

**print()** function: default = '\n'  
**print(objects, separator="", end='\n')**  
 print("Hello World!") ↵ Hello World!

## Operators

**Math:** =(execute/assign) +; -; \*; /; \*\* (exp); += a+=b ↵ a=a+b; -=; \*=; \*\*=; /=; //= floor div (int with no remainder); % (modulo) ↵ remainder from division; *value swap* a,b=b,a;  
**Boolean:** False, True (0, 1)  
**Logical:** and, or, not *modify compare*  
**Comparison:** == (same as); != (is not equal); <; <=; >; >=; is; is not; all ↵ **Boolean** values — (T/F)  
**Membership:** in; not in; - a list, tuple, string, dictionary, or set  
**Identity:** is; is not the same object  
**Binary:** & (and); | (or); ^ (xor - 1 not both); ~ inversion, = -(x+1); << (shift left); >> (shift right)  
 bin(0b0101 <<1) ↵ '0b1010'  
**Sequence Variable Operators**  
**strings:** + -concatenate, \* - repeat;  
**single char slice** s[i]; **range slice** s[i:j:k] **from, to, step** -> start at i, end j-1, increment by count

## Coding Operators

\ **Multiline (explicit join) Statements:**  
 Not needed within [], {}, or ()  
 ; **Multiple Statements on a Line:** not  
 used/needed with **for, if, while**  
 # **line comment**  
 """ **block comment** """

## Number Tools

**abs(x)** ↵ absolute value of x  
**bin(x)** ↵ int to binary bin(5) = '0b101' (a 4, no 2's, a 1); bin(7)[2:] = '111'  
**divmod(dividend,divisor)** from noncomplex numbers ↵ quotient and remainder tuple  
**float(x)** ↵ a floating point number from an integer or string; if x="1.1" print(float(x)\*2) ↵ 2.2  
**hex(x)** ↵ int to hex string hex(65536) ↵ 0x10000 or hex(65536)[2:] ↵ '10000'  
**oct(x)** ↵ integer to octal  
**int(x)** ↵ integer from float/string/hex **pow(x,y,z)** ↵ x to y, if z is present, returns x to y, modulo z **pow(5,2)=25, pow(5,2,7)=4**  
**round(number [,digits])** ↵ floating point number rounded to digits or nearest integer if digits not used **round(3.14159, 4) ↵ 3.1416**  
**max, min, sort** - see data containers  
**None** -> **constant** for null; x=None

## String Tools

**repr(object)** ↵ printable string  
**ascii(str)** ↵ like repr, esc non-ascii  
**eval("Python expression str")** ↵ value  
**chr(i)** ↵ character of Unicode 97 = 'a'  
**input(prompt)** ↵ user input as str  
**len()** ↵ length of str; count of iterable items (list/dictionary/tuple/set)  
**ord(str)** ↵ value of Unicode char.  
**str(object)** ↵ string val of object

**slice selection:** str[start:stop[:step]]; str[:stop]; ↵ a string created by the selection  
**String Formatting**

**.format()** - see 2022 Format Toolbox!

**method:** (1) substitution (2) pure format

(1) 'string {sub0}{sub1}'.format(0, 1)

print("Give {0} a {1}".format('me','kiss'))

(2) '{:format\_spec}'.format(value)

**function:** format(value, spec)

**format\_spec:** (format mini-language string)

[[fill] align] [sign] [# - alt form] [0-forced

pad] [width] [,] [.precision] [type]

x, fmt = 12345.678, "10.2f" ↵ see pg 4

print("Pay \$" + format(x, fmt)) or ↵ format strings

New in 3.6 **f-strings**

print(f"Pay \${x:{fmt}}")

↵ Pay \$ 12,345.68

**.center(width[, fillchar])** string centered in

width area using fill character 'fillchar'

**.capitalize()** ↵ First character capitalized

**.ljust(width [, fillchar])** or **.rjust(same args)**

**.lower()/ .upper()** ↵ change case

**.strip;** or **.lstrip;** or **.rstrip;** + ([chars]) ↵

a string with all or leading, or trailing, [chars] removed. If [chars] included, all are removed. If [chars] omitted or None, the argument removes whitespace

**.swapcase()** ↵ cases exchanged

**.title()** ↵ First Words Capitalized

**.zfill(width)** - left fill with '0' to len width

## String Methods

Str ".is" tests—(Note: tested here for characters 0 to 255) ↵ **True** if all chars in the string meet attribute condition and string =>1 character in length. ↵ **False** if Null

**.isalnum()**—True if all chars in a string are

either .isalpha(), .isnumeric(), .isdigit() or

.isdecimal() \*Note False if your number

contains a decimal point: to vet a variable

v1 as a float: if (type(v1) == float): or

convert in a **try/except** structure

**.isalpha()**—upper and lower case normal

letters plus 64 printable characters between

chr(170) and chr(255)

**.isdecimal()**—digits 0,1,2,3,4,5,6,7,8,9

**.isdigit()**—0 to 9 plus superscripts <sup>2</sup> (178),

<sup>3</sup> (179), and <sup>1</sup>(185)

**.isidentifier()**—tests a string to see if it is a

valid Python identifier or keyword

**.islower()**—lower case ltrs plus 36 printable

characters between chr(170) and chr(255)

**.isnumeric()**— .isdigit plus <sup>1</sup>/<sub>4</sub> (188),

<sup>1</sup>/<sub>2</sub> (189), and <sup>3</sup>/<sub>4</sub> (190)

**.isprintable()**—189 of the 256 characters

between 0 and 255 starting with the space

chr(32) sequentially to ~ chr(126), then chr

(161) to (255) except for chr(173)

**.isspace()**—true for chrs (9-13), (28-32),

(133) and (160). Note space: " " is chr(32)

**.istitle()**—for all practical purposes, every

word in a string begins with a capital letter

**.isupper()**—normal upper case plus 30

printable characters between chr(192-222)

**.casefold()** ↵ casefold - caseless matching

**.count(sub[,start[,end]])** ↵ # of substrings

**.encode(encoding="utf-8", errors="strict")**

**.endswith(suffix[, start[, end]])** ↵ T/F

**.expandtabs()** replace tabs with spaces

**.find(sub[, start[, end]])** ↵ the index of

**substring** start, or -1 if it is not found;

print('Python'.find("th")) ↵ 2

**.index(sub[,start[,end]])** = **.find** but failure

to find sub causes **ValueError**  
**separator.join([string list])**  
 joins strings in iterable with **sep**  
 char; can be null  
**.partition(sep)** ↵ 3 tuple: before,  
 sep, after  
**[new 3.9] .removeprefix(prefix,/)**  
 and **.removesuffix(suffix,/)**  
**.replace(old, new[, count])** ↵  
 substring old replaced by new in object;  
 if count is given, only the count  
 number of values are replaced  
**.rfind(sub[, start[, end]])** ↵  
 lowest index of substring in slice  
 [start:end]. -1 on fail  
**.rindex()** rfind but fail ↵ **ValueError**  
**.rsplit**—like **split**, except splits  
 from right  
**.split([sep] [maxsplit=])** ↵ word  
 list, default sep is space(s)  
**.splitlines(keepends=False)** ↵  
 list of lines broken at line boundaries  
**.startswith(prefix[,start[,end]])**  
 ↵ **True/False** prefix can be a tuple  
**.translate(table)** map to table  
 made with **.maketrans(x,[y[,z]])**  
 (maketrans takes/makes strings)

