CY105 Python Reference Card – AY25-1

As of 15AUG24. Material from https://docs.python.org/3/library/

The Four-Step Computational Problem Solving Method

Step 1: Understand the Problem

- **Objective(s):** what is/are the objective(s) of the solution? What is specified and what is implied?
- **Input:** What are the arguments? What kind of data (how much does it need?)
- **Output:** What is the result? What is produced? Is anything returned?
- **Constraints/Assumptions:** What assumptions are made? What data types are appropriate?
- **Relevant equations:** what mathematical equations, if any, are relevant to the solution?

Step 2: Design an Algorithm

Algorithms must:

- Be written in English and not contain computer code.
- Be easy to read and understand.
- Be ordered in sequence.
- Contain enough detail to solve the problem.

Step 3: Implement a Solution

Implement your algorithm from step 2.

- Include a multiline comment for problem description, including inputs, outputs, constraints and assumptions.
- Specify your algorithm using in-line comments.
- Expand your algorithm by writing the code that corresponds to each of the parts of your algorithm.
- Save and test your code after each block of code that you add.

Step 4: Test the Code

Develop test cases to check your program works as desired.

For example:

- **Sequencing:** Test normal values, boundary values, and bad values.
- **Conditionals:** Test all possible paths in conditional statements
- **Loops:** Test what happens when loops are not entered, loop boundaries, and normal values.
- **Combination:** Test each function and procedure individually, and the program in its entirety.

Example Prompt: Write a procedure called count_ones() that takes a list as its only parameter. Your function should count and display the number of times the integer 1 appears in that list.

Sample use case:

```
>>> count_ones([2,1,3,2,1])
There are 2 1s in the list
```

Step 1: Understand the problem:

- Objectives: Goal is to count the number of 1s in a list
- Input: one argument (list of integers)
- Output: displays number of 1s in list (no return value)
- Assumption is that input list contains only numbers
- Need a counter that keeps track of the number of 1s

Step 2: Design an Algorithm

Problem: Count_Ones

Input/parameters: list of integers

Displays: Frequency of 1s in the provided list

```
1. initialize a counter to 0
2. for each number in the list:
      3. determine if the number is equal to 1
      4. if so, increment the counter by one.
5. display the value of the counter to the screen
```

Step 3: Implement your Solution

```
"""
procedure: count_ones
parameters: sequence of integers (list)
displays: the number of times the integer 1 appears in
          the list
"""
def count_ones(integer_lst):
   counter = 0 # initialize a counter to 0

   for number in integer_lst: # for each number:
      if number == 1: # if number is equal to 1
         counter = counter + 1 #  increment count by 1

   # display the value of the counter to the screen
   print("There are", counter, "1s in the list")
```

CY105 Python Reference Card – AY25-1

As of 15AUG24. Material from https://docs.python.org/3/library/

Basic Data Types/Objects

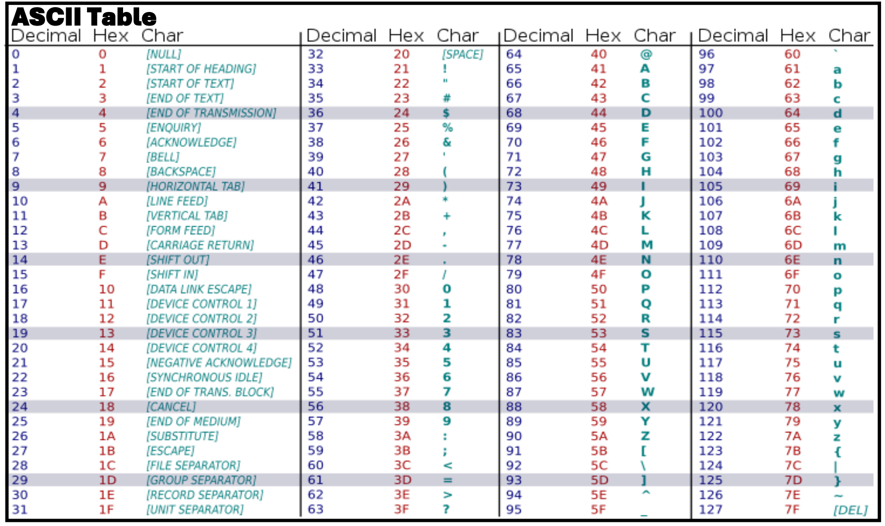
Data Type	Python Type	Description	Examples
integer	`int`	Number without a decimal point (a whole number)	`27, -540`
floating point	`float`	Number with a decimal point	`1.23, -2.38, 4.0`
string	`str`	List of characters (use `' '`, `" "`, or `""" """`)	`'CY105',"c@t", """etc"""`
list	`list`	List of items separated by commas, surrounded by `[]`	`["c",-27,4.0]`
boolean	`bool`	One of two values: `True` or `False`	`True, False`

