Pearson Edexcel International Advanced Level

Sample assessment material

Paper reference **WCP04/01**

Computer Science

Unit 4: Advanced Practical Programming and Problem-solving
Programming Language Subset (PLS)

Resource Booklet

Do not return this Booklet with the question paper.

S89029A

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

S 8 9 0 2 9 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 4 assessment. This does not stop a teacher/learner 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 4, 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>.
`<list>.clear ()`	Removes all items from <list>.
`<list>.copy ()`	Returns a new object with the same contents as <list>.
`<list>.count (<value>)`	Returns the number of items with <value>.
`del <list> [<index>]`	Removes the item at <index> from <list>.
`<list>.index (<value>)`	Returns the index of the first <value>.
`<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>.
`<list>.reverse ()`	Reverses the contents of <list>.

<table>
<tr><td><list>.sort (<option>)</td><td>Sorts <list> in place.
<table><tr><th>Option</th><th>Effect</th></tr><tr><td>blank</td><td>All defaults, i.e. ascending and case sensitive.</td></tr><tr><td>reverse = True</td><td>Sorts in descending order.</td></tr><tr><td>key = <afunction></td><td>Sorts <object> based on the returned value from <afunction>, which may be a lambda function.</td></tr><tr><td>key = str.lower()</td><td>Makes the sort case insensitive.</td></tr></table></td></tr>
<tr><td><list>.remove (<value>)</td><td>Removes the first item with a value of <value>.</td></tr>
</table>

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>.

Decimal library

The decimal library module is used for high-precision floating-point numbers. Using the decimal module prevents the precision loss that can happen when using floating-point numbers.

`decimal.Decimal (<value>)`	Creates a decimal object with the value of <value>.
`decimal.getcontext()`	Returns the current context for the decimal library module.
`decimal.getcontext ().prec = <value>`	Sets the output precision <value> for the decimal module.
`<object>.sqrt ()`	Returns the square root of the decimal <object>.
`decimal.compare (<x>, <y>)`	Returns -1, when <x> is less than <y>. Returns 1, when <x> is greater than <y>. Returns 0, when <x> is equal to <y>.
`<x>.copy_abs ()`	Returns the absolute value of the decimal number <x>.
`<x>.min (<y>)`	Returns the minimum of <x> and <y>.
`<x>.max (<y>)`	Returns the maximum of <x> and <y>.
`<x>.logical_and (<y>)`	Returns the result of a logical AND on the two decimal numbers <x> and <y>.
`<x>.logical_or (<y>)`	Returns the result of a logical OR on the two decimal numbers <x> and <y>.
`<x>.logical_xor (<y>)`	Returns the result of a logical XOR on the two decimal numbers <x> and <y>.

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 Coordinated Universal Time (UTC).
Text files	Store as a text string in ISO 8601 format.
pandas library	Loading CSV files • Read CSV data into a dataframe, using pandas.read_csv () • Convert date and time columns to datetime object, using pandas.to_datetime (<df_column>) Saving CSV files • Convert datetime object columns to string format, using <df_column>.dt.strftime () with an argument of = "%Y-%m-%d %H:%M:%S" • Write the dataframe to CSV using <dataframe>.to_csv ()

RegEx

<table>
<tr><td>re.findall (<target>, <text>)</td><td>Returns a list holding all found matches for <target> in <text>.</td></tr>
<tr><td>re.search (<target>, <text>)</td><td>Returns a Match object if there is a match anywhere in the string.
<table><tr><th>Match attribute</th><th>Description</th></tr><tr><td>.start</td><td>Index in <text> for the first letter of <target>.</td></tr><tr><td>.end</td><td>Index in <text> one past the last letter of <target>.</td></tr></table></td></tr>
<tr><td>re.split (<chars>, <text>)</td><td>Returns a list where <string> is split at each match of <chars>.</td></tr>
<tr><td>re.sub (<target>, <chars>, <text>)</td><td>Replaces one or many matches of <target> in <text> with <chars>.</td></tr>
</table>