## Admin Built-in Functions

**pass** (placeholder - no action)  
**del** deletes variables, data containers,  
 items in iterables: del mylist[x]  
**breakpoint** enters debugger - with  
 wrapper ensures **\_exit** method  
**bool(expression)** ↵ T/F (default)  
**callable(object)** ↵ True if it is  
**help(object)** invokes built-in help  
 system, (for interactive use)  
**id(object)** ↵ unique identifier  
**:= (New [3.8])** - assignment  
 expression operator assigns values to  
 variables inside a larger expression  
**bytearray([source[, encoding**  
 [, errors]]) ↵ a new bytearray; source  
 can be an iterable of integers 0 to 255,  
 an integer defining array size, or a  
 string with encoding which will be  
 converted to bytes using **str.encode()**  
**globals()** ↵ a dictionary of current  
 global symbols of the current module  
**isinstance(object, classinfo)** ↵  
 True if object is an instance of classinfo  
**issubclass(object, classinfo)** ↵  
 True if object is a subclass of classinfo  
**locals()** ↵ a dictionary of the current  
 local symbol table  
**vars([object])** ↵ the **\_\_dict\_\_**  
 attribute for a module, class,  
 instance or object

## Looping

**while (True expression ):**  
 process data statements;  
**[else:]** if expression is false, do once  
**for** expression to be iterated: **usually**  
**with in or range (start, stop [,step])**  
**[else:]** executed unless a break  
 statement interrupts execution cycle  
 In **both for** or **while** loops:  
**break** ends the innermost loop and  
 prevents **else:** from executing,  
**continue** skips to next loop cycle.  
 \*if also supports an else statement and  
 can be confusing if not placed as a peer

## Decision Making

**if** **elif** **else**:  
**if** some True statement: #execute code  
**elif alt** True statement: # do this code  
**else**: # otherwise execute this code  
**Ternary if**: an inline **if** that can be use in formulas  
`print(x if x in myword else "", end="")`

## Error Management

use in error handling blocks  
**try**: #code with error potential  
**except [error type]**: #code if any error or a specified error occurs  
**else**: #otherwise do this code  
**finally**: #do this either way  
**assert**: condition = **False** will raise an **AssertionError**  
**raise** forces a specified, usually custom, exception. Custom errors are created as their own class. **ex**:  
`class TempTooHigh(Error):`  
 `"Arduino input over max range"`  
 `pass`

## File Access and Methods

`filepath=r"C:\files\mytest.txt"`  
 Python natively handles only strings in files  
**open(filepath [,mode], buffering)**  
 Typical usage: open in with structure:  
**with open("wholefilepath") [as xfile]:**  
 `xfile=mytest.read().splitlines()`  
**\*\*with** structure automatically closes a file  
 Helpful *methods*: **.read()**, **.read(size)**, **.readline()**, **.readlines()**, **.write(string)**, **.close()**, **.splitlines([keepends])**, **list(openfile)**.  
**.close()** - not needed in with structure  
 \*Many other functions **not** shown here  
**File Modes:** **open** for  
**'r'** reading (default)  
**'w'** writing, truncating the file first  
**'x'** exclusive creation, fails if it exists  
**'a'** writing, appending to the end of the file **if** it exists  
**'b'** binary mode  
**'t'** text mode (default)  
**'+'** for updating (reading and writing), ie. "r+" or "w+"

## Object Methods

Working with object attributes (most useful for created class objects)  
**getattr(object, 'name' [, default])**  
`listatr = getattr(list, '__dict__')`  
 for item in listatr:  
 `print(item, listatr[item], sep=" | ")`  
**setattr(object, 'name', value)**  
**hasattr(object, 'name')**  
**delattr(object, 'name')**  
**exec(string or code obj[, globals [, locals]])** dynamic code execution  
**compile(source, filename, mode, flags=0, don't\_inherit=False, optimize=-1)** create a code object that **exec()** or **eval()** can execute  
**hash(object)**  $\rightarrow$  integer hash value if available  
**dir()**  $\rightarrow$  names in current local scope  
**dir(object)**  $\rightarrow$  valid object attributes

## Functions \* boldface not in this basic toolbox

<code>abs()</code>	<code>callable()</code>	<code>enumerate()</code>	<code>hasattr()</code>	<code>list()</code>	<code>pow()</code>	<code>staticmethod</code>
<code>all()</code>	<code>chr()</code>	<code>eval()</code>	<code>hash()</code>	<code>locals()</code>	<code>print()</code>	<code>str()</code>
<code>any()</code>	<b><code>classmethod</code></b>	<code>exec()</code>	<code>help()</code>	<code>map()</code>	<b><code>property()</code></b>	<code>sum()</code>
<code>ascii()</code>	<code>compile()</code>	<code>filter()</code>	<code>hex()</code>	<code>max()</code>	<code>range()</code>	<b><code>super()</code></b>
<code>bin()</code>	<code>complex()</code>	<code>float()</code>	<code>id()</code>	<b><code>memoryview</code></b>	<code>repr()</code>	<code>tuple()</code>
<code>bool()</code>	<code>delattr()</code>	<code>format()</code>	<code>input()</code>	<code>min()</code>	<code>reversed()</code>	<code>type()</code>
<code>breakpoint()</code>	<code>dict()</code>	<code>frozenset()</code>	<code>int()</code>	<b><code>object()</code></b>	<code>round()</code>	<code>vars()</code>
<code>bytearray()</code>	<code>dir()</code>	<code>getattr()</code>	<code>isinstance()</code>	<code>oct()</code>	<code>set()</code>	<code>zip()</code>
<code>bytes()</code>	<code>divmod()</code>	<code>globals()</code>	<code>issubclass()</code>	<code>open()</code>	<code>setattr()</code>	<code>__import__()</code>
			<code>iter()</code>	<code>ord()</code>	<code>slice()</code>	
			<code>len()</code>		<code>sorted()</code>	

