print() = print out the words(can be variable or string or int or argv...)

you can print sentence like print(a + b + c) while a,b,c are string variables, it would behave differently if they are int or float

if we print print(a+b), it will be different with (a,b), we can remove the spacing between a and b

input() = inputting data into variable, you can edit input() by adding things into (), like you can try input("> ")

float() = convert the words to float number

int() = convert the words to int

str() = convert the words to string

() it is important to have it close properly all the time

# = comment, will not take in count into coding

""" = writing words across multiple lines"""

arithmetic operation:

+ = plus

- = minus

\* = asterisk(multiply)

/ = slash (divide)

% = percent (modulo operator) (a % b meaning that a is divided by b and give back the remainder)(e.g. 3%4 = 3)

// = floor division, round down the result to the nearest whole number

\*\* = exponentiation (eg 5\*\*2 = 25)

comparision operator:

== = equal

!= = not equal

> = greater than

< = less than

>= = greater than or equal to

<= = less than or equal to

+= = x + 3 = x

-= = x - 3 = x

we can set variable to true and false

"when a = true" is actually equals to "when a"

argv = parameter need to be entered before program run

eg. from sys import argv

a, b = argv

Also a defintion(def function(a, b):) can also called them as argv for a and b

When we are using argv, we have to add f before "" in print() function, in order to make {} work differently

end = "" =dont change lines after the sentence end

"""

skills of printing:

formatter = "{} {] {} {}"

formatter.format(1,2,3,4)

print("formatter is use like this {} {} {} {}".format(formatter))

for this function, it is like puting the number in formatter.format() into {}

you can also call formater.format(formatter, formatter, formatter, formatter) to have loop of strings of{}

here formatter acts as variable

it is just the same as print(f"We'd have {beans} beans.") if we saved the argunments

"""

"\n" = open new line

"\\" = \ in string

"\t" = tab for once

for more about backslash function, you can read about escape sequences in exercise 10

from os.path import exists(for reference to exercise 17)

it can check your file exist or not, returning true or false result. (eg. {exists(to\_file)}), it also need f before"" because it is also argv

\*args = tell python to take all arguments to function and then put them in args as a list, and so you can unpack the arguments later(read exercise 19 for more)

global variable and local variable:

global variable can be accesseed anywhere in program, while local variable can be accessed locally, and cannot be accessed globally

The best way to prevent mixing up variables are to use different names!!

a = file.open() = open the file and store this file in a, you can adjust the mode of opening(like for reading or writing or append...), not a must to create a new file if it is not exist, so you have to adjust the mode that you need, you can also required the format you want to encode by adding(, encoding="...")

file.close() = close the file, it is important to prevent bug

file.read() = read the file, you can adjust the bytes of reading by entering numbers into ()

file.seek(0) = seek the file from 0 bytes, it is important when changing file, like to move the read head to original position

file.readline() = reading the line from the first line, works with print() to print the line. the file is resoponsible for maintainnig the current position after each readline() calls, so it will keep reading each line

file.readlines() = reading all the lines of the file and store it into a list, each line act as an element

return = return the result from a loop or a defined function, like return a + b

import sys

script, parameter = sys.argv

and then we can treat the argument just like variable and be used complicatedly (example in ch23)

UTF-8 = encode string in 8 bits(unicode transformation format 8 bits)

unicode = universal encoding(e.g. ASCII)

some format of encoding(Big5, UTF-32, UTF-8, etc..., if encoding have errors, you can replace the error code by choosing "replace" on encountering errors, but not "strict")

the detail encoding stuff can be referenced to ch23

bits -> bytes(8bits) -> Mb -> Gb -> if Tb..:

Decode bytes Encode String!!

.strip() = strip out the things inside (),for here is "\n"

.enocde() = encode the things before .

.decode() = decode the things before . (must be bytes type, or use "raw\_unicode\_escape" and " unicode\_escape")

For detail decoding things, read ch23 changed.py file

we can import python file if we want, by running terminal with python, we can import our py.file to use it.

so we can make a py.file with a lot of defined function, and then import it to see how it helps.(ch25)

import ex25 = from ex25 import \*

""" if this thing is used in function. it will show up like explanation when we use help(py.file) or help(py.file.function()), we call it as documentation comments"""

sorted() = sort the things inside the () by alphabetic words order, and it depends on the first letter

.pop() = pop out the item before the . , we can use variable = item.pop(0) to save the popped item, 0 means the first item, -1 means the last item

