|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Datatypes** | Casting | | **Sample** | **Description** |  |
| Complex() | | 3+5.6j | Number made of a <.Real()>+<.Imaginary()>j parts | **m**utable |
| int() | | 17 | whole number negative or positive | **m**utable |
| float() | | 2.8 | Number negative or positive | **m**utable |
| bool() | | True | True=1 & False =0 when used in sums | **m**utable |
| N/A | | **None** |  | **i**mmutable |
| Iterables | str() | "name" | A Iterableof characters, order is preserved | **i**mmutable |
| list() | [‘a’,1,1.2] | A Iterableof values characters, order is preserved | **m**utable |
| tuple() | (‘a’,1,1.2) | A Iterableof values characters, order is preserved | **i**mmutable |
| set() | {‘a’,1,1.2} | A Iterableof **unique** values | **m**utable |
| frozenset() | {‘a’,1,1.2} | A **immutable** Iterableof **unique** values | **i**mmutable |
| dict() | {k:v, k1:v1} | A table of unique immutable keys & values | **m**utable |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ***Mathematics Operation*** |  | **Name** | | | **Description** | | | | | |
| a\*\*b | ab | | | a//b : a to the power b | | | | | |
| a//b | Floor division | | | The largest integer that is less than or equal to the result of the division a÷b | | | | | |
| a%b | Modulo | | | The remainder of a division | | | | | |
| Division | | a/b | Add | | a+b | Subtract | a-b | Repeat “str”/[list] n times | “Str”\*2 #Returns ”StrStr” |
| *Mathematics Comparison* | | | | | | | | | |
| < | *is l*ess than | | | Returns True if the value on the left is smaller the value on the right;*else False*. | | | | | |
| > | is greater than | | | Returns True if the value on the left is bigger the value on the right; else False. | | | | | |
| == | is equal | | | Returns True if the value on the left is equal the value on the right; else False. | | | | | |
| *!=* | Is Not equal | | | Returns True if the value on the left is Not equal the value on the right, otherwise Returns False. | | | | | |
| *>=* | Is Greater than **or** Equal | | | Returns True if the value on the left is Greater than **or** Equal the value on the right, otherwise Returns False. | | | | | |
| *<=* | Is *l*ess than **or** Equal | | | Returns True if the value on the left is *l*ess than **or** Equal the value on the right, otherwise Returns False. | | | | | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **>> Order of Operations Mathematics >>** | | | | | | | | | | | | | | |
| Parentheses | () | Exponentiation | | \*\* | *Multiplication* | \* | *Division* | / | // | % | *Addition* | + | *Subtraction* | - |
| ***P.E.M.D.A.S*** | | **<<** R to L **<<** | | | **Share a level** >> L to R >> | | | | | | **Share a level** >> L to R >> | | | |
| If two adjacent Operations Share a level of president (ie, - & +)Operations will be done from left to right | | | | | | | | | | | | | | |
| Note 20/2\*2 will be (20/2)\*2 | | | **Note:** Arithmetic operators take precedence over logical operators | | | | | | | | | | | |

|  |  |
| --- | --- |
| **Indexing[]** | **Slicing[::]** |
| ***iterable[index]***  Returns a value (or **e**lement) as index. | ***iterable[start:stop:step]***  Returns values for the start index up to but not up to but not including stop index, in increments of step. |
| my\_list=[‘a’,’b’,’c’]  my\_list[0]= ‘a’ | my\_list=[‘a’,’b’,’c’,’d’,’e’]  my\_list[1:3]= [’b’,’c’,’d’] |
| #negative indexing will also work  my\_list=[‘a’,’b’,’c’]  my\_list [-1]= ’c’ | #negative indexing & negative step will also work  my\_list=[‘a’,’b’,’c’,’d’,’e’]  my\_list[1:3]= [’b’,’c’,’d’] |
| **Note:** Indexing and Slicing works on Works: **str(); list(); tuple()** | |
| **Note:** Indexing for zero is used so the first value (or **e**lement) is at index 0 | |