## Errors

### ArithmeticError\*

**AssertionError**  
**AttributeError**  
**BaseException**  
**BlockingIOError**  
**BrokenPipeError**  
**BufferError\***  
**BytesWarning**  
**ChildProcessError**  
**ConnectionAbortedError**  
**ConnectionError**  
**ConnectionRefusedError**  
**ConnectionResetError**  
**DeprecationWarning**  
**EOFError**  
**EnvironmentError**  
**FileExistsError**  
**FileNotFoundError**  
**FloatingPointError**  
**IOError**  
**ImportError**  
**IndentationError**  
**IndexError**  
**InterruptedError**  
**IsADirectoryError**  
**KeyError**  
**KeyboardInterrupt**  
**LookupError\***

### MemoryError

**ModuleNotFoundError**  
**NameError**  
**NotADirectoryError**  
**NotImplementedError**  
**OSError**  
**OverflowError**  
**PermissionError**  
**ProcessLookupError**  
**RecursionError**  
**ReferenceError**  
**RuntimeError**  
**SyntaxError**  
**SystemError**

### TabError

**TimeoutError**  
**TypeError**  
**UnboundLocalError**  
**UnicodeDecodeError**  
**UnicodeEncodeError**  
**UnicodeError**  
**UnicodeTranslateError**  
**ValueError**  
**WindowsError**  
**ZeroDivisionError**  
 \*non-system-exiting exceptions

**Helpful definitions:** **Iterable**: an object that can return members 1 at a time  
**Mutable**: can be changed **Immutable**: can't  
**Ordered**: held in a fixed sequence **Unique**: can not contain any duplicate values  
 Set concepts and terms: diagram next page

## Universal Iterable Tools

**all(iterable)**  $\rightarrow$  True if all elements are True  
**any(iterable)**  $\rightarrow$  True if any element is True \*all and any are both FALSE if empty  
**del(iterable instance)** - delete  
**enumerate(iterable, start = 0)**  $\rightarrow$  tuples list  
`alist = ['x','y','z']; l1 = list(enumerate(alist)); print(l1)`  
 $\rightarrow$  [(0,'x'), (1,'y'), (2,'z')]  
**filter(function, iterable)** selector for elements for which function is True  
**iter and next(iterator, default)** create iterator with **iter**; fetch items with **next**; default returned if iterator exhausted, or **StopIteration**  
`team = ['Amy', 'Bo', 'Cy']; it1 = iter(team); myguy = ""`  
`while myguy is not "Cy":`  
 `myguy = next(it1, "end")`  
 `print(myguy)`  
 $\rightarrow$  Amy  
Bo  
Cy

**map(function, iterable)** can take multiple iterables - function must take just as many  
`alist=[5,9,13,24]; x = lambda z: (z+2)`  
`list2 = list(map(x, alist)); print(list2)`  $\rightarrow$  [7,11,15,26]  
**range([start,] stop [,step])**  
`alist=["Amy", "Bo", "Cy"]`  
`for i in range(0, len(alist)):`  
 `print(str(i), alist[i])` # note slice  
**reversed()** reverse **iterator**: list or tuple  
`alist=["A", "B", "C"]; print(alist)`  
`alist.reverse(); print(alist)`  
`rev_iter = reversed(alist)`  
`for letter in range(0, len(alist)):`  
 `print(next(rev_iter), end=", ")`  
 $\rightarrow$  'C', 'B', 'A'  
A, B, C,  
**sum(iterable [, start])** all numeric  
**ex**: `if a=[8,7,9] then sum(a)`  $\rightarrow$  24  
**type([iterable])**  $\rightarrow$  object datatype  
**zip()** creates aggregating iterator from multiple **iterables**,  $\rightarrow$  iterator of tuples of  $i^{th}$  iterable elements from each sequence or iterable.

## Iterable Data Container

### Methods & Operations

$\downarrow$  i,j,k: indexes | x: values/ objects  
**L / T / D / S / F / SF**  $\rightarrow$  instances of:  
**list, tuple, dictionary, set, frozen set, both**  
**Unique Data Type Statements/Methods**  
**LISTS:** **[ ]** - Ordered, Mutable  
**create** `L=[]; L=[[x,x]...]; L=list(L/T/S/F); list(D)`  $\rightarrow$  list of all dictionary **keys**; `list(D.values())` for list of values  
**L=L2[i:j:k]** new list from slice of L2  
**add/remove items** `L1+L2` concatenate (lists only); **.append(x)** where x is string or data object; **.clear()** remove all members; **.copy()** duplicate list; **.extend(iterable)** adds iter members; strings add letters as members; **insert** (item, position); **.pop(i)** return and remove  $i^{th}$  item, last item if no i; **.remove(x)** remove first item = x  
**query** `L[x]`  $\rightarrow$  value at position x, can be multiple values: `a,b=L[2:4]; x.count(x)` find number of instances of x in list; **.index(x[,at/after index i[,before index j])**  $\rightarrow$  slice position of string or value x in list, **ValueError** if not in found; **len(L)**; **max(L)**; **min(L)**; `x in L`; `x not in L`  
**manipulate** **.sort(key=None/function, reverse=False)**; **sorted(L[,reverse])**; **L.reverse()** reverse item order;  
**TUPLES:** **( )** - Ordered, Immutable  
**create** `T=()`; `T=(x,[x],(x)...) ;`

## List Comprehensions

Make a new list with exclusions and modifications from an existing list or tuple: brackets around the expression, followed by 0 to many **for** or **if** clauses; clauses can be nested:  
**new\_list = [(modified)item for item in old\_list if some-item-attribute of (item)]**  
`atuple=(1,-2,3,-4,5)`  
`mylist=[item*2 for item in atuple if item>0]`  
`print(atuple, mylist)`  
 $\rightarrow$  (1, -2, 3, -4, 5) [2, 6, 10]  
**if modifying items only:** `up1list = [x+1 for x in L]`



**TUPLES: (continued from pg 2)****T = tuple(T/L/S/F)**

**add members** +=(x,[x]) add 1 or more items, note comma for 1 item;  
**T1 + T2** concatenate (tuples only)  
**query** =T[i:j] get slice values, j is last item + 1; **.count(x)** find number of instances of x in tuple; **T.index(x[,at/after index i][,before index j])** ↵ slice position of possible member x; **min(T); max(T); len(T); x in T; x not in T**  
**manipulate** sorted (T, reverse=T/F); **T[::-1]** reverse order

**DICTIONARIES:** { } Mutable, Unordered, Unique keys **k** ↵ 'key', **v** ↵ 'value'

**create** D={k:v, [,k:v]}; =dict(i=j [,k=i]); =dict(zip(L1, L2)); **D2=D1.copy()**; =dict.fromkeys (L/T/F, pair members with v/None/ iterable); **add/remove members**