.split() = break the stuffs before . into separate items, and we can save it into a list by list = file.split(' '), while the place to split depends on item inside(), for the example i just show, it will split when it sees a space bar

logic table:

Not False = True

Not True = False

True or True = True

True or False = True

False or True = True

False or False = False

True and True = True

True and False = False

False and True = False

False and False = False

Not True or True = False

Not True or False = False

Not False or True = False

Not False or False = True

Not True and True = False

Not True and False = True

Not False and True = True

Not False and False = True

1 != 0 = True

1 != 1 = False

0 != 1 = True

0 != 0 = False

1 == 0 = False

1 == 1 = True

0 == 1 = False

0 == 0 = True

if, elif, else: loop are important

for loop: use this when we do a countable iteration

while loop: use this when we do an infinite loop

for i in range(len(list)): old method, not directly using list, heavily relied on variable(i in here), but commonly use by everyone

for a, list in enumerate(list, index): new method, suggest to use, index is for the counting of the [0] item and mark it as index

when we use while loop, it is useful to have an incrementor(eg. i = i + 1), so when it reaches to certain value, the while loop stops

if a in b = using if loop when item a present inside b, eg a="a", b="abcefg", in this case a is present in b

cardinal system is like cards, you can draw the card anywhere without considering its arrangement, just like the list index system, so it will start from 0 to count the 1st item

ordinal system means ordering, we have to arrange things from 1st to the last, so 1st indicates 1 in ordinal system

ordinal system number - 1 = cardinal system number

list:[]

2d list: [[],[]]

list[0] = take the item from 0 index and save as variable

list(range(0,6)) = save the range object(0~5) into list

.append(obj) = add the obj item to the last of the list

.pop() = pop out the item before the . , we can use variable = item.pop(0) to save the popped item, 0 means the first item, -1 means the last item(mentioned above)

try:, except ValueError:, finally: are useful when detecting errors(reference to ch35)

changing the variable boolean value is helpful to control loopings

Rules for if statements:

1. Every if-statements must have an else

2. If this else should neverrun because it doesn't make sense, then you must use a die function in the else that prints out an error message and dies, just like we did in the last exercise. this will find many errors

3. Never nest if-statements more than two deep and always try to do them in one deep.

4. Treat if-statements like paragraphs, where each if-elif-else grouping is like a set of sentences. Put blank lines before and after.

5. Your Boolean tests should be simple. If they are complex, move their calculations to variables earlier in your function and use a good name for the not variable.

Rules for loops:

1. Use a while-loop only to loop forever, and that means probably never. this only applies to Python; other languages are different.

2. Use a for-loop for all other kinds of looping, especially if there is a fixed or limited number of things to loop over.

Tips for Debugging:

1. Do not use a "debugger". A debugger is like doing a full-body scan on a sick person. You do not get any specific useful information, and you find a whole lot of information that doesn't help and is just confusing.

2. The best way to debug a program is to use print to print out the values of variables at points in the program to see where they go wrong.

3. Make sure parts of your programs work as you work on them. Do not write massive files of code before you try to run them. Code a little, run a little, fix a little.

Functions about list:

list[:0]='foo'

> ['f','o','o']

list():

append():

extend():

insert():

remove():

pop():

clear():

index():

count():

sort():

reverse():

copy():

.join():

slicing[0:5]:

enumerate():

class: tell python to make a new type of thing

object: two meaning: the most basic type of thing, and any instance of some thing.

instance: What you get when you tell to create a class.

def: How you define a function inside a class.

self: Inside the functions in a class, self is a variable for the instance/object being accessed.

inheritance: The concept that one class can inherit traits from another classes, much like you and your parents.

composition: The concept that a class can be composed of other classes as parts, much like how a car has wheels.

attribute: A property classes have that are from composition and are usually variables.

is-a: A phrase to say that something inherits from another, as in a 'salmon' is-a 'fish'.

has-a: A phrase to say that something is composed of other things or has a trait, as in 'a salmon has-a mouth.'

# class X(Y): Make a class named X that is-a Y.

# class X(object):

def\_\_init\_\_(self, J): class X has-a \_\_init\_\_ that takes self and J params.

# class X(object):

def M(self, J): class X has-a function named M that takes self and J params.

# foo=X(): Set foo to an instance of class X.

# foo.M(J): From foo, get the M function(J), and call it with parameters self, J.

# foo.K=Q: From foo, get the K attribute, and set it to Q.

<https://github.com/vinta/awesome-python> for python libraries