictionary	Data is stored in key:value pairs. Ordered. Mutable. Duplicate keys not allowed.	dict
List	A sequence of items with the same (homogeneous) or mixed (heterogenous) data types. Ordered. Mutable. Duplicates allowed.	list
Record	A sequence of items usually of mixed (heterogenous) data types.	list
Set	A sequence of items with the same (homogeneous) or mixed (heterogenous) data types. Unordered. Mutable. Duplicates not allowed.	set
String	A sequence of characters. Ordered. Immutable. Duplicates allowed.	str
Tuple	A sequence of items with the same (homogeneous) or mixed (heterogenous) data types. Ordered. Immutable. Duplicates allowed.	tuple

Dimensions

`<struct>[<dim_one>]`	Accesses an item in a one-dimensional structure.
`<struct>[<dim_one>][<dim_two>]`	Accesses an item in a two-dimensional structure.

Ragged data structures are not supported. therefore, in a list of records, each record will have the same number of fields.

Abstract data types

An abstract data type is a complex data type, implemented using a simpler data structure. For example, a stack is an abstract data type. It can be implemented as a list.

List

Lists can be:

- indexed using []
- concatenated using the + operator
- sliced using the : operator.

`<aList> = list ()` `<aList> = []`	Two methods of creating a list structure. Both are empty.
`<list>.append (<item>)`	Adds <item> to the end of <list>.
`del <list> [<index>]`	Removes the item at <index> from <list>.
`<list>.insert (<index>, <item>)`	Inserts <item> just before an existing one at <index>.
`<list>.pop (<index>)`	Removes and returns the item at <index> from <list>. <index> is optional. The default is -1, which returns the last item appended to <list>.

Dictionary

`<aDict> = dict (<key1> = <value1>,` `<key2> = <value2>)`	Creates a dictionary with two key:value pairs, using the dict () constructor.
`<aDict> = {<key1>: <value1>,` `<key2>: <value2>}`	Creates a dictionary with two key:value pairs.
`<aDict>[<new key>] = <new value>`	Causes a <new key>:<new value> pair to be added to <aDict>.
`<aDict>[<key>] = <new value>`	Changes the value associated with <key> in <aDict> to <new value>.
`<aDict>.clear ()`	Empties <aDict>.
`del <aDict>[<key>]`	Removes the <key> and its associated value from <aDict>.
`<aDict>.get (<key>)`	Returns the value associated with <key> in <aDict>.
`<aDict>.items ()`	Returns an object of tuples. Each tuple is a key:value pair. Use list () to convert the returned result to a list.

`<aDict>.keys ()`	Returns all the keys in <aDict>. Use list () to convert the returned result to a list.
`<aDict>.values ()`	Returns all the values in <aDict>. Use list () to convert the returned result to a list.

Tuple

Tuples can be:

- indexed using []
- concatenated using the + operator
- sliced using the : operator.

`<aTuple> = (<item>,)`	Creates a tuple of one item. An extra comma is needed. Otherwise, the single <item> will be treated as a different data type.
`<aTuple> = tuple ()`	Creates an empty tuple.

Set

`<aSet> = {<value1>, <value2>}`	Creates a set with two items. The items can be different data types.
`<aSet> = set ((<value1>, <value2>))`	Creates a set with the set () constructor. Two sets of round brackets are required. The items can be different data types.
`<aSet>.add (<value>)`	Adds <value> to the existing set <aSet>.
`<aSet>.clear ()`	Empties <aSet>.
`<set1>.difference (<set2>)`	Returns the set of items that exist only in <set1> but not in <set2>. This is the difference operation for sets.
`<aSet>.discard (<value>)`	Removes <value> from <aSet>. If <value> is not in the set, an error will not occur.
`<value> in <aSet>`	Returns true, if the <value> is in <aSet>. Otherwise, returns false. This is the membership operation for sets.
`set.intersection (<set1>, <set2>)`	Returns a new set with all the items that occur in both sets. This is the intersection operation for sets.
`<aSet>.remove (<value>)`	Removes <value> from <aSet>. If <value> is not in the set, an error will occur.
`<set1>.symmetric_difference (<set2>)`	Returns a new set with all items that are in <set1> or in <set2>, but not in both sets. This is the exclusive or operation for two sets.
`<set1>.union (set2)`	Returns a new set with all the items from both sets. This is the union operation for sets.