Units of Data

Size	Description	Size	Description
1 byte (B)	Smallest unit of addressable data	1 terabyte (TB)	10^{12} bytes
1 kilobyte (kB)	10^{3} bytes	1 petabyte (PB)	10^{15} bytes
1 megabyte (MB)	10^{6} bytes	1 exabyte (EB)	10^{18} bytes
1 gigabyte (GB)	10^{9} bytes	1 zettabyte (ZB)	10^{21} bytes

Operators

Type	Sym-bol	Operation	Sym-bol	Operation	Sym-bol	Operation
Arithmetic	+	Addition *or* concatenation (strings)	/	True Division (7/4 evaluates to 1.75)	**	Exponentiation
	-	Subtraction	//	Floor division (integer part) (7//4 evaluates to 1)	E	Powers of 10 (6E1 evaluates to 60.0)
	*	Multiplication *or* replication (strings)	%	Modulo (integer remainder) (8%3 evaluates to 2)	@	Matrix Multiplication (not used in basic types!)
Com-parison	<	Less than	>	Greater than	==	Equal to
	<=	Less than or equal to	>=	Greater than or equal to	!=	Not equal to
Assign-ment	=	simple assignment	-=	x -= y -> x = x - y	/=	x /= y -> x = x / y
	+=	x += y -> x = x + y	*=	x *= y -> x = x * y	%=	x %= y -> x = x % y
Logi-cal	not	Flips bool value True to False or False to True	and	True if both conditions are True otherwise False	or	True if at least one condition is True otherwise False
Membership Operator			in	`x in s` evaluates to True if `x` is a member of `s`, and False otherwise.		



ASCII Table

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[END OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

Built-In Functions

Function name	Arguments	Return Type	Description
`print(item, [item2, item3])`	One or more items (any type)	`n/a`	Displays `item` on the screen. If more than one item, displays them separated by spaces.
`input(prompt)`	prompt (`string`)	`string`	Prompts user for input with the text `prompt` and returns a string containing the result.
`int(item)`	item (`float` or `string`)	`int`	Returns the integer representation of `item`. If `item` is a string, it must be a whole number.
`float(item)`	item (`int` or `string`)	`float`	Returns the floating point representation of item. If `item` is a string, it must be a number.
`str(item)`	item (any type)	`string`	Returns a printable representation of a data type.
`list(seq)`	seq (sequence type)	`list`	Returns a list where each element is a member of the sequence. Example: `list('abc')` returns `['a', 'b', 'c']`
`type(item)`	item (any type)	`varies`	Returns the type of item
`round(num, d)`	num `(float)`, d `(int)`	`float`	Returns num rounded to the nearest *d* digits. By default, d is 0.
`ord(char)`	char (Unicode character)	`int`	Returns the integer representation of a particular ASCII/Unicode character, `char`
`chr(num)`	num (`int`)	`int`	Returns the corresponding ASCII/Unicode character representation of num.
`len(seq)`	seq (sequence type)	`int`	Returns the length of the sequence seq (generally a list or string)
`min(seq)`	seq (sequence type)	`int`	Returns the smallest value (numerically) in a sequence type.
`max(seq)`	seq (sequence type)	`int`	Returns the largest value (numerically) in a sequence type.
`range([start,] stop, [step])`	start (`int`), stop (`int`), step (`int`)	`sequence`	Returns a sequence of numbers based on the values of start, stop and step. The sequence starts at the start value and goes up to (but does not include) the stop value, in increments of step. The step value (defaults to 1) and start value (defaults to 0) are both optional.