|  |  |  |
| --- | --- | --- |
| **Functions() vs .Methods()** | | **[Mutable] vs (Immutable)** |
| Functions and methods very simmer similar, both can Returns values/objects, both can potential accept arguments/args, both when called end with open closed Parentheses*()*.  ***Where they differ is how we call them.*** | | Mutable data types are those whose values can be modified after they are created, while immutable data types are those whose values cannot be modified once created. When we try to change an immutable data type we overweight it give the data types a new “Id” the id of a value can be viewed using the id() |
| **Functions()** | **.Methods()** | **Interpreter VS Compiler** |
| Name of the arg  functions(arg) | Obj.mehtod()  my\_list.appent(args) | A compiler translates the entire source code into machine code before execution, resulting in faster execution & An interpreter translates code line by line during execution, making it easier to detect errors but potentially slowing down the program |
| **Note:** we can change *Method calls like so* Obj.mehtod().mehtod().mehtod() | |

|  |  |  |
| --- | --- | --- |
| **Built-ins *Functions()*** | ***Functions*** | **Description** |
| help(str) | ***help(“global”)****:* Returns the help page on Of a module, function or class, |
| type(obj) | ***type(variable\_name):*** Returns the Datatypes of the variable |
| id(obj) | ***id(variable\_name):*** This will return the unique id for thevariable |
| input(str) | **input(“optional user prompt message”)** : Return a **string** that user has type in |
| len(obj) | **len(Immutable\_object):** Returns the number of **e**lement in the Immutable\_object |
| range() | **range(*start:stop:step*):** often used with for loop, Returns Immutable for numbers |
| enumerate() | **enumerate(iterable, start):** often used with for loop, Returns index & **e**lement |
| eval(str) | **eval(“print(“Hi”)”):** runs the string as a line of python |
| sum(obj) | Returns the sum of elements in of an iterable object |
| all(obj) | Returns True if all items in an obj are true, otherwise it returns False. |
| any(obj) | Returns True if any items in an obj are true, otherwise it returns False. |
| zip(\*obj) | Returns a of tuples where the first item in each passed iterator is paired together |
| map(**f**un,obj) | Returns a list of the results after applying the given **f**unction to each item of a given obj |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Variable scope** | | **Declaring variable** | | **Some Key word operators** | |
| x =1 ; y=2 # global  def myfunc():  **global** x  x = "fantastic";  y=9 #local | All variable Declared in a *Functions are Local, unless the* **global** key word is used to refer to a variable in the global scope**.** Variable outside a *Functions will be* global | x =1 | On a Single line, | **not** | Not(True) #-> False |
| x , y = 1, 2 | By unpacking (see trupls) | **and** | true if both values are true ;*else False*. |
| x =1 ; y=2 | Or be ending statement with ; | **or** | true if one value is true; *else False*. |
| **Note:** when declaring **String** you may use “ ” or ‘ ’.  **Note** : “”” “”” will keep line breaks & tabs | | **in** | true if the value is in the collection |

.

.

.