Set operator	Description
-	Difference
&	Intersection
^	Symmetric difference (Exclusive or)
|	Union

Numeric data

`chr (<integer>)`	Returns the string which matches the Unicode value of <integer>. The first 128 characters of Unicode are equivalent to ASCII.
`int (<item>)`	Returns <item> converted to the equivalent integer value. This removes the fractional part of <item>.
`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.
`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>.

String data

Strings can be:

- indexed using []
- concatenated using the + operator
- sliced using the : operator
- repeated using the * operator.

`<string>.capitalize ()`	Returns a new string, equivalent to <string> where the first letter is capitalised.
`<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. <string>. Returns -1, if not found.
`<string>.format (<values>)`	Formats <values> and puts them into <string>. The content of <string> is described by symbols and placeholders.
`<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>.isalnum ()`	Returns true, if all characters are alphabetic A–Z, a-z or digits 0–9.
`<string>.isalpha ()`	Returns true, if all characters are alphabetic A–Z or a-z.
`<string>.isdigit ()`	Returns true, if all characters are digits 0–9, exponents are digits.
`<string>.islower ()`	Returns true, if all characters in <string> are lowercase.
`<string>.isnumeric ()`	Returns true if all the characters in <string> are the values 0 to 9.
`<string>.isspace ()`	Returns true if all the characters in <string> are whitespaces.
`<string>.isupper ()`	Returns True, if all characters in <string> are uppercase.
`len (<string>)`	Returns the length of <string>.
`<string>.lower ()`	Returns original <string> in lowercase.

`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.
`<string>.strip (<char>)`	Returns original <string> with all occurrences of <char> removed from the front and back.
`<string>.split (<char>)`	Returns a list of all substrings in original <string>, using <char> as the separator.
`<string>.title ()`	Returns a string where the first character of each word is converted to upper case.
`<string>.upper ()`	Returns original <string> in uppercase.

Formatted string literals (f-strings)

The value of expressions can be included inside a string, where the string has been prefixed with the character f or the character F. The expressions are supplied inside curly braces, { }. An optional format specifier may follow each expression, separated from the expression by a semi-colon.

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

Placeholder	Option	Description
align	<	Left aligned. Default for most items, such as 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.

Date and time

Dates and times follow the ISO 8601 standard, designed to eliminate ambiguity and support worldwide exchange of data. The ordering, field lengths and separators are fixed.

String	ISO 8601 Format
`2024-09-27`	YYYY-MM-DD
`14:30:00`	HH:MM:SS
`24-09-27 14:30:00`	YYYY-MM-DD HH:MM:SS

All dates and times should be processed as Python datetime objects. No localisation of dates and times is required for processing or presentation.

The Python format string to describe the presentation of ISO 8601 format is: "%Y-%m-%d %H:%M:%S".

	Storing and processing dates and times
datetime library	Library that contains subprograms to manipulate a datetime object that represents a Coordinated Universal Time (UTC).
Text files	Store as a text string in ISO 8601 format.
SQLite3 library	Store as a text string in ISO 8601 format.

Text files

Where lines in the file contain more than a single column of data, a comma is used as a delimiter.

`<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 (<struct>)`	Writes <structure> to a file. <struct> is a list of strings.
`<fileid>.write (<aString>)`	Writes a single string to a file.
`<fileid>.close ()`	Closes the file.

Control characters

Name	Abbreviation	ASCII hexadecimal	String
Carriage return	CR	0x0D	`"\r"`
Line feed	LF	0x0A	`"n"`
Horizontal tab	HT	0x09	`"\t"`

Relational databases

The SQLite3 library module is used to support relational database operations.