Metacharacters

Character	Description
[ ]	Indicates a set of characters.
\	Indicates a special sequence. Also used as for escape characters.
.	Matches any character, except a newline character.
^	Matches at the beginning of the line, as in starts with. Used as a NOT operator.
$	Matches at the end of the line, as ends with.
*	Zero or more repetitions of the preceding regular expression.
+	One or more repetitions of the preceding regular expression.
?	Zero or one repetitions of the preceding regular expression.
{<options>}	{<m>} – match exactly <m> copies of the preceding regular expression. {<m>, <n>} – match from <m> to <n> copies of the preceding regular expression.
|	The OR operator, as in either one or the other.
( )	Referred to as a capture group that allows extraction of specific parts of a match.

Character sets

Character set	Description
[<char>…<char>]	Returns a match for any individual <char> in the list.
[<start>-<end>]	Returns a match for any character between <start> and <end>.
[^<char>…<char>]	Returns a match for any individual <char> not in the list.
[<s1>-<s2>] [<s3>-<s4>]	Returns a match for any two-character string where the first character is from the range defined by <s1> and <s2> and the second character is from the range defined by <s3> and <s4>.
[a-zA-Z]	Returns a match for any character from a to z or from A to Z.
[+]	Returns a match for any + symbol in the text.

Special sequences

Character	Description
\A	Matches only if the specified characters are at the beginning of the text.
\b	Matches where the characters are at the beginning or end of the text. Precede the target with r, as in r"<target>".
\B	Matches where the target is found in the text, but NOT at the beginning or at the end of the text. Precede the target with r, as in r"<target>".
\d	Matches any decimal digit, [0-9].
\D	Matches any non-digit character, ^[0-9].
\s	Matches any whitespace character, [\t\n\r\f\v].
\S	Matches any non-whitespace character, [^ \t\n\r\f\v].
\w	Matches any alphanumeric character, [a-zA-z0-9_].
\W	Matches any non-alphanumeric character, [^a-zA-z0-9_].
\Z	Matches if the target is at the end of the text.

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"

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

<table>
<tr><td><code>input (<prompt>)</code></td><td>Displays <prompt> on the screen and returns the line typed in.</td></tr>
<tr><td><code>len (<object>)</code></td><td>Returns the length of the <object></td></tr>
<tr><td><code>print (<item>)</code></td><td>Displays <item> on the screen.</td></tr>
<tr><td><code>sorted (<object>,
<option>)</code></td><td>Returns a sorted list of items in <object>. The original <object> is not changed.

<table>
<tr><th>Option</th><th>Effect</th></tr>
<tr><td><code>blank</code></td><td>All defaults, i.e. ascending and case sensitive.</td></tr>
<tr><td><code>reverse = True</code></td><td>Sorts in descending order.</td></tr>
<tr><td><code>key = <afunction></code></td><td>Sorts <object> based on the returned value from <afunction>, which may be a lambda function.</td></tr>
<tr><td><code>key = str.lower()</code></td><td>Makes the sort case insensitive.</td></tr>
</table>
</td></tr>
</table>

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.
`random.uniform (<a>, <b>)`	Returns a random floating-point number X so that <a> <= X < <b>

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>.

NumPy

`<arr>[<dim1>][<dim2>] …`	Accesses the item in <arr> found at index values given by <dim1> and <dim2>. Index may be negative.
`<arr>[<start>:<stop>:<step>]`	Array slicing. <start> is the beginning, up to but not including <stop>, by <step> interval.
`<arr>[<filter>]`	Array filtering. Selects only those items in <arr> where the corresponding value in <filter> is true.

`numpy.array (<obj>, <type>)`	Returns a numpy ndarray made from items in <obj>. <type> is optional and may have these values: **<type>** / **Type** `'i'` / int `'b'` / bool `'f'` / float `'M'` / Datetime `'S'` / str
`numpy.array ([<filter>])`	Returns a ndarray containing only the items where the corresponding item in <filter> is true or the expression <filter> evaluates to true.

<table>
<tr><td><arr1> <arith> <arr2></td><td>Returns a ndarray with items indicating the evaluation of <arr1> <arith> <arr2>.