|  |  |  |
| --- | --- | --- |
| ***“Strings”*** | **Methods** | **Returns Description** |
| obj.split() | **obj.split(string argument):** Returns a list to the parts of a string split on the string argument give. If no string argument it will split on a white space “ ” |
| obj.find() | **obj.find(string argument):** Returns the index of the string argument given, if the string argument is not found it Returns -1 |
| obj.index() | **obj.index(string argument):** Returns the index of the string argument given, if the string argument is not found it Returns “ValueError” |
| obj.replace() | **obj.replace(old\_string\_argument, new\_string\_argument):** Returns an string with all occasion of the old\_string\_argument replaced with the new\_string\_argument. |
| obj.strip() | **obj.strip():**Returns an string with all preceding & trailing white spaces removed |
| obj.rstrip() | **obj.strip():**Returns an string with all trailing white spaces removed |
| obj.lstrip() | **obj.strip():**Returns an string with all preceding white spaces removed |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Scape characters** | New Line | \n | Carriage Return | \r | Tab | \t | Backspace | \b | Escape characters | \’ | \” |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **F-{str}s** | **Datatypes** | Int | d | Float | f | String | S | #will Returns a formatted string  ***f”{value :Fill, Align,* Min\_with*, Precision, Datatypes}”***  num1= 1; num2 =1.265; name= "James";  print(f"{num1:d} {num2:.2f} {name:\*>10s} ")) |
| **Align** | Left | < | Right | > | Center | ^ |
| **Precision** | .2f will round a Float to two decimal places. | | | | | |
| **Min\_with** | Will pad a values to this with. | | | | | |
| **Fill** | Sets padding character | | | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| ***[Lists]*** | **Methods** | **Description** | **Sample** |
| .append() | Adds an element at the end of the list | My\_list = [1,2,3] |
| .clear() | Removes all the elements from the list |
| .copy() | Returns a copy of the list |
| .count(**e**) | Returns the number of **e**lements with the specified value |
| .extend(**obj**) | Add the elements of **obj** (or any iterable), to the end of the current list. |
| .index(**e**) | Returns the index of the first **e**lement with the specified value |
| .insert(**i, e**) | Adds an **e**lement at the specified **i**ndex |
| .pop(**i**) | Removes the element at **i**ndex |
| .remove(**e**) | Removes the **e**lements |
| .reverse() | Reverses the order of the list |
| .sort() | Sorts the list in ascending order |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ***(Tuples)*** | Packing | | Unpacking | | **Sample** |
| *tup = (1,2,3)*  *x,y,z = tup*  *print(x,y,z) #1,2,3* | | *x=1; y=2; z=3*  *tup = x,y,z*  *print(tup) #(1,2,3)* | **#Unpacking args**  *tup= (32,1,7)*  *print(\*tup,sep=* *'\n* *')* | *my\_tupls = (1,2,3)* |
| **Methods** | **Returns** | | |
| .count(e) | Returns the number of times a specified **e**lementoccurs in a tuple | | |
| .index(e) | Searches the tuple for a specified **e**lement and returns the position of where it was found | | |

|  |  |  |  |
| --- | --- | --- | --- |
| ***Dictionaries {K:V}*** | **Methods** | **Returns** | **Sample** |
| .get(**k**ey) | Returns a value for a given key | my\_dict= {"a":1,"b":2,"c":3}  print(my\_dict.pop("a"))  # Returns ‘a’  print(my\_dict.pop("a"))  # Returns ‘a’ |
| .keys() | Returns all *dict\_keys* for a Dictionary |
| .values() | Returns all *dict\_ values* for a Dictionary |
| .items() | Returns all *dict\_items* for a Dictionary |
| .update({**K:V**}) | Adds new item/s from {**k**ey**:V**alues} |
| .popitem() | Returns a tuple containing the {key:values} |
| .pop(**k**ey) | Returns a value for a given key & remove the item from the Dictionary |
| del dict[e] | Remove **e**lements from Dictionary |
| .copy() | Returns Dictionary: copy of **obj** with new id |
| .clear() | Remove all item in Dictionary |
| .fromkeys() | returns a dictionary with the specified keys and the specified value. |
| .setdefault() | method returns the value of the item with the specified key.If the key does not exist, insert the key, with the specified value, |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **{*Sets}*** | | | | | | | | **{frozenset Sets}** | | | |
| my\_set = {"apple", "banana", "cherry"}  print(my\_set)  print(my\_set[1])  my\_set[1]= "avocado" | | | | | | | | frozenset as immutable version of a set as such only operations that do not change the objet will work, *ie: Union, intersection, difference,& so on.* | | | |
| **Methods** | | **Returns ->** | **Description** | | | | | | | | |
| .add() | | None | **obj.add(new\_element):** adds new\_**e**lement to Set\_obj | | | | | | | | |
| .clear() | | None | **obj.clear():** remove all **e**lements in Set | | | | | | | | |
| .remove(**e**) | | None | **obj.remove(element):** remove fist instances of give **e**lements in Set\_obj | | | | | | | | |
| .pop(**i**) | | **e**lement | **obj.pop(index):**remove **e**lements at give **i**ndex in Set\_obj | | | | | | | | |
| .discard(**e**) | | None | **obj.remove(element):** remove fist instances of give **e**lements Will not throw an error if **e**lement is not found | | | | | | | | |
| **op** | **Functions** | | | **Returns** | **Sample** | | | | | | |
| | | union() | | | set | **a.union(b):**Returns a set containing all item in boat sets | | | | | | |
| |= | update() | | | None | **a.update(b):** updates a sets adding item in **b** | | | | | | |
| & | intersection() | | | set | **a.intersection(b):** Returns set a containing all shared item boat sets. | | | | | | |
| ^ | symmetric\_difference() | | | set | **a.symmetric\_difference(b):**Returns set containing all **not** shared boat. | | | | | | |
| - | difference() | | | set | **a.difference(b):** Returns set a containing all item a but not in **b** set. | | | | | | |
| *Comparison* operators | | | Is a subset | | > | Returns Bool | Is a superset | | > | Returns Bool |  |