Communication

`<connection>.close ()`	Closes the database connection, when no longer required.
`<cursor>.commit ()`	Ensures any outstanding operations are immediately committed against the database <cursor>.
`sqlite3.connect (<database>)`	Returns a connector handle to <database>.
`<connection>.cursor ()`	Returns a cursor to the database <connection>.
`<cursor>.execute (<expression>)`	Executes <expression> against the database <cursor>.
`<cursor>.fetchall ()`	Retrieves all the rows in the result of the last execution call on <cursor>.
`<cursor>.fetchone ()`	Retrieves the next row in the result of the last execution call on <cursor>.

Tables

`CREATE TABLE <table> (`
`  <name1 type constraint>,`
`  <name2 type constraint>,`
`  . . .);`
	Creates <table> with columns of the indicated names to hold values of <type>. The constraint is optional.

`DROP TABLE <table>;`	Deletes an existing table. All the table contents are destroyed.

Constraint keywords

`DEFAULT`	Assigns a default value for a column where no value is supplied.
`FOREIGN KEY`	Ensures that links between tables are not destroyed.
`NOT NULL`	Null values not allowed.
`PRIMARY KEY`	Uniquely identifies each row. Combination of NOT NULL and UNIQUE.
`UNIQUE`	No duplicates are allowed.

Type keywords

BLOB	The value is a blob of data, stored exactly as it was input.
INTEGER	The value is a signed integer, stored in 0, 1, 2, 3, 4, 6, or 8 bytes depending on the magnitude of the value.
NULL	The value is the NULL value.
REAL	The value is a floating- point.
TEXT	The value is a text string, stored using the database encoding. Use this type for storing datetimes in ISO 8601 format.

Manipulating records

DELETE FROM <table>;	Deletes all the records in <table>. The table structure is not destroyed.
DELETE FROM <table>; WHERE <criteria>;	Deletes all the records in <table> that match the <criteria>.
INSERT INTO <table> (<columns>) VALUES (<values>);	Inserts a record into <table>. The column names are given by <columns>. The values to put into the <columns> are given by <values>. The ordering must match.
TRUNCATE TABLE <table>;	Deletes all the data in <table>, but does not delete the <table> structure.
UPDATE <table> SET <field1> = <value1>, <field2> = <value2>, . . . WHERE <criteria>	Modifies the existing records in an existing table.

Retrieving data

`SELECT <columns>`
`FROM <table>`
`WHERE <criteria>;`
	Returns a set of records containing <columns> from the existing <table>. The WHERE clause is optional. For all the columns, use *.

`SELECT DISTINCT <columns>`
`FROM <table>`
`WHERE <criteria>;`
	Returns a set of records containing <columns> from the existing <table>. The returned values are unique across columns. The WHERE clause is optional.

Filtering and refining data

`WHERE <criteria>`	Returns only those records that meet <criteria>.
`<expr1> AND <expr2>`	Includes only those records where both <expr1> and <expr2> are true.
`BETWEEN <begin> AND <end>`	Used to select values in a range. <begin> and <end> values are included.
`DISTINCT`	Used to ensure there are no duplicate rows in the result.
`LIKE`	Used to find records that match a pattern. % indicates zero, one, or more characters. _ indicates a single character.
`IS NOT NULL`	Used to test for non-empty values.
`IS NULL`	Used to test for empty values.
`NOT <condition>`	Used to return only records where <condition> is not true.
`<expr1> OR <expr2>`	Includes only those records where either or both expressions are true.

`=`
`<>`
`>`
`>=`
`<`
`<=`
	Relational operators that may be used in creating <criteria>, <expression>, or <condition>.

`<expr1> UNION <expr2>`	Returns the combined results from two SELECT statements.

Presentation

`ASC`	Sorts returned records in ascending order.
`DESC`	Sorts returned records in descending order.
`GROUP BY <column>`	Used with aggregate functions to group the returned records.
`ORDER BY <column>`	Specifics the <column> on which to base the ordering.