**D[k]=new\_value**; **D.update(D2)** add D2 items to D replacing dup values; **D=(\*\*D|\*\*D2)**; **D.setdefault(k[,default])** return value if k in dict, if not, insert and return default; **D.clear()**; **del D[k]** remove member; **D.pop(k)** ↵ v and removes k; **new [3.9]: D=D2|D3**; **D|=k/v pairs**; **query** x=D[k] ↵ v or **keyerror** if no k; **x=D.get(k[,x])** like D[k] but ↵ x if no k; **len(D)**; **Dictionary views:** **D.keys()**, **D.values()**, **D.items()** for **items view**, x ↵ a **list** of key:value **tuples**; all **views** can all be **iterated**

**x in D.view**; **x not in D.view**;  
**manipulate** D[existing k]=value change value; **[new in 3.8]** where **ri** is a reversed iterator **ri=reversed(D.view)** iterate with **next(ri)**; **sorted(D.items())**

Use enumerate to make a dictionary. **ex:** mydict = dict(enumerate(mylist))

**SETS:** Unique, Mutable, Unordered  
**create** S={x,x,x}; **S=set(L/T/F)**;  
**S='string'** ↵ unique letters

**FROZENSETS:** immutable after creation;  
**create** F=frozenset([iterable]) ↵ only

**Set & Frozenset Methods and Operations**

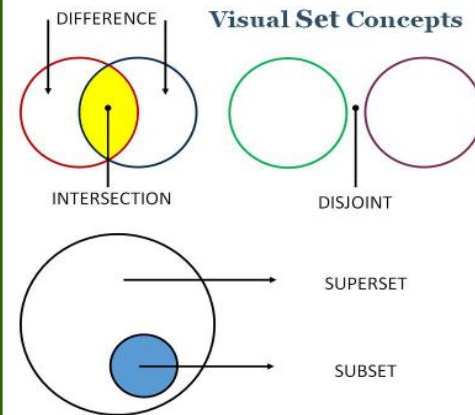
**SF.copy()** Return a shallow copy.  
**SF.symmetric\_difference(SF2)** or **SF^SF2** elements in either, not both  
**SF.union(SF2)** or **SF3=SF1 | SF2[|...]** merge the sets  
**SF.intersection(SF2)** or **SF1 & SF2** intersection of S1 & S2  
**SF.difference(S2)** or **SF-SF2** unique in SF  
**query (Sets & Frozensets)** **len(SF)**;  
 Boolean Tests: **x in SF**; **x not in SF**;  
**SF.isdisjoint(SF2)** T if no common items  
**SF.issubset(SF2)** & **SF1<=SF2** One set is contained by the other.

**SF1<SF2** set is a proper subset  
**SF1.issuperset(SF2)** or **SF1>=SF2** Every element of SF1 in SF2  
**SF1>SF2** set is a proper superset

**Sets ONLY add/remove members**

**S.remove(element)** Key Error if missing;  
**S.discard(element)** no error if missing;  
**S.pop()** remove/return random element; **S.clear()**; **S.add(i)**; **in**; **not in**;  
**S.update(iterable)**; or **S1 |= S2**; These add members from iterable(s) or set(s).

**S.intersection\_update(other iterables);** or **S1 &= S2**; Keep universal elements.  
**S.difference\_update(iterable)** or **S1 -= S2** Remove members found in others.  
**S.symmetric\_difference\_update(iterable)** **S1 ^= S2**; keep unique elements only

**\*/\*\* for iterable (argument) unpack**

\* for list & tuples: **Ex:** a,\*b,c = [1,2,3,4,5]  
 ↵ a=1, c=5, b=[2,3,4]  
 \*\* for dictionaries  
**d1={1:'a', 2:'b'}**; **d2={2:'c', 3:'d'}**;  
**d1={\*\*d1, \*\*d2}** or new in [3.9] **d1|=d2**  
 ↵ **d1={1:'a',2:'c',3:'d'}**

**User Functions**

**def** - command to create a user function

**def function\_name (args or kwargs):** → **return(variable object)** return the value(s) that a function derived - or - **yield/next** in a generator function, **yeild** returns a sequential value incremented by **next** after the function call (see below)

**global x** creates global variable - defined **inside** a function  
**nonlocal** makes a variable in a nested function valid in an outer function

**Creating a Function**

(required in red, optional in green)

↵ **command key word** ↵ **arguments**

**\*1 def name** (input or defined params):  
 ↵ **new function name** ↵ **colon** ↵  
**[ \*2 """ a docstring """** (can be multiline) ]  
**\*next segment code block**  
**\*last segment return**(value to pass back)  
 or a **generator** passed using **yield**:

**Example**  
 vowels, myword = 'aeiouy', 'idea'  
**def geni(wordin):**  
 for letter in wordin:  
     **yield**(letter)  
 for letter in geni(vowels):  
     **print**(letter if letter in myword **else** "")  
     **next**

**Lambda:** an unnamed **inline function**

**lambda [parameter(s)]: expression**  
**z = lambda x: format(x\*\*3, ".2f");**  
**print(z(52.1))** ↵ **141,420.76**

**CLASS** - an object **blueprint** (required in red, optional in green)

Common components of a class include:

**1 \*inheritance** creates a "derived class"  
 ↵ **command key word** ↵ **colon** ↵

**class class-name** (inheritance):  
 your class name ↵ **class definition header**  
 Class creates a namespace and provides **instantiation** and **attribute references**

**2 a docstring**, "Docstring example"

**3 instantiation with special method:**

**def \_\_init\_\_(self, arguments):**  
 ~ autoinvoked when class is created;  
 ~ arguments are passed when a class instantiation is called.  
 ~ Includes variable name assignments, etc.

**\*4 function definitions and local variable assignments**

**Example**  
**1** class mammalia(object):  
**2** "A class for mammal classification"  
**3** def \_\_init\_\_(self, order, example):  
     self.ord = order  
     self.ex = example  
     self.cls = "mammal"  
**4** def printInfo(self):  
     info="class/order: " + self.cls + "/" + \ self.ord + ", Example:" + self.ex  
     print(info)

**mam\_instance = mammalia("cetacea", "whales")**  
**mam\_instance.printInfo()**

↵ **class/order: mammal/cetacea, Example: whales**

**\*args and \*kwargs**

used to pass an unknown number of arguments to a function.

**\*args** is a **list**

**def testargs(a1, \*argv):**  
 print("arg#1: ", a1)  
 for ax in range(0, len(argv)):  
     print ("arg#" + str(ax+2) + " is " + argv[ax])  
 testargs('B', 'C', 'T', 'A')

**\*kwargs** is a **keyword -> value pair**  
 keyword is **not** an expression

**def testkwargs(arg1, \*\*kwargs):**  
 print ("formal arg:", arg1)  
 for key in **kwargs**:  
     print ((key, **kwargs**[key]))  
 testkwargs(arg1=1, dog2="two", dog="cat")

**Example of: function, \*, \*args**

**def myfunc(\*args):** # function unknown # args  
 print(\*args)  
 my\_list = ['a1','b2','x','c3'] # create list  
 myfunc(\*my\_list) # new list expanding old  
 del my\_list[2] # remove 2nd item  
 myfunc(\*my\_list) # reprint to prove

**NEW IN 3.10****Case Pattern Matching**

"Takes an expression and **compares** its value to successive patterns given in one or more **case blocks**."

