Python has **Dynamic Strong Typing**, int + str is not allowed == **Type Error**.
Type Checking : makes sure programmers don't make a certain category of mistakes.
Static Type Checking : Types are checked during compile time. (C,C++,Java,ML)
Dynamic Type Checking: Types are checked during runtime. (Python)
Strongly Typed: You cannot do int + char (Python, Java)
Weakly Typed: You can do int + char (C)
Introspection: Ability of a program to ask questions about itself. help(function) will give info about function

Python Types
String : embedded in single quotes or double quotes not both. 'Hello' or "Hello", str class
Boolean: True or False
Float: 4.3, division will return a float
Int: Integers are arbitrary, can support really big numbers.

String Splicing:
s1 = "This is a long string"
s1[0] == 'T' **#first character in string**
s1[-1] == 'g' **#last character in string**
s1[-4] == 'r' **#4th to last character in string**
s1[0:4] == 'This' **#first index is inclusive : second index is exclusive**
s1[0:3] == 'Thi'
s1[:3] == 'Thi' **#will assume you start at the beginning**
s1[4:7] == 'is'
s1[:-1] == 'This is a long strin" **#everything but the last character**
STRINGs ARE IMMUTABLE, THEY CANNOT BE CHANGED
s1[0] = 't' #NOT ALLOWED
HOWEVER YOU CAN SET S1 TO SOMETHING ELSE
s1 = "new string"
#You can concatenate two strings or characters by using the + operator
"Hello"+"World" == "Hello World"
split(delimiter,**how many splits**) **#method will split string based on delimiter, default is white space. -1 == everything**
s1.split() == ['new','string'] **#returns a list**
join() **#method will join a list of strings into one big string**
'/'.join(["hello","world"] == **will join and add the character /** , "hello/world"

Lists:
[ ] **#square brackets for lists**
Lists can contain [1, 2, '4', [1, 2], True] **#elements don't have to be same type**
You can access any element in the list similar to strings
list1 = [1, 2, [True,False], 4, (2, 3, 4)]
list1[1] == 2
list1[2][0] == True **#you can retrieve list elements inside the list**
lists can be spliced, basically anything a string can do.
LISTS ARE MUTABLE
[1, 2, 3, 4, 5, 6][0:2] = [True] == [True, 3, 4, 5, 6]

sorted() method returns a **NEW** list, so it does not modify the original list.
**You can sort a list of ints or strings, but not a combination of different types.
You can also sort sublists
sort() #method will return the original list sorted

Tuples
( ) **#parenthesis for tuples**
Tuples can contain different types
(1, ”two”, True, [1, 2, 3, 4], (1, 2, 3));
You can splice and extract elements similar to lists and strings.
Tuples must have more than one element or it’s not a tuple
TUPLES ARE IMMUTABLE
However like strings, you can set the variable to a new tuple
x,y,z,p = (1,2,3,4) will automatically extract the elements into the variables
x = 1, y = 2, z = 3, p = 4 You can do the same to lists and strings

Casting
list(parameter) method will convert parameter to a list
tuple(parameter) method will convert parameter to a tuple

pass a placeholder for functions

Dictionaries
{ } **# curly brackets for dictionaries**
d = {1:’a’ , ‘b’:3,}
d[1] will return the value for key 1 == ‘a’
d[‘b’] will return the value for key ‘b’ == 3
CAN DO:
d = {(1, 2, 3): 7}
CANNOT DO:
d = {[1, 2, 3]: 7} **#list is an unhashable type**
dict()

fold and reduce are the same

Legal or Illegal
L2 = [(‘a’,’b’), (1,2), [3,4]]
L2[1][1] = 0 **#not legal**

T1 = ((‘a’,’b’), (1,2), [3,4])
T1[1][1] = 0 **#not legal**
T1[2][1] = 0 **#legal**

Anonymous Functions
lambda : keyword for anonymous functions
lambda x , y : x + y
lambda a : a + 10
If key doesn't exist and you try to get the value, will get an error
d[10]
Use the get method instead it will return none

Global and Local Variables
Inside Function = Local
Outside Function = Global

#Error: local variable s is called before assignment
s = "Cpts355"
def f():
    print(s) **#called before assignment of s not global s local s**
    s = "Cpts322" **#so functions only use local variables**

#in order to print the global variable we must use the global keyword
s = "Cpts355"
def f():
    global s
    print(s) **#will print Cpts355**
    s = "Cpts322"

#global
s = "Cpts355"
def f():
    print(s)

#local
s = "Cpts355"
def ():
    s = "Cpts322"
    print(s)

#power
x = 5**2 == 5^2

nonlocal keyword makes the variable not point to the global variable but the one outside it's scope

Higher Order Functions

map/transform : takes a function and a list and performs the function on the elements in the list returns the mapped list.
#you can pass more than one list in the map, your function should have 2 parameters in this case
map ((lambda x , y: x + y), [1,2,3,4]. [5,6,7,8]) = [6,8,10,12]

filter: takes a function and list and returns a list of elements that satisfy the function
filter ((lambda x : x >= 0), [1,2,-1,3,-2,4]) = [1,2,3,4]

reduce(fold or accumulate): takes a function and list and combines the elements into one single value
reduce ((lambda x, y: x + y), [1,2,3,4,5]) = 15
#not built in need "from functools import reduce"

Class
class <name> <base class>
    <suite> **#attributes**

As soon as the instance is created it is passed to __init__

```
class Account(object):
    def __init__(self, account_holder):
            self.balance = 0
            self.holder = account_holder
```

Iterators
#THESE ARE NOT LISTS!
Two attributes in the iterator interface
__next__ , compute next element in series
StopIteration exception raised if end of series
#don't use next() only for python2, an old method.

__iter__ , returns the iterator

For loop more efficient when going through the series