CY105 Python Reference Card – AY25-1

As of 15AUG24. Material from https://docs.python.org/3/library/

Math Module

Function name	Arguments	Return Type	Description
`floor(x)`	x (`int`)	`int`	Returns the floor of x, the largest integer less than or equal to x.
`ceil(x)`	x (`int`)	`int`	Returns the ceiling of x, the smallest integer greater than or equal to x.
`isclose(a, b)`	a (`float`), b (`float`)	`bool`	Returns True if the values a and b are close to each other and False otherwise.
`exp(x)`	x (`int` or `float`)	`float`	Returns *e* to the power of x, where *e* = 2.718281… the base of all natural logarithms
`log(x, [base])`	x (`int` or `float`) optional: base (`int`)	`float`	With one argument, return the natural logarithm of x (to base *e*). With two arguments, returns the logarithm of x to the given base.
`pow(x, y)`	x (`float`), y (`float`)	`float`	Return x raised to the power y. Unlike the built-in ** operator, math.pow() converts both arguments to type float. ** is better for integers.
`sin(x), cos(x), tan(x)`	x (`float`)	`float`	Returns the sine of x radians, cosine of x radians, and tangent of x radians, respectively.

Random Module

Function name	Arguments	Return Type	Description
`random()`	n/a	`float`	Returns a random floating-point number `x`, such that 0.0 <= `x` < 1.0
`randint(a, b)`	a (`int`), b (`int`)	`int`	Returns a random integer N such that `a` <= N <= `b`
`randrange([start], stop, [step])`	stop (`int`) optional: start (`int`), end (`int`)	`int`	Returns a random integer N such that 0 <= N < `stop`. Optional parameters `start` and `step` work identically to `range()`.
`choice(seq)`	seq (`list`)	`varies`	Returns a random element from the non-empty list seq. If seq is empty, raises an error.
`choices(seq, k)`	seq (`list`), k (`int`)	`list`	Returns list with k elements that were selected from the population `seq`, with replacement.
`sample(seq, k)`	seq (`list`), k (`int`)	`list`	Returns list with k unique elements that were selected from the population `seq`, without replacement.
`shuffle(seq)`	seq (`list`)	`n/a`	Randomly shuffles the list `seq` in-place (this function is mutable). To return a new list, use `sample(seq, len(seq))`

Stats Module

Function name	Arguments	Return Type	Description
`mean(seq)`	seq (`list`)	`float`	Returns the arithmetic mean of `seq`
`median(seq)`	seq (`list`)	`varies`	Returns the median (middle value) of numeric data using the common "mean of middle two" method
`mode(seq)`	seq (`list`)	`varies`	Returns the single most common data point in `seq`. If there are multiple modes with same frequency, returns first encountered.
`stdev(seq)`	seq (`list`)	`float`	Returns the sample standard deviation of `seq`.
`correlation(seq1, seq2)`	seq1 (`list`), seq2 (`list`)	`Float`	Return's the Pearson's correlation coefficient for two inputs. Returned value will be between –1 and 1.

Precedence of Operators

Evaluate left to right when precedence is equal

1. Parentheses `()`
2. Exponent `**`
3. Multiplication `*`, Division `/`, Floor Division `//`, Modulus `%`
4. Addition `+`, Subtraction `–`
5. Comparison (<, >, ==, etc.), Membership Operators (`in`, `not in`)
6. `not`
7. `and`
8. `or`

Input and Print

```
name = input('What is your name? ')
print('Hi', name+'!')
num = int(input('Guess a number: '))
print(name+', your number was:', num)
```

File I/O

```
in_file = open('data.txt') #open file
out = open('out.txt', 'w') #open file
for writing
for line in in_file: #reads line
  print(line) #prints line to screen
  out.write(line) #writes line to out
in_file.close() #close in_file
out.close() #close out
```

For Loop

```
#print each item in a sequence
for item in sequence:
  print(item) #loop body
```

While Loop

```
num = int(input('Guess a number: '))
count = 1  #initialize accumulator
while (num != 5):
  num = int(input('Guess again: '))
  count = count + 1 #update accumulator
print('Right! It took', count, 'tries.')
```