<table><tr><th><arith></th><th>Value</th></tr><tr><td>+</td><td>Addition</td></tr><tr><td>-</td><td>Subtraction</td></tr><tr><td>*</td><td>Multiplication</td></tr><tr><td>/</td><td>Division</td></tr></table></td></tr>
<tr><td><arr>.astype (<type>)</td><td>Returns a new array based on the items in <arr> with the data types converted to <type>.</td></tr>
<tr><td>numpy.average (<arr1>,
weights = <values>)</td><td>Returns the arithmetic mean of all items in <arr1>, when weights is omitted. Where <values> is an ndarray of weights, the weight is applied to each item <arr1> during the calculation.</td></tr>
<tr><td>numpy.column_stack (<arr1>, …)</td><td>Stacks a sequence of one or more one-dimensional arrays as columns into a two-dimensional array. All of the arrays must have the same first dimension.</td></tr>
<tr><td>numpy.loadtxt (<file>,
<arg1>, …)</td><td>Returns a ndarray constructed from the data in <file>. Additional arguments are optional and may have these values:

<table><tr><th><arg></th><th>Description</th></tr><tr><td>delimiter</td><td>The character used to separate the values in the file. The default is a space.</td></tr><tr><td>dtype</td><td>The data type of the resulting ndarray. The default is float.</td></tr><tr><td>skiprows</td><td>Skips the number of rows specified from the beginning of the file. The default is 0.</td></tr></table></td></tr>
<tr><td>numpy.mean (<arr1>,
axis = <num>)</td><td>Returns the arithmetic mean of all items in <arr1>, along axis number <num>. Axis is optional. The default for axis is 0.</td></tr>
<tr><td>numpy.mod (<arr1>, <arr2>)

numpy.remainder (<arr1>,
<arr2>)</td><td>Returns a ndarray with the remainders of all <arr1> divided by corresponding item in <arr2>.</td></tr>
</table>

<table>
<tr><td>numpy.ndmin (<arr>)</td><td>Returns the rank (number of dimensions) in <arr>.</td></tr>
<tr><td>numpy.power (<arr1>,
<arr2>)</td><td>Returns a ndarray with each item in <arr1> raised to the power of the corresponding item in <arr2>.</td></tr>
<tr><td>numpy.ravel (<arr>)</td><td>Returns a flattened ndarray of all the items in <arr>. *<arr> is a shortcut for ravel.</td></tr>
<tr><td><filter> = <arr1>
<relation>
<arr2></td><td>Returns a Boolean ndarray with items indicating the evaluation of <arr1> <relation> <arr2>.
<table><tr><th><relation></th><th>Description</th></tr><tr><td>></td><td>Greater than</td></tr><tr><td>>=</td><td>Greater than or equal to</td></tr><tr><td>==</td><td>Equal to</td></tr><tr><td>!=</td><td>Not equal to</td></tr><tr><td><=</td><td>Less than or equal to</td></tr><tr><td><</td><td>Less than</td></tr></table></td></tr>
<tr><td>numpy.reshape (<arr>,
<shape>)</td><td>Returns the contents of <arr> reshaped to the dimensions in <shape>. <arr> is an integer or tuple of integers. The original <arr> and the new <shape> must be compatible. Where <shape> is an integer, that number of elements of <arr> is returned. Where <shape> is -1, the result is all the elements of <arr>.</td></tr>
</table>