Aggregation

`HAVING <criteria>`	May be used instead of WHERE for aggregate functions. The <criteria> is made up of an aggregate function with a condition.
`AVG (<column>)`	Returns the mean of <column> where <column> holds numbers.
`COUNT (<column>)`	Returns the number of rows that match the criteria given in the WHERE clause. Null values will not be counted. Use * for all rows in a table. Null values will be counted.
`MAX (<column>)`	Returns the largest value in a <column>
`MIN (<column>)`	Returns the smallest value in a <column>
`SUM (<column>)`	Returns the sum of <column> where <column> holds numbers.

Joins

Joins are used to combine rows from two or more tables based on a related column. The joins require SELECT to identify the intended columns and FROM to identify the intended tables. The <criteria> is expressed in terms of a relationship between the <left> table and the <right> table.

`FULL OUTER JOIN <right> ON <criteria>` `FULL JOIN <right> ON <criteria>`	Returns all the records where there is a match in either the left or right table. This is all the records in <left>, all the records in <right>, and the intersection.
`INNER JOIN <column> ON <criteria>` `JOIN <column> ON <criteria>`	Returns all the records that have matching values in both tables. This is the intersection operation.
`LEFT OUTER JOIN <left> ON <criteria>` `LEFT JOIN <left> ON <criteria>`	Returns all the records from the left table along with the matched records from the right table. This result is the original <left> and the intersection with <right>.
`RIGHT OUTER JOIN <right> ON <criteria>` `RIGHT JOIN <left> ON <criteria>`	Returns all the records from the right table and the matched records from the left table. This result is the original <right> and the intersection with <left>.

Errors

`sqlite3.Error`	Base class for all SQLite errors. If the exception is from the SQLite library, two additional attributes are added: • sqlite_errorcode – the numeric error code • sqlite_errorname – the symbolic name of the numeric error code

Name	Code	Description
`SQLITE_OK`	0	Successful result
`SQLITE_ERROR`	1	Generic error
`SQLITE_INTERNAL`	2	Internal logic error in SQLite
`SQLITE_PERM`	3	Access permission denied
`SQLITE_ABORT`	4	Call-back routine requested an abort
`SQLITE_BUSY`	5	The database file is locked

SQLITE_LOCKED	6	A table in the database is locked
SQLITE_NOMEM	7	A memory allocation failed
SQLITE_READONLY	8	Attempt to write a readonly database
SQLITE_INTERRUPT	9	Operation terminated by sqlite3_interrupt()
SQLITE_IOERR	10	Some kind of disk I/O error occurred
SQLITE_CORRUPT	11	The database disk image is malformed
SQLITE_NOTFOUND	12	Unknown opcode in sqlite3_file_control()
SQLITE_FULL	13	Insertion failed because database is full
SQLITE_CANTOPEN	14	Unable to open the database file
SQLITE_PROTOCOL	15	Database lock protocol error
SQLITE_EMPTY	16	Internal use only
SQLITE_SCHEMA	17	The database schema changed
SQLITE_TOOBIG	18	String or BLOB exceeds size limit
SQLITE_CONSTRAINT	19	Abort due to constraint violation
SQLITE_MISMATCH	20	Data type mismatch
SQLITE_MISUSE	21	Library used incorrectly
SQLITE_NOLFS	22	Uses OS features not supported on host
SQLITE_AUTH	23	Authorization denied
SQLITE_FORMAT	24	Not used
SQLITE_RANGE	25	2nd parameter to sqlite3_bind out of range
SQLITE_NOTADB	26	File opened that is not a database file
SQLITE_NOTICE	27	Notifications from sqlite3_log()
SQLITE_WARNING	28	Warnings from sqlite3_log()
SQLITE_ROW	100	sqlite3_step() has another row ready
SQLITE_DONE	101	sqlite3_step() has finished executing

Best practice

Global keyword

The PLS supports considered and minimal use of the global keyword.

Blocked code

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

Built-in subprograms

`input (<prompt>)`	Displays <prompt> on the screen and returns the line typed in.
`len (<object>)`	Returns the length of the <object>
`print (<item>)`	Displays <item> on the screen.