**match** value | string | list | T/F:

**case** value | string | list | T/F:

<responding code>

**case...**

**case \_:** # nothing matched must be the last case match

~ case can match multiple objects

~ a list case object can be unpacked \* case ["paint", \*colors]:

for color in colors ..... etc.

~ can capture subpattern using or/as

case [x, (1 | 3 | 5 | 7) as choice]:

Code for using the filter command, filter takes 2 components, (1) a function and, (2) a data container.

Command word that lets Python know you are applying an anomonus inline function

Variable(s) delimited by a colon

Code for the filter function

SelectedContacts=[filter( **lambda** x: x[0]=="G" , ContactTuple)]

the filter command creates an iterator % NOT a list

the lambda command can be used as filter's function,

... the 2nd filter parameter is a list, tuple or string

**f-string Formatting**

[new 3.6]

**Conversion Types**

- 'd' Signed integer decimal.
- 'i' Signed integer decimal.
- 'o' Signed octal value.
- 'u' Obsolete type - it is identical to 'd'.
- 'x' Signed hexadecimal (lowercase).
- 'X' Signed hexadecimal (uppercase).
- 'e' Floating point exponential format (lowercase).
- 'E' Floating point exponential format (uppercase).
- 'f' Floating point decimal format.
- 'F' Floating point decimal format.
- 'g' Floating point format. Uses lowercase exponential format if exponent is less than -4 or not less than precision, decimal otherwise
- 'G' Floating point format. Uses upper-case exponential format if exponent is less than -4 or not less than precision, decimal format otherwise.
- 'c' Single character - accepts integer or single character str
- 'r' String - uses repr() to convert object
- 's' String - uses str() to convert object
- 'a' String - uses ascii() to convert object
- '%' Puts '%' character before result

**conversion flags**

- '#' conversion will use "alternate form"
- '0' conversion zero padded for numerics
- '-' value is left adjusted (overrides '0')
- ' ' (space) Leave a space before a + or #
- '+' A sign character ('+' or '-') will precede conversion (overrides "space" flag).

**Integer Bitwise Operations**

Operation / Result

- x | y**  
bitwise **or** of x and y
- x ^ y**  
bitwise **exclusive or** x and y
- x & y**  
bitwise **and** of x and Y
- x << n**  
x shifted left by n bits
- x >> n**  
x shifted right by n bits
- ~x**  
the bits of x inverted

**Bytes and Bytearray Operations**

- x. = method can be used w/ "bytes." or "bytearray."
- i.e.,  
x.count(sub[, start[, end]]) is same as bytes.count(sub[, start[, end]]) or bytearray.count(sub[, start[, end]])
- x.decode(encoding="utf-8", errors="strict")
- x.endswith(suffix[, start[, end]])
- x.find(sub[, start[, end]])
- x.index(sub[, start[, end]])
- x.join(iterable)
- static bytes.maketrans (from, to)
- static bytearray.maketrans (from, to)
- x.partition(sep)
- x.replace(old, new[, count])
- x.rfind(sub[, start[, end]])
- x.rindex(sub[, start[, end]])
- x.rpartition(sep)
- x.startswith(prefix[, start[, end]])
- x.translate(table[, delete=b])
- x.center(width[, fillbyte])
- x.ljust(width[, fillbyte])
- x.lstrip([chars])
- x.rjust(width[, fillbyte])
- x.rsplit (sep=None, maxsplit=-1)
- x.rstrip([chars])
- x.split(sep=None, maxsplit=-1)
- x.strip([chars])
- x.capitalize()
- x.expandtabs(tabsize=8)
- x.isalnum()
- x.isascii()
- x.isalpha()
- x.isdigit()
- x.islower()
- x.isspace()
- x.istitle()
- x.isupper()
- x.lower()
- x.splitlines (keepends=False)
- x.swapcase()
- x.title() x.upper()
- x.zfill(width)

**Operators and Precedence**

- lambda**
- if - else**
- or · and · not x** (Boolean)
- in · not in · is · is not**
- < · <= · > · >= · != · ==**
- | · ^ · &** bitwise OR, XOR, AND
- << · >>**
- +**
- 
- \***
- @ · / · // · %** (multiply, matrix multiply, division, floor division, remainder)
- +x · -x · ~x** (pos, neg, bitwise NOT)
- \*\*** (exponentiation)
- await** (Await expression)
- x[index] · x[index:index] · x(arguments...)**
- x.attribute** (subscription, slicing, call, attribute ref)

**Built-in Types** numerics, sequences, mappings, classes, instances, exceptions**Numeric Types**

int, float, complex constructors:

complex(real, imaginary) *imaginary defaults to 0***Numeric Operations**

- x + y** sum of x and y
- x \* y** product of x and y
- x // y** floored quotient of x and y
- x % y** remainder of x / y
- x** x negated
- abs(x)** absolute value x
- int(x)** x converted to integer
- float(x)** x converted to floating point
- complex (real, imaginary)** imaginary defaults to 0
- c.conjugate()** conjugate of complex number c
- divmod(x, y)** the pair (x // y, x % y)
- pow(x, y)** x to the power y
- x \*\* y** x to the power y
- round(x[,n])** round to n digits, half to even

**math** module (import math) adds these rounding operations:

- math.trunc(x); math.floor(x); math.ceil(x)**

**Sequence Operations**

- x in s** True if an item of s is equal to x
- x not in s** False if an item of s == x
- s + t** the concatenation of s and t
- s \* n or n \* s** concatenate s n times
- s[i]** ith item of s, origin 0
- s[i:j]** slice of s from i to j
- s[i:j:k]** slice of s from i to j step k
- len(s)** length of s
- min(s)** smallest item of s
- max(s)** largest item of s
- s.index(x[, i[, j]])** index of the first occurrence of x in s (at or after index i and before index j)
- s.count(x)** number of occurrences of x in s

**Mutable Sequence Operations**

- s[i] = x** item i of s is replaced by x
- s[i:j] = t** slice of s from i to j is replaced by the contents of the iterable t
- del s[i:j]** removes i to j; same as **s[i:j] = []**
- s[i:j:k] = t** the elements of s[i:j:k] are replaced by those of t; start, stop, step
- del s[i:j:k]** removes the elements of s[i:j:k] from the list
- s.append(x)** appends x to the end of the sequence
- s.clear()** removes all items from s (same as **del[:]**)
- s.copy()** creates a shallow copy of s (same as **s[:]**)
- s.extend(t)** or **s +=** extends s with the contents of t (for the most part the same as **len(s); len(s) = t**)
- s \*= n** updates s with its contents repeated n times
- s.insert(i, x)** inserts x into s at the index given by i (same as **s[i:i] = [x]**)
- s.pop([i])** retrieves the item at i and removes it from s
- s.remove(x)** remove the first item from s where s[i] == x
- s.reverse()** reverses the items of s in place