<table>
<tr><td>numpy.savetxt (<file>,
<arr>,
<arg1>, …)</td><td>Saves <arr> to <file>. Additional arguments are optional and may have these values:
<table>
<tr><th><arg></th><th>Description</th></tr>
<tr><td>delimiter</td><td>The character used to separate the values in the file. The default is a space.</td></tr>
<tr><td>fmt</td><td>A single string or multiple strings in a list that specify a format for the data in <arr>. Common strings are:
<table>
<tr><th>String</th><th>Type</th></tr>
<tr><td>"%d"</td><td>Integer</td></tr>
<tr><td>"%.2f"</td><td>Float to two decimal places</td></tr>
<tr><td>"%s"</td><td>String</td></tr>
</table></td></tr>
<tr><td>header</td><td>A string that will be written at the top of <file>.</td></tr>
</table></td></tr>
<tr><td>numpy.shape (<arr>)</td><td>Returns a tuple with the same number of items as dimensions in <arr>, with each value indicating the number of items in that dimension.</td></tr>
<tr><td>numpy.size (<arr>, <axis>)</td><td>Returns the number of elements along the given <axis>. Where <axis> is omitted, the total number of elements is returned.</td></tr>
<tr><td><arr>.sort ()</td><td>Sorts the items in <arr> in place. This destroys the original <arr>.</td></tr>
<tr><td>numpy.sort (<arr>)</td><td>Returns an ndarray of the sorted values in <arr>. This does not change the original <arr>.</td></tr>
</table>

<table>
<tr><td>numpy.sort (<arr>,
axis = <int>)</td><td>Returns an ndarray of <arr>, sorted on the indicated axis.

<table><tr><th><int></th><th>Description</th></tr><tr><td>0</td><td>Along the first axis.</td></tr><tr><td>1</td><td>Along the second axis.</td></tr><tr><td>-1</td><td>Along the last axis.</td></tr><tr><td>None</td><td>Along no axis, which results in a flattened <arr>.</td></tr></table></td></tr>
<tr><td>numpy.std (<arr1>)</td><td>Returns a measure of standard deviation. This is the average distance of the values in <arr1> from the mean of <arr1>, expressed in the same units as the original values.</td></tr>
<tr><td>numpy.sum (<arr1>,
axis = <num>)</td><td>Returns a single value for the sum of all items in <arr1>, along axis number <num>. Axis is optional. The default for axis is 0.</td></tr>
<tr><td><arr>.tolist ()</td><td>Converts <arr> to a list.</td></tr>
<tr><td>numpy.transpose (<arr1>)</td><td>Returns an ndarray, with the same values as <arr1>, where the columns and rows are swapped.</td></tr>
<tr><td>numpy.var (<arr1>)</td><td>Returns a measure of variance. This is how much the group of values in <arr1> differs from the mean of the group. This indicates the spread of the values.</td></tr>
<tr><td>numpy.vstack (<arr1>, …)</td><td>Stacks a sequence of one or more one-dimensional arrays as rows into a two-dimensional array. All of the arrays must have the same first dimension.</td></tr>
<tr><td>numpy.where (<arr>,
<condition>)</td><td>Returns a ndarray of indexes, in <arr>, where <condition> is true.</td></tr>
<tr><td>numpy.where (<arr>,
<condition>,
<true>,
<other>)</td><td>Returns a ndarray of <true> from <arr> here <condition> is true, otherwise returns <other>.</td></tr>
</table>

Pandas

Dataframes can be thought of as a two-dimensional table with rows and columns.

Series can be thought of as a one-dimensional array representing a single column.

Indexes can be thought of as the row labels. Dataframes without explicit row labels are addressed by indexes, starting at zero.

Dataframes

`DataFrame (<obj>)`	Returns a pandas dataframe made from items in <obj>. Where <obj> is a dictionary, the keys become the column names.

`DataFrame (<obj>,`
`  columns = <names>,`
`  index = <labels>)`
	Returns a pandas dataframe made from items in <obj>, with column headers given in <names> and row identifiers given in <labels>. Both columns and index are optional.

<table>
<tr>
<td>

```
pandas.read_csv (<file>,
    <arg1>, …)
```

</td>
<td>
Returns the contents of <file> into a dataframe. Additional arguments are optional and may have these values:
<table>
<tr><th><arg></th><th>Description</th></tr>
<tr><td>delimiter</td><td>The character used to separate the values in each line of the file. The default is a comma.</td></tr>
<tr><td>dtype</td><td>The data type of the resulting dataframe or individual columns in the dataframe. The specifier may be a dictionary, where key is the column name and value is the data type.


```
{'col1': str,
    'col2': float,
    'col3': int}
```


