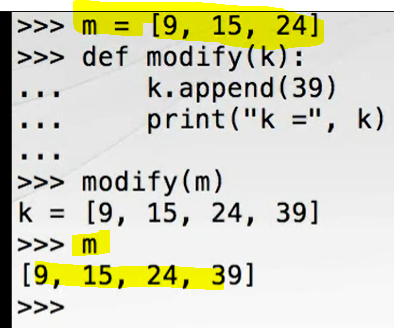
* REPL – Read Evaluate Print Loop
* Underscore in REPL gives last printed value.
* According to unwritten convention, mostly underscores are used as variable for dummy or unused values.
* Python uses indentation level rather than using parenthesis.
* Statement terminated by Colon (:) starts new line and introduces the additional level indentation.
* PEP – Python Enhancement Proposals is a series of documents to manage Python.
* PEP8 is guide for code style
* PEP20 – Zen of Python – 20 aphorisms describing the principles of python
  + import this
* For help, help(math), returns available functions.
* Division using single forward slash (/) returns floating point. With double slash returns integral string
* Scalar Types:
  + Int
    - 10 - 10
    - 0b10 – 2 binary
    - 0o10 – 8 octa
    - 0x10 - 16 hexa
  + Float
    - 3.15
    - 3e8 - 300000000.0
  + None: represents the absence of the value
  + Bool: Logical representation (True or False)
    - Bool(0) and bool(“”) – False
    - Rest of ‘0’ True
* Conditional statement – if…elif..else
* While - break
* input() – accept input from user
* type(e) function to test the type of the element e
* Idiomatic swaping:
  + A= ‘jelly’
  + B= ‘bean’
  + A,b = b,a
  + Print(a) # bean
  + Print(b) # jelly
* **Strings:** Immutable sequences of Unicode codepoints. Default source code encoding is UTF8
  + **“\n” PEP 278 - universal newline support**
  + **Raw string doesn’t support the escape sequence --** r”C:\sample\s.txt”
  + **String constructors:** Str(496)
  + **S= “sample” ->s[5] returns ‘e’**
  + **help(str)**
* **Byte:** Immutable sequences of bytes.
  + **Prefix with small ‘b’. --- > b’data**
  + **String1 = “some text “ bytedata = string1.encode(“utf-8”)**
* **Collections:**
  + **list:** Mutable sequences of objects represented within [ ]. 0 based index
    - Heterogeneous with respect to types.
    - list(‘characters’)
    - **[1,2,3]**
  + **dict:** Mutable mapping of keys to values. {key: value , key1:val2}
    - get value using key with [] operator 🡪 d[k]
    - no order maintained till version XXXX
* **For-Loop**:
  + for item in iterable:
* Special attributes in python are delimited by **double underscores.**
  + \_\_name\_\_ evaluates the module name in REPL
  + \_\_name\_\_ evaluates \_\_main\_\_ while running as script
  + Module code only executes once on first import in REPL
* **Scripts** are directly runnable **Python** programs that do something when executed.
* **Modules** are **Python** files that are intended to be imported into **scripts** and other **modules** so that their defined members like, classes and functions that can be used.
* **Packages**  are a collection of related modules.
* **Standard Python Library** is collection of modules and packages that come bundled with every installation of python.
* **Advanced command line argument parsing library:** argparse and 3rd party: docopt
* **Documenting the code: Docstring convention - PEP 257**
* **The whole shebang:** The first line of the script with **#!.** The program loader to identify which interpreter shall be used.
* Integer object in python cannot be changed.
* id() returns the unique identifier of an object.
* Variables are named references to a object.
* p = [1,2,3] ,q = [1,2,3] : p== q returns True. But, ‘p is q’ returns False
* == Value equality. Value comparison can be controlled programmatically.
* ‘is’ checks the identity. Identity comparison can be controlled language and can’t be changed.
* **Functions:**
  + Function arguments are passed by object references:



* + Default arguments are evaluated when def is evaluated. Will not be changed later (by default).
  + Always use immutable object such as string, int, None as default values
* **Pythons Type System:**

Python can be characterized as having dynamic and strong type system.