|  |  |
| --- | --- |
| ***Command Line Arguments*** | ***PY to.EXE(CLI) Steps*** |
| import sys; print(str(sys.argv[:]))  1.Open CMD and enter: ***python file\_name.py 1 1.5 test*** | 1.Open CMD and enter: *pip install pyinstaller*  ***pyinstaller --onefile* file\_name*.py*** |

.

.

|  |  |
| --- | --- |
| **Conditional Statement** | |
| a,b = 1,2; tup=(2,2,2); fruits = ["apple", "banana", "cherry"] | |
| if b > a:  print(b ,”is greater than” a)  elif a < b:  print("a is greater than b")  #....    else:  print("a and b are equal") | if b in tup:  print("b is in tup") |
| #[on\_true] if [expression] else [on\_false]  bigger a if a > b: else b  print(bigger) |
|  |
| **Loop while** |  |
| count= 1  while count < 100:  print(count)  count += 1 | Continue :: |
| Break :: |
| Pass:: |
| **For Loop (foreach)** | |
| for e in fruits:  print(e) | for e in range(2,10,2):  print(e) |
| #For Loop in a Single Line  [print(e) for e in fruits] |  |

|  |  |
| --- | --- |
| **Exception Handling** | **python file handling** |
| try:  print(missing)    except FileNotFoundError as e:  print("something went wrong")  except Exception as e:  print("something went wrong")  raise | with open({credentials\_file\_path}", "r") as f:  for x in f:  credentials+=x  f= open({credentials\_file\_path}, "r") |

|  |
| --- |
| **Functions** |
| #  def add\_function(arg\_1, arg\_2, defaultPar =4) -> str:  """Doc string"""  return (f"{arg\_1} + {arg\_2} + {defaultPar}")  print(add\_function(1, 2))  print(add\_function(1, 2, 2))  print(add\_function(arg\_2=1, arg\_1=2)) |
| def my\_function(\*kids)->str:  ”””Doc string ”””  return "The youngest child is " + kids[2]  my\_function("Emil", "Tobias", "Linus") |
| #  def my\_function(\*\*kid)->int:  ”””Doc string ”””  return "His last name is " + kid["lname"])  my\_function(fname = "Tobias", lname = "Refsnes") |
| “->” denotes return type |
| “\*” |
| “\*\*” |
| “defaultPar=” |

|  |
| --- |
| Lambda **Functions** |
| # lambda arguments : expression  x = lambda a : a + 10  print(x(5)) |

**Recursions & Memoization**

|  |
| --- |
| Class |
| class User(inheritance):  """ Bunnys Rock !!!!"""  def \_\_init\_\_(self, full\_name,birthday):  pass  user1=User("212","122")  user1.first\_name="Dave"  print(user1.first\_name) |

|  |
| --- |
| **python file handling** |
| with open(file\_path}, "MODES") as f:  for x in f:  credentials+=x  f= open(f"{credentials\_file\_path}\\{file\_name}.csv", "r") |
| MODES |
| w(/x) write /(exclusive write)  w+(/x+) write +read/(exclusive write+read)  r read  r+ read+write  a append  a+ append+write |
| DATA types |
| t text  b binary |