Where this argument is not supplied, Pandas infers the type from the values in each column.</td></tr>
<tr><td>header</td><td>Specifies which row in the file will be used as the header for the columns. The default is line 0, which means the column names are inferred from the first line of the file.</td></tr>
<tr><td>skiprows</td><td>Skips the number of lines specified from the beginning of the file. The default is none.</td></tr>
<tr><td colspan="2">

```
parse_dates = [<col_name>]
dayFirst = True
```


Use to read European date and times from file into the dataframe. <col_name> is the header for the column that contains the date time string.</td></tr>
</table>
</td>
</tr>
<tr>
<td><df>[<column>]</td>
<td>Returns a series of the data in <df> in the column named <column>. See the following section for information about series.</td>
</tr>
<tr>
<td><df>[<column>] = <obj></td>
<td>Adds a column named <column> to <df> with the values in <obj>.</td>
</tr>
</table>

<table>
<tr><td><df>[<condition>]</td><td>Returns a dataframe with rows where <condition> is true.
<table><tr><th>Relation</th><th>Description</th></tr><tr><td>></td><td>Greater than</td></tr><tr><td>>=</td><td>Greater than or equal to</td></tr><tr><td>==</td><td>Equal to</td></tr><tr><td>!=</td><td>Not equal to</td></tr><tr><td><=</td><td>Less than or equal to</td></tr><tr><td><</td><td>Less than</td></tr></table></td></tr>
<tr><td><df>.describe ()</td><td>Returns statistical information about <df>.</td></tr>
<tr><td><df>.drop (<row>,
inplace = True)</td><td>Causes the <row> to be removed from <df>. The inplace key word cause the original <df> to be changed.</td></tr>
<tr><td><df>.drop_duplicates (
inplace = <bool>)</td><td>Removes all the duplicate rows from <df>. The inplace keyword, which may be omitted, if set to true, changes the original <df>.</td></tr>
<tr><td><df>.dropna ()</td><td>Returns a new dataframe where each row with a null value is left out.</td></tr>
<tr><td><df>.duplicated ()</td><td>Returns a series reporting true for every row that is a duplicate and false for every row that is not a duplicate.</td></tr>
<tr><td><df>.head (<value>)</td><td>Returns the first <value> rows in <df.
Where.<value> is omitted, the first five rows are returned. Column headers are also returned.</td></tr>
<tr><td><df>.iloc[<value>]</td><td>Returns the item at the location indicated by <value>. <value> is integer-based and can include:
A single integer
A list of integers
A slice of integers, where both the start and stop are included</td></tr>
<tr><td><df>.info ()</td><td>Returns information about <df>.</td></tr>
</table>

<table>
<tr><td><df>.loc[<value>]</td><td>Returns the item at the location indicated by <value>. <value> is label-based and can include:
A single label
A list of labels
A slice of integers, where the stop is not included</td></tr>
<tr><td><df>.loc[<index>, <column>]</td><td>Accesses value at <index>, <column> with a new <value>. Can be used to retrieve, change, or check a value.</td></tr>
<tr><td><df>.sort_values (
by = <column>,
ascending = <bool>)</td><td>Returns a dataframe sorted by <column>. Where <bool> is true, the order is ascending. This is the default when ascending is omitted. For descending order, use ascending = false.</td></tr>
<tr><td><df>.tail (<value>)</td><td>Returns the last <value> rows in the dataframe. Where <value> is omitted, the first five rows are returned. Column headers are also returned.</td></tr>
<tr><td><df>.to_csv (<file>,
<arg1>, …)</td><td>Saves <df> to a csv file with the name <file>. Additional arguments are optional and may have these values:<table><tr><th><arg></th><th>Description</th></tr><tr><td>sep</td><td>The character used to separate the values in the file. The default is a comma.</td></tr><tr><td>float_
format</td><td>Format string for floating point numbers, such as "%.2f", for two decimal places.</td></tr><tr><td>index</td><td>Writes row indexes to the file. The default is true.</td></tr><tr><td>header</td><td>Writes column headers to the file. The default is true.</td></tr><tr><td>date_format</td><td>Format string for datetime objects, such as "%d/%m/%Y %H:%M:%S"</td></tr></table></td></tr>
<tr><td><df>.to_datetime (<data>)</td><td>Converts all the data in <data> to a datetime format. <data> is a column specified by <df>[<column>].</td></tr>
<tr><td><df>.to_string ()</td><td>Returns the contents of <df> converted to strings, suitable for printing. Where the <df> is large, only the first five and the last five rows are returned.</td></tr>
</table>