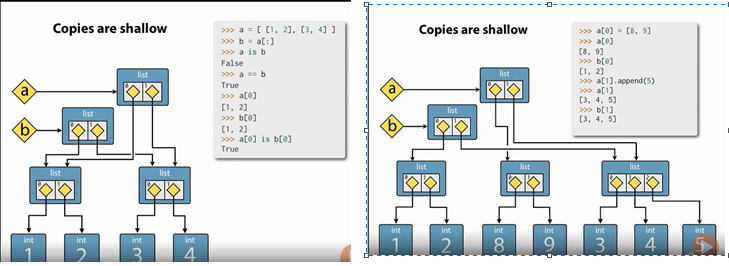
* + Dynamic type: Object types are resolved only at runtime.
  + Strong type: No implicit type conversion
* String can be multiplied with integer that produces a new string with multiple copies of the operand. This is called as “repetition” Operation.
* **Python Name Scopes:**

Scopes are contexts in which named references can be looked up. (LEGB)

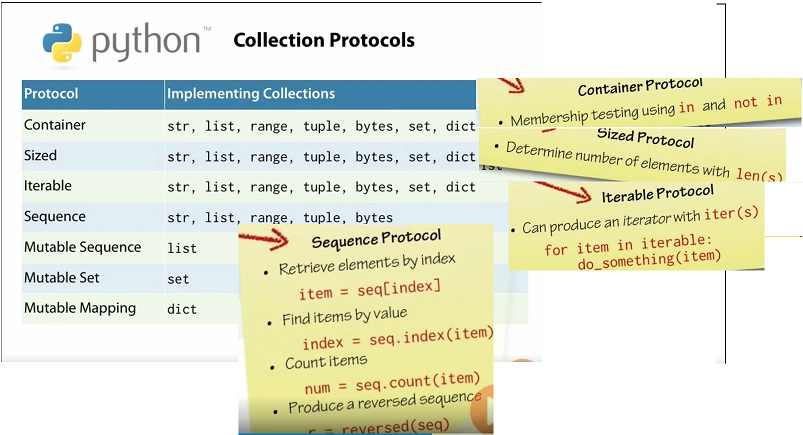
* + **Local**: function scope
  + **Enclosing**: Any and all enclosing function
  + **Global**: Top level of module
  + **Built-in**: provided by Built-in module
* dir() function returns the sorted list of the module attributes including the defined functions.
* Use global keyword to make the global variable available locally.

**Collections:**

* **tuple**: Immutable sequence of arbitrary objects. ()
  + single element tuple can be defined with single element with trailing comma. a=(100,) Otherwise, it will be treated as int or string as per the element.
  + Delimiting parentheses are optional for one or elements.
  + Tuples are useful for multiple return values
  + Tuple unpacking allows us to de-structure directly into named references.
    - Lower, upper = minmax([1, 3, 6, 7,8])
  + (a, (b, (c, d))) = (4, (3, (2,1))) -🡪>>> a=4, b=3, c=2….
  + Tuple constructor -> tuple([2,2,3,5]) or tuple(“hai”)
* **str: sequence of the Unicode characters**
  + Concatenation can be done with ‘+’ and += rebinds the reference to a new object which causes performance degradation.
  + Join: call the join on the separator string. Colors = “;”.join([‘red’, ‘blue,’ white’])
  + For concatenation, use empty string a separator.
  + partition() splits in 3 parts and returns tuple
  + format() - > “I am {0}. My age is {1}”.format(‘jim’, 32)
    - pos(1,2,3)
    - “Galastic position x= {pos[0]} y={pos[1]} z={pos[2]}.format(pos=pos)
* **Range:** is a type of sequence to represent arithmetic progression of integers.
  + range(0,5) 🡪 0,1,2,3,4
  + range(0,10,2) 🡪0,2,4,6,8 with step argument is 2
  + Avoid using the range for iterating over lists. It is un-pythonic
    - Use enumerate() instead that yields (index, value):
    - t= [6,372,88,14,20]
    - for p in enumerate(t):
    - print(p) 🡪 (0,6) (1,372) (2,88)…….etc
* **list: negative index gives value from reverse**
  + slicing is extracting the part of a list**.** Range is half-open **; Slice = seq[start:stop]**
  + **# copy ways - Shallow copy  
    a = s[:]  
    c = s.copy()  
    l = list(s)**