Libraries of subprograms

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

`import <library>`	Imports the whole of the <library> module into the current program.
`import <library> as <alias>`	Imports the whole of the <library> which can be addressed as <alias>.
`from <library>` `  import <identifier>`	Imports only <identifier> from the <library> module.

`from <library>`
`  import <identifier>`
`  as <alias>`
	Imports only <identifier> from the <library> module which can be addressed <alias>.

Random

`random.randint (<a>, <b>)`	Returns a random integer X so that <a> <= X <= <b>.
`random.random ()`	Returns a floating-point number in the range of 0.0 to 1.0
`random.shuffle (<list>)`	Randomises the contents of <list>, in place.

Math

`math.ceil (<x>)`	Returns the smallest integer greater than or equal to <x>.
`math.fabs (<x>)`	Returns the absolute value of <x>. Equivalent to |<x>|.
`math.floor (<x>)`	Returns the largest integer less than or equal to <x>.
`math.pi`	The constant Pi (Π).
`math.pow (<x>, <y>)`	Returns the value of <x> raised to the power of <y>.
`math.sqrt (<x>)`	Returns the square root of <x>.
`math.trunc (<x>)`	Returns the integer part of <x>.

Copy

`copy.copy (<obj>)`	Returns a shallow copy of <obj>.
`copy.deepcopy (<obj>)`	Returns a deep copy of <obj>.

Time

Although there are many different ways of storing and manipulating date and time values, the PLS uses the number of seconds since January 1, 1970, 00:00:00 Coordinated Universal Time (UTC). This base time is known as the epoch. This format is time-zone independent. Individual time zones or local time can be ignored.

Datetime subprograms

`datetime.fromtimestamp ( <stamp> timezone.utc)`	Returns a Coordinated Universal Time (UTC) datetime object equivalent to <stamp>, an integer. The timezone.utc argument enforces the format.
`datetime.now (timezone.utc)`	Returns a Coordinated Universal Time (UTC) datetime object equivalent to the current local time. The timezone.utc argument enforces the format.
`datetime.strptime ( <string>, <format>)`	Returns a datetime object equivalent to a <string> described by <format>. Set <format> to "%Y-%m-%d %H:%M:%S" for ISO 8601 compatibility.

Datetime object

`<obj>.date ()`	Returns only the date portion of <obj>.
`<obj>.replace (tzinfo-timezone.utc)`	Returns a datetime object with the timezone adjusted for a Coordinated Universal Time (UTC).
`<obj>strftime ('%H:%M:%S')`	Returns only the time portion of <obj>, formatted for hours, minutes and seconds.
`<obj>.timestamp ()`	Returns a timestamp, which is a real number, representing the date and time in <obj>.

Time

`time.sleep (<sec>)`	The current process is suspended for the given number of seconds, then resumes at the next line of the program.
`time.time ()`	Returns a floating-point number which is the current time in seconds since the epoch.

Turtle Graphics

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)
```

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.

Window and drawing canvas

`<window>.setup (<width>, <height>)`	Sets the size of the drawing canvas to <width>, <height> in pixels. Requires use of turtle.Screen () to create <screen> 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 counter clockwise. A turtle in logo mode, initially points up (north) and angles are clockwise.
`turtle.Screen ()`	Returns a variable to address the turtle indow. Use with <window>.setup ().

Creation, visibility and movement

`<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.

Positioning and direction

`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>).

Filling shapes

`<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.

Controlling the pen

`<turtle>.pencolor ( <colour>)`	Sets the colour of the pen. The input argument is a string or an RGB colour.
`<turtle>.pendown ()`	Puts the pen down.
`<turtle>.pensize (<width>)`	Makes the pen the size of <width> (positive number).
`<turtle>.penup ()`	Lifts the pen up.

Circles

`<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.

Colour strings

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

Clean code

Line continuations

Method	Characters	Comment
Inside matched delimiters	Delimiters are: • Round brackets ( ) • Square brackets [ ] • Curly braces { }	Very easy to read. Adding additional delimiters to allow line breaking at sensible locations is good practice.
Continuation character	The continuation character is: • \	Difficult to read

Comments

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