Series

`pandas.Series (<obj>, index = <labels>)`	Returns a pandas Series from items in <obj>. Where the index keyword is provided, <labels> represent the unique identifier for each row. The length of <labels> must match the length of <obj>. Where the index keyword is omitted, each row is assigned an integer value, beginning at position 0.
`Series (<obj>, index = <labels>)`	Returns a pandas Series from items in <obj>. Where the index keyword is provided, <labels> represent the unique identifier for each row. The length of <labels> must match the length of <obj>. Where the index keyword is omitted, each row is assigned an integer value, beginning at position 0.
`<series>.apply (<func>)`	Applies <func> to each value.
`<series>.count ()`	Returns the count of non-null values.
`<series>.describe ()`	Where <series> is a column of strings from a dataframe, this method returns count of values, number of unique values, the most frequent value, and the frequency of the most common value.
`<series>.dt.strftime (<arg>)`	Returns an ndarray of formatted strings using the specified date format. This is equivalent to a column in a dataframe. <arg> is a string format for datetime, e.g. "%Y-%m-%d %H:%M:%S".
`<series1> <arith> <series2>`	Returns a series with items indicating the evaluation of <series1> <arith> <series2>.
`<series>[<label>]`	Returns the item in <series> at the position named <label>. Series without <labels> are accessed with integers, beginning at position 0.
`<series>.max ()`	Returns the largest value.
`<series>.mean()`	Returns the mean for all the values in column named <column>.

<arith>	**Description**
+	Addition
-	Subtraction
*	Multiplication
/	Division

`<series>.median()`	Returns the median for all the values in column named <column>. Median is the middle item when the values in <column> are sorted into ascending order.
`<series>.min ()`	Returns the smallest value.
`<series.mode()`	Returns the mode for all the values in column named <column>. Mode is the value that appears most frequently in <column>.
`<series>.std ()`	Returns a measure of standard deviation. This is the average distance of the values from the mean, expressed in the same units as the original values.
`<series>.sum ()`	Returns the total of the values.
`<series>.var ()`	Returns a measure of variance. This is how much the group of values differs from the mean of the group. This indicates the spread of the values.
`<series>.value_counts ()`	Returns a series containing counts of unique values.

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.

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.

Additional programming paradigms

Object-oriented

`class <class>`	Creates a new definition for <class>.
`class <class> (<base>)`	Creates a new definition for <class> inheriting <base> class.
`super().<method>`	Invokes a method in the parent class.
`_<attribute>`	Indicates a protected attribute.
`__<attribute>`	Indicates a private attribute.
`<obj>.<method> (<params>)`	Invokes the <method> of the instance <obj> with the parameters <params>.
`<obj>.<attribute>`	Accesses the <attribute> of the instance <obj>.

Functional

`lambda <params> : <expression>`	Creates an anonymous, inline function with parameters <params> that consists of the code in <expression>.
`<generator> = <func> ( <args>)`	Assigns the output of <func> to <generator>. <func> may or may not need <args> passed to it.
`next (<generator>)`	Retrieves the next value from a lazy evaluation <generator> function.
`yield (<value>)`	Used in lazy evaluation generator functions to return one value at a time from a sequence.
`filter (<func>, <obj>)`	Applies <func> to each item in <obj> to determine if the item should be included in the returned result.
`map (<func>, <obj>)`	Executes <func> for each item in <obj>.
`from functools import reduce` `reduce (<func>, <obj>)`	Applies <func> to each item in <obj>, in turn, to produce a single returned result.
`zip (<obj1>, <obj2>, …)`	Items from each <obj> are grouped together based on position, i.e. first items together, second items together, etc. The shortest <obj> determines the number of items in the result. The result is an object made up of tuples.