Python variables do not need to be declared. Just state the input. They may need to be casted however. No chars in python, ‘ ‘ and “ “ are used instead

Multiple variables may be defined at once:

I,j,k=1,2,3

print(“Text Here”)

delete with

del variable

Printing variables in strings can be done through these. Multiple variables may be stored

print(“variable ”+Goes+” here”)

print(f”variable {Goes} here”)

string functions include

.upper() ##makes everything upper case

.lower() ##makes everything lower case

.title() ##makes everything word start with an uppercase

strip takes out \t and \n. Variants are

string.strip()## takes all of them

.rstrip takes ##from right side

.lstrip takes ##from left side

Get random ints with

Import random

random.randint(1,10) #between 1 and 10

4\*\*2=16 ##(int\*\*int is to the power of, .5 would be a square root)

Python supports string addition and multiplication, but not power, subtraction, or division

Lists are the python equivalent of arrays for the most part. Accessed using similar method of [int], but can store more than one type of variable in one list. This may create problems for variable specific functions like .title() however if any are not that variable.

List+=”text” ##adds it to the list as the new last element.

.append(“text) ##works just as well

For loops in python are like

For n in list:

print(n)

lists may be created with a range:

nums= list(range(1,101)) ##does not include 101

nums=list(range(1,101,2) ##will skip to the value of current + last number, odds in this case

list functions include

min(nums)

sum(nums)

sorted(list)

sorted(list, reverse=True) #Sorts in reverse order

Access multiple elements of a list

List[1:3]

List[1:-1]

Copy list with

List2=list.copy()

Remove last element of list with

List.pop

Remove specific element with

.remove(‘element’)

You can create a list of lists

Multilist = [[‘yamcha’, ‘tien’], [‘goku’, ‘vegeta’]]

Multilist[1] ##accesses second list

Mulitlist[1][1] ##accesses second of second list

for i in multilist:

for t in i:

print(t) ##access individual ellements of sublist

To make list copies that don’t change with the original

List2=copy.deepcopy(list)

Dictionaries have one key element and then a value element

desserts= {'france': 'crepes',

'netherlands': 'donuts',

'italy': 'gelato',

'germany': 'funnel cake',

'france': 'macaron'} ##it could take a list of values

For k in desserts.keys():

Print(k) ## Get keys

For v in desserts.values():

Print(v) ## Get values

For k, v in desserts:

Print(k, v) ## get both

Create a tuple with

Tuple=(‘1’, ‘2’, ‘1234’)

## Tuples cannot be changed and .reverese() and .sort will not work on it, can be access similarly to lists though, such as with for loops.

Try-except is the python equivalent of try-catch

try:

Age =. Int(input(“How old are you? “))

except ValueError:

Conditionals work similar to java, but without any parentheses:

While b: ##runs so long as b==true

If int < 4: #runs if the int is less than 4

elif int < 18: ##python equivalent of else if and closest counterpart to a switch statement, with last else becoming the default

else: #if previous tests failed

comparisons

str1 == str2

str1 in str2 ##similar, but not identical, in checks if they use the same memory, returning false if not, which happens in a long string.

Str in list ##checks if value is in list, is case sensitive for strings.

Str not in list ## not inverts the in comparison

Str1 > str2 ## Checks if variable one is greater than variable 2, works with any variable, uses ascii values for strings.

Create a function by doing this

Def adder(num1, num2):

‘’’Take two arguments as integers. Returns sum’’’ ##Explains function

Try:

S = int(num1) + int(num2)

Except:

Return ‘NAN

Print(adder(20, 22))

Def hello\_world():

‘’’Says Hello world’’’

Print(“Hello, World!)

Hello\_world()

def say\_hello(first='world', last=''): ##first is a default, last is an optional

'''Takes two arguments as string, print to screen'''

if last:

print(f"Hello, {first.title()} {last.title()}!")

else:

print(f"Hello, {first.title()}!")

say\_hello('john', 'doe')

say\_hello(last='johnson', first='cave') ##can declare order by variable instead of by implication

say\_hello("yamcha")

say\_hello()