\*\* see: <https://docs.python.org/3.10/library/stdtypes.html>**Escape Codes**

- \n** newline
- \t** tab
- \\** backslash
- \'** quote sgl/db
- \a** ascii bell
- \000** octal val 000
- \xhh** hex val hh
- \r** carriage return

**Keywords (reserved)**

and, as, assert, async, await, break, class, continue, def, del, elif, else, except, False, finally, for, from, global, if, import, in, is, lambda, nonlocal, None, not, or, pass, raise, return, True, try, while, with, yield

**Built-in Constants**

False, True, None, NotImplemented, Ellipsis (same as literal '...'), \_\_debug\_\_, quit(), exit(), copyright, credits, license

**Boolean Operations**

Operation / Result (ascending)

- x or y** if x is false, then y, else x
- x and y** if x is false, then x, else y
- not x** if x is false, True, else False



## The Python Standard Library

**Content:** docs.python.org/3/py-modindex.html

**Text Processing Services** - 7 modules including:

**string** - Common string operations

**re** - Regular expression operations

**textwrap** - Text wrapping and filling

**Binary Data Services** - 2 modules

**Data Types** - 13 modules including:

**datetime** - Basic date and time types

**calendar** - Calendar-related functions

**copy** - Shallow and deep copies

**enum** - Support for enumerations

**pprint** - Data pretty printer

**Numeric and Mathematical Modules** - 7 modules including:

**numbers** - Abstract base classes

**math** - Mathematical functions

**cmath** - complex #; decimal - accurate

**random** - Generate pseudo-random #s

**statistics** - Statistical functions

**fractions** - Rational numbers

**Functional Programming** - 3 modules

**File and Directory Access** - 11 modules including:

**pathlib** - Object-oriented file paths

**os.path** - Common path functions

**fileinput** - iterate lines—multiple inputs

**filecmp** - File and directory compare

**shutil** - High-level file operations

**Data Persistence** - 6 modules including:

**pickle** - Python object serialization

**marshal** - Internal Python object

serialization

**sqlite3** - DB-API 2.0 interface for SQLite

databases

**Data Compression and Archiving** - 6

modules including:

**zipfile** - Work with ZIP archives

**tarfile** - Read and write tar archive files

**File Formats** - 5 modules including:

**csv** - CSV File Reading and Writing

**Cryptographic Services** - 3 modules:

**Generic Operating System Services** - 16

modules inc:

**os** - Miscellaneous operating system

interfaces

**time** - Time access and conversions

**io** - Core tools working with streams

**platform** - Access to platform identifying

data

**Concurrent Execution** - 10 modules

including:

**threading** - Thread-based parallelism

**multiprocessing** - Process-based

parallelism

**Interprocess Communication and**

**Networking** - 9 mods

**Internet Data**

**Handling** - 10 modules:

**Structured Markup Processing Tools** -

13 modules:

**Internet Protocols and Support** - 21

modules

**Multimedia Services** - 9 modules

including:

**wave** - Read and write WAV files

**Internationalization** - 2 modules:

**Program Frameworks** - 3 modules

including:

**turtle** - Turtle graphics

**Graphical User Interfaces with Tk** - 6

modules including:

**tkinter** - Python interface to Tcl/Tk

**IDLE**

**Development Tools** - 9 modules:

**Debugging and Profiling** - 7 modules:

**Software Packaging and Distribution** - 4

modules NOTE: distutils deprecated -

Setuptools now includes it

**ensurepip** - bootstrapping pip installer

**Python Runtime Services** - 14 modules

including:

**sys** - System-specific parameters and

functions

**sysconfig** - Access to Python's config

information

**\_\_main\_\_** - Top-level script environ.

**inspect** - Inspect live objects

**Custom Python Interpreters** - 2 mods

**Importing Modules** - 5 modules including

**zipimport** - Import modules from Zip

archives

**runpy** - Locating and executing Python

modules

**Python Language Services** - 13 mods :

**keyword** - Testing for Py keywords

**py\_compile** - Compile Python source

files

**Miscellaneous Services** - 1 module:

**MS Windows Specific Services** - 4

modules

**Unix Specific Services** - 13 modules:

**Superseded Modules** - 2;

**Undocumented Modules** - 1

**pypi.org** another 257M+ modules

including: RPI.GPIO, Pillow, pandas,

fuzzywuzzy, Anaconda, miniconda, conda,

playsound, Poetry, Numpy, etc.

To find installed modules from Python:

>>> help('modules')

## Selected Standard Library Module Constants and Methods for New Users

**calendar** import calendar

a couple of fun examples:

c=calendar.TextCalendar(calendar.SUNDAY)

c.pryear(2021,w=2,l=1,c=6,m=3) or try

c=calendar.TextCalendar(calendar.MONDAY)

c.setfirstweekday(calendar.SUNDAY)

print(c.formatmonth(2021,1,w=0,l=0))

many functions - see: www.wikipython.com ->

OTHER MODULES -> calendar

**cmath** - A suite of functions for complex #

**copy** - import copy relevant for compound

objects, (objects containing other objects)

**.copy(x)** <-relies on references to objects

**.deepcopy(x[, memo])** <-copies objects (so you

can change the copy and not the original)

**csv** See Data on Disk Toolbox

**datetime** from datetime import \*

hundreds of functions and attributes

today = date.today()

**decimal** fast, correctly rounded fp math with a

gazillion functions and pages of instruction

**ensurepip** - bootstrap pip into an existing

Python environment - pip is the installer for

modules not in the Standard Library

Windows command line invocation:

python -m ensurepip -- upgrade

**enum** - from enum import enum

mimicks enum in C, fast integer access and iter.