Chained if, elif, else

```
if boole1:
  #runs if boole1 is True
elif boole2:
  #runs if boole2 is True
  #but boole1 is False
elif boole3:
  #runs if boole3 is True
  #but boole1 and boole2 are False
else:
  #runs if everything else is False
```

CY105 Python Reference Card – AY25-1

As of 15AUG24. Material from https://docs.python.org/3/library/

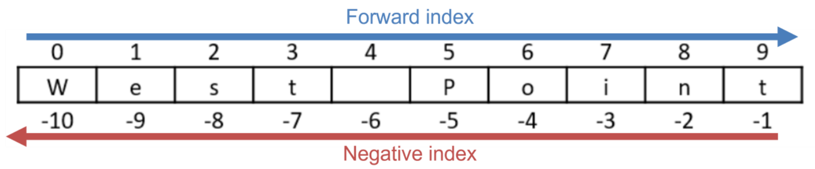
List Methods

Method name	Arguments	Return Type	Description
.count(item)	item (any type)	int	Returns the number of occurrences of item in the list. Does not modify the list.
.append(item)	item (any type)	n/a	Adds item to the end of the list.
.insert(pos, item)	pos (int), item (any type)	n/a	Inserts item at the given position pos.
.remove(item)	item (any type)	n/a	Removes the first occurrence of item in the list. It raises an error if item is not in the list.
.pop(*[pos]*)	optional: pos (int)	varies	Removes and returns the item at position pos in the list. If no argument is provided, removes and returns the last item. Raises an error if the list is empty or the index is outside of the list's range.
.index(item)	item (any type)	int	Returns the zero-based index of the first occurrence of item. Raises an error if item is not in the list. Does not modify the list.
.reverse()	n/a	n/a	Reverses the elements of the list, in place.
.sort(*[reverse=False]*)	optional only	n/a	Sorts the elements of the list in ascending order, in place. If reverse=True is set, will sort the list in descending order.

String Methods

Method name	Arguments	Return Type	Description
.count(substring)	substring (str)	int	Returns the number of occurrences of substring in the string.
.find(substring)	substring (str)	int	Returns the lowest index in the string where substring is found. Returns –1 if the substring is not found.
.index(substring)	substring (str)	int	Like find, but raises an error if substring is not found.
.isalnum()	n/a	bool	Returns True if string is non-empty and all the characters in the string are alphanumeric; otherwise, returns False.
.isalpha()	n/a	bool	Returns True if string is non-empty and all the characters in the string are alphabetic; otherwise, returns False.
.isdigit()	n/a	bool	Returns True if string is non-empty and all the characters in the string are digits; otherwise, returns False.
.isspace()	n/a	bool	Returns True if string is non-empty and all the characters in the string are spaces; otherwise, returns False.
.islower()	n/a	bool	Returns True if all the cased characters in the string are lowercase and there is at least one cased character; otherwise, returns False.
.isupper()	n/a	bool	Returns True if all the cased characters in the string are uppercase and there is at least one cased character; otherwise, returns False.
.lower()	n/a	str	Returns a copy of the string with all the cased characters converted to lowercase.
.upper()	n/a	str	Returns a copy of the string with all the cased characters converted to uppercase.
.strip(*[chars]*)	optional: chars (str)	str	Returns a copy of the string with leading and trailing characters removed. The optional chars string specifies the set of chars to be removed.
.split(*[sep=None]*)	optional: sep (str)	list	Returns a list of words in the string, using sep as the delimiter string. If sep is not specified, spaces are used as the delimiter.
.join(list)	list of strings (list)	str	Returns a string which is the concatenation of the elements in list, which must all be strings. The separator between elements is the string providing this method.

Indexing Lists and Strings



0	1	2	3	4	5	6	7	8	9
W	e	s	t		P	o	i	n	t
-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

Writing Functions

```
# procedure example (note no return)
# print word in all uppercase:
def cow_say(word):
  print(word,'moo!')   #hello moo!

cow_say('hello') #invocation
```

```
#fruitful function example
def my_func(val):
  return val**3 #cubes val

ans = my_func(3) #invocation
print(ans)       # ans is 27
```