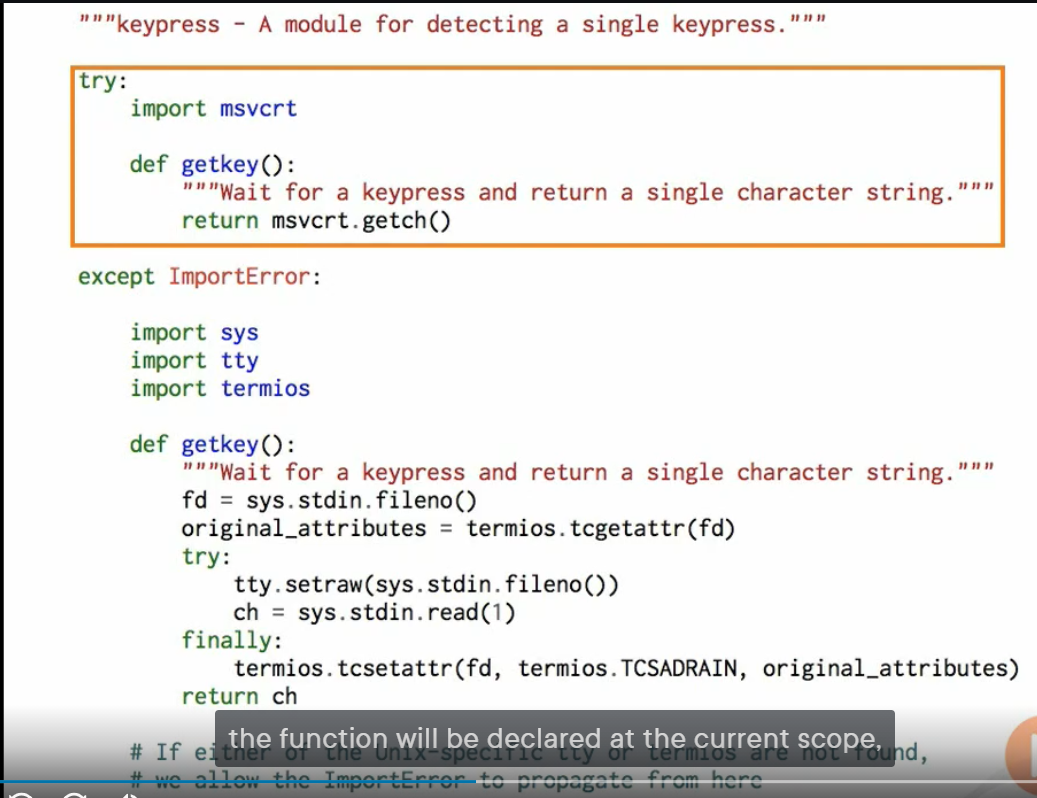


* + **Repetition is shallow:**
  + **S= [0]\*10**
  + **Deletion:**
    - **del seq[index]**
    - **seq.remove(item)** raise ValueError if not present
    - **seq.remove(item)** equivalent to **del seq[seq.index(item)]**
  + **insert:**
    - **seq.insert(index, item)**
    - **using + and +=**
    - **using extend() method**
    - **‘ ‘.join(list) method returns the string**
  + Reverse list with **reverse**() method.
  + Sorting: ist.**sort**() with the optional parameters, key and reverse
  + sorted() built-in function sorts any iterable series and returns a list
  + reversed() built-in function reverse any iterable series and returns an iterator
* **dict: {}** Unordered mapping from unique, immutable keys to mutable values
  + Internally dict maintains the pair of the references to the key object and value objects.
  + Keys must be immutable: str, numbers, tuple ok. Lists are not fine.
  + dict() constructor accepts: iterable series of keys-value 2-tuple
  + **Copying the dict: - shallow copy of key-values**
    - d.copy()
    - dict(d)
  + Update and extend the dict: d.update(d1)
  + Values() method to get values iterable
  + Keys() method to get keys iterable. Default iterable is key. So this not useful method.
  + items() for itereable view on the series of key-value tuple
  + membership test with ‘in’ and ‘not in’ operators.
  + del d[key]
* Pretty Printing: Standard library for comprehensive display of the messages.
  + From pprint import pprint as pp
* **set: Unordered collection of unique, immutable objects. {} with single object rather than a pair by a colon.**
  + set() constructor accepts iterable series of values. Duplicates are discarded.
  + S.add() adds single element
  + S.update([1,2,3]) adds items
  + S.remove(item) - > KeyError if item not present
  + S.discard(item) -> Always succeed. Ignores if not present
  + S.copy() -> shallow copy
  + **Set algebra:**
    - blue\_eyes.union()  
      blue\_eyes.intersection()  
      blond\_hair.difference()  
      blonde\_hair.symmetric\_difference()  
      blond\_hair.issubset()  
      blond\_hair.issubset()  
      blond\_hair.isdisjoint()