**filecmp** import filecmp

**.cmp(f1, f2, shallow=True)** Compare f1 and f2,

returning True if they seem equal

**fileinput** import fileinput

for line in fileinput.input():

process(line)

**.input** (files=None, inplace=False,

backup="", \*, mode='r', openhook=None)

**.filename()** ↵ file being read

**.fileno()** ↵ file descriptor (-1 is none open)

**.lineno()** ↵ cumulative # of last line read

**.filelineno()** ↵ line # in current

**.isfirstline()** ↵ True if first line of its file

**.isstdin()** ↵ True if last line was read

from sys.stdin

**.nextfile()** close file, read next line from next file

**.close()** close

**fractions.py** import fractions

**.Fraction** (numerator=0, denominator=1)

**.Fraction**(other\_fraction)

**.Fraction**(float)

**.Fraction**(decimal)

**.Fraction**(string)

a= '3.03125'; print(fractions.Fraction(a))

↵ 97/32

print(fractions.Fraction(3.14159))

↵ 3537115888337719 / 1125899906842624

**idlelib** IDLE is Python's native IDE see:

<https://docs.python.org/3.10/library/idle.html>

**io** import io: three types: text, binary, raw Ex:

f = open("myfile.txt", "r", encoding="utf-8")

f = open("myfile.jpg", "rb")

f = open("myfile.jpg", "rb", buffering=0)

**json** - See Data on Disk Toolbox

**math** - import math functions include:

**.ceil(x)** smallest int >= x

**.comb(n,k)** ways to choose k items from n

**.copysign(x,y)** absolute value of x, sign of y

**.fabs(x)** ↵ absolute value of x

**.factorial(x)** ↵ x factorial as integer

**.floor(x)** ↵ largest int <= x

**.fmod(x,y)** mathematically precise ver of x%y

**.frexp(x)** ↵ mantissa and exponent of x (m,e)

**.fsum(iterable)** ↵ returns fp sum of values

**.gcd(a,b)** ↵ greatest common divisor of a & b

**.isclose(a, b, \*, rel\_tol=1e-09, abs\_tol=0.0)** True

if a & b are close, otherwise False, relative or abs

tolerance

**.isfinite(x)** ↵ True if x not infinity or a NaN

**.isinf(x)** True if x is a positive or negative infinity

**math.isnan(x)** ↵ True if x is a NaN (not a

number), False otherwise.

[new 3.8] **.isqrt(n)** ↵ the integer square root of the

nonnegative integer n. This is the floor of the exact

square root of n, or equivalently the greatest

integer such that a<sup>2</sup> ≤ n. To compute the ceiling of

the exact square root of n, a positive number, use

a = 1 + isqrt(n - 1).

**.ldexp(x, i)** ↵ x \* (2\*\*i); inverse of **frexp()**

**.modf(x)** ↵ fractional and integer parts of x

**.trunc(x)** ↵ Real value of x truncated to integral

**.exp(x)** ↵ e\*\*x.

**.expm1(x)** ↵ e\*\*x - 1

**.log(x[, base])** 1 argument, ↵ natural logarithm of

x (to base e). 2 arguments, ↵ the logarithm of x to

the given base, calculated as log(x)/log(base).

**.log1p(x)** ↵ the natural logarithm of 1+x (base e).

accurate for x near zero

**.log2(x)** ↵ the base-2 logarithm of x

**.log10(x)** ↵ base 10 log of x

**.pow(x,y)** ↵ x raised to y

**.sqrt(x)** ↵ square root of x

**Trigonometric Functions:** ↵ radians **.atan2(y,x)**

**.hypot(x,y)** ↵ sqrt(x\*\*2 + y\*\*2) **.acos(x)** **.asin(x)**

**.atan(x)** **.cos(x)** **.sin(x)** **.tan(x)**

**.degrees(x)** ↵ angle from radians to degrees

**.radians(x)** ↵ angle from degrees to radians

**math.pi** π = 3.141592... **math.e** e = 2.718281...

**math.nan** A floating-point "not a number" (NaN)

**numbers** - operations from abstract base

classes - four classes defined: Complex(components: real, imaginary), Real, Rational (adds numerator and denominator properties), Integral

**os** import os **\*\*hundreds of functions, many os specific; a few universal**  
**.environ**['HOME'] *home directory*,  
**.chdir**(path) change working dir  
**.getcwd**() current working dir  
**.listdir**(path) **.mkdir**(path) **.mkdirs**(path)  
*make all intermediate directories* **.remove**(path)  
**.strerror**() translate error code to message  
**.curdir**() **.rename**(src, dst) **.rmdir**(path)  
**.walk**(start directory, topdown=True) produces a generator of filenames in a directory tree  
**.system**(command) Unix and Windows, execute the command in a subshell

**os.path** *Lib/posixpath or Lib/ntpath (windows)*

import os.path [as osp]  
**.abspath**(path) normalized absolutized version of the pathname path.  
**.basename**(path) base name of pathname path.  
**.commonpath**(paths) longest common sub-path.  
**.commonprefix**(list) ↵ the longest prefix  
**.dirname**(path) ↵ directory name of path  
**.expandvars**(path) ↵ *environment variables expanded*  
**.exists**(path) ↵ True if path exists  
**.getsize**(path) ↵ n the size, in bytes, of path.  
**.isabs**(path) ↵ True if path is absolute pathname  
**.isfile**(path) ↵ True if path is *existing* file  
**.isdir**(path) ↵ True if path is *existing* directory  
**.islink**(path) ↵ True if ref is an *existing* directory  
**.join**(path, \*paths) Join one or more path components intelligently.  
**.normcase**(path) Normalize case of a pathname  
**.normpath**(path) On Windows, converts forward slashes / to backward slashes \.  
**.relpath**(path, start=os.curdir) ↵ relative filepath from the current directory or an optional start  
**.samefile**(path1, path2) ↵ True if both pathname arguments refer to the same file or directory.  
**.sameopenfile**(fp1, fp2) ↵ True if the same  
**.samestat**(stat1, stat2) Return True if the stat tuples stat1 and stat2 refer to the same file.  
**.split**(path) Split path into a pair, (head, tail)  
**pathlib** (3.5) from pathlib import Path [as pt]  
SEE DATA ON DISK TOOLBOX—this is now THE critical file access module

**pickle** import pickle - non-human-readable  
See Data on Disk Toolbox

**platform** import platform  
**.machine**() ↵ *machine type*  
**.node**() ↵ *network name*  
**.processor**() ↵ *real processor name*  
**.python\_version** ↵ *version as string*  
**.system**() ↵ 'Linux', 'Darwin', 'Java', 'Windows'

**pprint** import pprint  
allows output of objects, including objects holding other objects in a reasonably readable format.  
Begin by creating an instance: (assume "mylist")  
pp = pprint.PrettyPrinter(indent=3) *set indent*  
then use your instance ("pp" above) to output:  
pp.pprint(mylist)  
some PrettyPrinter objects new/changed in [3.8]  
.pformat(obj), .pprint(obj), pp.isreadable(obj), more  
ex: print(pp.isreadable(mylist))

**py\_compile.py** import py\_compile  
**.compile**(file) - the compiled file is placed on file path in added directory *"/\_pycache\_"*

**random** import random  
*only for non-cryptographic applications*  
**.seed** initialize the random number generator  
**.getstate**() ret object with internal generator state  
**.setstate**() restores internal state to getstate value  
**.getrandbits**(k) ret integer with k random bits  
For integers:  
**.randrange**(stop) **.randrange**(start, stop[, step])  
**.randint**(a, b) fileinput.filename() a random

integer N such that a <= N <= b. Alias for randrange(a, b+1).

For sequences:

**.choice**(sequence) ↵ random element  
**.random**() ↵ the next random floating point number in the range (0.0, 1.0).  
**.uniform**(a, b) ↵ fp between a and b  
**re** import re complex search and match  
**re.search**(pattern, string, flags=0)  
**re.match**(pattern, string, flags=0)  
**re.ignorecase**  
**shutil** import shutil  
**.copyfileobj**(fsrc, fdst[, length])  
**.copyfile**(src, dst, \*, follow\_symlinks=True)  
**.copymode**(src, dst, \*, follow\_symlinks=True)  
Copy the permission bits from src to dst.  
**.copystat**(src, dst, \*, follow\_symlinks=True)  
Copy the permission bits, last access time, last modification time, and flags from src to dst  
**.copy**(src, dst, \*, follow\_symlinks=True)  
Copies the file src to the file or directory dst. src and dst should be strings.  
**.copy2**(src, dst, \*, follow\_symlinks=True)  
copy2() also attempts to preserve file metadata  
**.copytree**(src, dst, symlinks=False, ignore=None, copy\_function=copy2, ignore\_dangling\_symlinks=False, dirs\_exist\_ok=False)  
**.disk\_usage**(path) ↵ disk usage stats as tuple (total, used and free) in bytes—a file or a directory  
**Sound** if your objective is to play a sound using a Python Standard Library module save your time - **none** of the modules listed under Multimedia Services do that. SEE: PyPi — **playsound**  
**sqlite3** See Data on Disk Toolbox  
**statistics** import statistics  
**.mean**(data) average  
**.harmonic\_mean**(data) harmonic mean  
**.median**(data) middle value  
**.median\_low**(data) low middle value  
**.median\_high**(data) high middle value  
**.median\_grouped**(data) 50th percentile  
**.mode**(data) most common  
**.pstdev**(data, mu=None) population std dev  
**.pvariance**(data, mu=None) pop variance  
**.stdev**(data, xbar=None) sample std dev  
**.variance**(data, xbar=None) sample variance  
more...extensive normal distribution functions

## string

**Constants**  
string.ascii\_letters,  
string.ascii\_lowercase string.ascii\_uppercase  
string.digits string.hexdigits  
string.octdigits string.punctuation  
string.printable string.whitespace  
string.capwords(str, sep=None)

**sys** import sys mostly advanced functions  
**.exit**([arg]) - exit python  
**.getwindowsversion**()  
**.path** - search paths list  
**.version** - Python version #

**tarfile** import tarfile extensive archive including gzip, bz2 and lzma compression  
ex: (assumes import tarfile - to extract to cwd)  
tar = tarfile.open("sample.tar.gz")  
tar.extractall()  
tar.close()

**textwrap** import textwrap  
textwrap.wrap(text, width=x, \*\*kwargs) **Lib/Lib/**  
**time** import time or from time import  
a new user must understand terminology found at:  
<https://docs.python.org/3.8/library/time.html>  
print(time.time()) *#seconds since the epoch*  
↵ 1596486146.11275  
mytime = time.time() *#capture it*  
print(time.localtime(mytime)) *#demo the tuple*  
↵ time.struct\_time(tm\_year=2020,  
tm\_mon=8, tm\_mday=3, tm\_hour=16, tm\_min=22,

tm\_sec=26, tm\_wday=0, tm\_yday=216,  
tm\_isdst=1)  
time\_tuple=time.localtime(mytime) *#capture it*  
print("The hour is: " + str(time\_tuple[3])) *#demo*  
↵ The hour is: 16  
print(time.strftime("%a, %d %b %Y %H:%M:%S +0000", time.localtime()))  
↵ Mon, 03 Aug 2020 20:22:26 +0000  
seconds=5 ; print("Wait 5 seconds!")  
time.sleep(seconds) *# delay of five seconds*  
print(time.asctime(time.localtime()))  
↵ Mon Aug 3 16:22:31 2020  
print(timectime(mytime))  
↵ Mon Aug 3 16:22:26 2020

**tkinter** from tkinter import \* a 16 page  
tkinter toolbox is available for review at  
[www.wikipython.com](http://www.wikipython.com)—free download on GitHub  
**A Few PyPi Modules** <https://pypi.org>  
**Anaconda, Conda, MiniConda** - 3 related  
programs offering environment management at  
different levels. **Anaconda** manages all variations  
and compatibility issues unavoidable with many  
modules. Over 300 applications come "installed" in  
the base (root) environment, with thousands  
available. **Installation(s) can be huge.** It qualifies  
as a language within itself. Numerous IDEs are  
available in any Anaconda environment including  
**Spyder, Visual Studio Code, IDLE, Jupyter**  
**Notebooks ... more.** **Miniconda** is a lightweight  
version. **Conda** is similar to pip but is also an  
environment manager.

**NumPy** - powerful **N-dimension array** objects  
NumPy says installation works best with a prebuilt  
package, see: <https://scipy.org/install.html> where  
they suggest a "scientific distribution" but do give  
**pip** directions

**Rpi.GPIO** - module to control Raspberry Pi  
GPIO channels; see GPIO toolbox and download  
link at: [www.wikipython.com](http://www.wikipython.com)

**Pillow** - by Alex Clark, updated Aug 2020, a friendly  
version of Fredrik Lundh's **Python Imaging Library**  
Pillow version 7.2 works in Python 3.5 to 3.8  
install: **python3 -m pip install --upgrade Pillow**  
from PIL import Image  
im = Image.open(testfilepath)  
print(im.format, im.size, im.mode)  
im.show()

**playsound** is a cross platform program pulled  
from **PyPi** that is very easy to use. From windows:  
python -m pip install playsound for example:  
from playsound import playsound  
testwave = "C:\\Windows\\Media\\Alarm09.wav"  
**playsound**(testwave)

**pandas** for tabular data — "aims to be the funda-  
mental" module for "real world data analysis" - it is  
part of the Anaconda distribution (also installs with  
Miniconda) but can be installed with pip:

**pip install pandas** **plotly.express** and  
**Kaleido** - **plotly.express** is built-in to the **plotly**  
library and is considered a "starting point" but may  
be all you ever need. *Plotly is an MIT Licensed*  
*module.* **plotly.express** requires a determined effort  
to learn because it creates more than 35 types of  
graph images. It does **not** export your graph as a  
static image—which is why you need **Kaleido**.  
**plotly** has many dependencies, **kaleido** has none.  
**pip install kaleido.**

## Module Management

**import** get module, ex: import math or  
**from module import \***  
**from** get a single module function: from  
math import cos; print(cos(9))  
**as** creates an alias for a function  
**What is NOT mentioned in this General Toolbox?**  
About 99.83% of Python capability now available  
has no mention in this toolbox. Happy Coding!