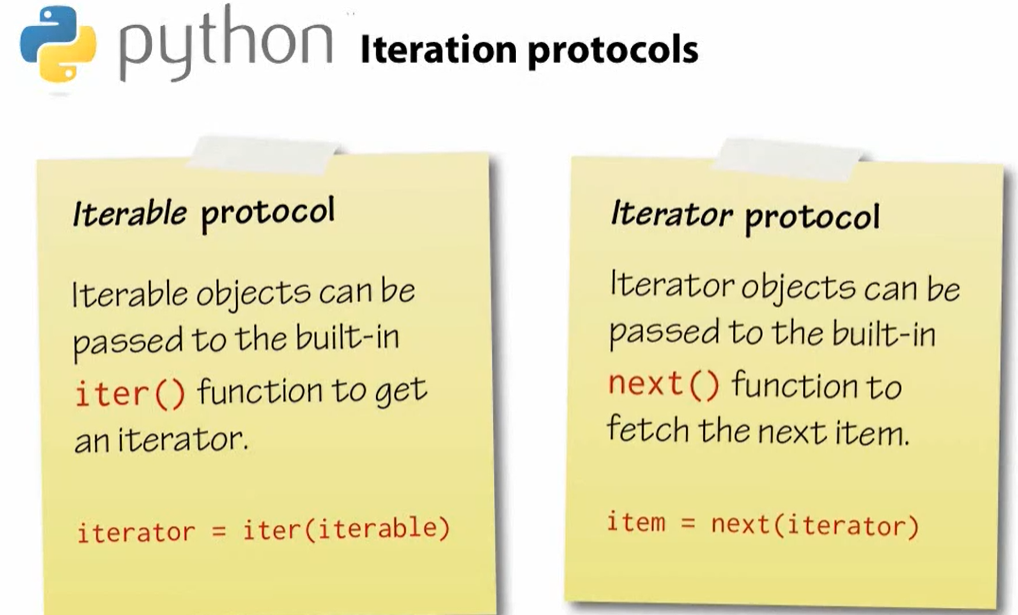
**Exception Handling:**

* IndentationError, SyntaxError, NameError are the Programmer Errors. Shall not be caught normally.
* Pass keyword does precisely nothing
* IndexError, ValueError, KeyError, TypeError
* Don’t catch typeError
* **Philosophies:**
  + **LBYL:** Look Before You Leap:
  + **EAFP**: Easier to Ask Forgiveness than Permission
* **Key press detecting:**
  + **Windows: msvcrt**
  + **Linux/mac OSX: sys,tty,termios**



**Iterables:**

**Comprehensions definition:**

* **List Comprehensions: [ expr(item) for item in iterable ]**
* **Set Comprehensions: : { expr(item) for item in iterable } 🡪 eliminates the duplicate items and order not guaranteed**
* **Dictionary Comprehensions:**
* 
* **Generators in Python:**
  + Definition:
  + At least one yield. Generators are iterators.
  + Generators are Lazy. Good for the infinite.
  + Generator comprehension expression: **(exp(item) for item in iterator)**
  + Generators are single used object.
  + Each time when we call a generator function, it creates new generator object.
  + To recreate the generator from the generator comprehension expression, we must recreate the expression itself.
* squares = (x\*x for x in range(1,5))  
  print(squares)  
  print(list(squares)) #prints: [1, 4, 9, 16]  
  print(list(squares)) #prints: []
* Sum(),any(), all(), zip(), min(), max(), enumerate 🡪 Built-in
* chain(), Islice(), count() 🡪 Standard Library ‘itertools’ module
* **Classes:**
  + Method: A function defined within the class
  + Instance method: Functions which can be called on object
  + Self: The first argument to all instance methods.