|  |
| --- |
| Illinois Python Cheat Sheet  by Elizabeth de Sa e Silva, Tamara Nelson-Fromm, Wade Fagen-Ulmschneider |



|  |  |
| --- | --- |
| **Basic Data Types** | |
| **Integers** are whole numbers  int1 = 8 int2 = -5  int3 = 0 int4 = int(4.0) | **Floats** have a decimal point  float1 = 5.5 float2 = 0.0  float3 = 1e6 float4 = float(2) |
| **Strings**  A **string literal** has quotes: ‘CS101’, ‘CS107’, ‘5.67’  (it’s *literally* the exact characters of the string)  A variable name does not: course\_name, stat107, my\_string  A string can be indexed the same way as a list  **Example**  my\_string = ‘literal’ #’literal’ is the literal  print(‘my\_string’) #prints “my\_string”  Warningprint(my\_string) #prints “literal”  print(literal) #ERROR | |

|  |  |
| --- | --- |
| **Booleans** | |
| **Booleans** are **True** or **False** values  x == y Is True if x is equal to y x **in** y is True if x is an element of y  **not** x == y Is True is x is not equal to y | |
| **And**  True **and** True = True  True **and** False = False  False **and** False = False | **Or**  True **or** True = True  True **or** False = True  False **or** False = False |

|  |  |
| --- | --- |
| **Slicing** | |
| **Strings, lists, and other iterable data types** (data with many elements) **can be indexed over a range of values, or sliced**  Replace any [i] with a range to select many elements at once:  [start:stop:step]  Selects position start through position stop, not including stop, but only  elements step positions apart;  start defaults to zero, so [ :10:7 ] starts at 0  stop defaults to one past the last index, so [ 10: :2 ] selects through the end of the data  step defaults to one, so [ 1:5 ] steps by 1 (a negative step will count backwards) | |
| **Examples**  my\_string = ‘abcdefghijk’  my\_string[2:4] == ‘cd’  my\_string[:5] == ‘abcde’ | my\_string[5:] == ‘fghijk’  my\_string[:] == ‘abcdefghijk’  my\_string[2:8:2] == ‘ceg’  my\_string[8:2:-2] == ‘ige’ |

|  |  |
| --- | --- |
| **Lists** | |
| **Creating a new list**  empty\_list = []  my\_list =[1,2,3] | **Adding to a list (appending)**  list\_name.append(v) #adds just the  #element v to  #list\_name  list\_name += [v1,v2] #adds v1 and v2  #to the end of  #list\_name |
| **Indexing**  list[i] is equal to the element in list at zero-based index i  Negative index values count from the end of the data  list[-i] is equal to  list[ len(list) - i ] |
| **Changing a list**  #changes the element  list[i] = v #in list at position  #i to the value v |
| **Example**  my\_list = [10,20,30] #my\_list is declared as [10,20,30]  my\_list.append(40) #my\_list becomes [10,20,30,40]  my\_list += [50,60] #my\_list becomes [10,20,30,40,50,60]  my\_list[2] == 30 # True  my\_list[4] = “fifty” #my\_list becomes [1,2,3,4,”fifty”,60]  Warningmy\_list[-1] == “fifty” # True  my\_list[60] #ERROR | |

|  |  |
| --- | --- |
| **Dictionaries** | |
| **Creating a new dictionary**  my\_dict = {key1:value1, key2:value2, …, keyn:valuen}  empty\_dict = {} #keys and values can be **any** data type | |
| **Adding to a dictionary (appending)**  dict\_name[key] = value  #adds key:value to dict\_name | **Indexing**  dict[key] is equal to the value in dict with key key |
| **Changing a dictionary**  dict\_name[key] = value #changes key’s value to v so dict\_name  # now has the pair key:v | |
| **Getting Keys and Values**  dict\_name.keys() #returns a list of keys in dict\_name  dict\_name.values() #returns a list of values in dict\_name | |
| **Example**  my\_dict = {‘a’:5, ‘b’:6} #my\_dict is declared as {‘a’:5,’b’:6}  my\_dict[‘c’] = ‘4’ #my\_dict becomes {‘a’:5, 6:’b’, ‘c’:’4’}  my\_dict[‘a’] == 5 # True  Warningmy\_dict[‘b’] = ‘a’ #my\_dict becomes {‘a’:5,‘b’:’a’,‘c’:’4’}  my\_dict[5] #ERROR  my\_dict.keys() #equal to [‘a’, ‘b’, ‘c’] | |

|  |
| --- |
| **If Statements** |
| **if**  Indicates a block of code that only runs if its boolean condition is True  **elif**  Short for “else if”, this block is associated with an if block and has a condition; it only runs  if its condition is true and the original if block condition was false  **else**  This block has no condition and runs only if the associated if statement and any of its  elif blocks did **not** run |
| **Example**  if x < 5:  #this indented code only runs if x is less than 5  elif x < 10:  #this only runs if x is greater than 5 and less than 10  elif x == 13:  #this only runs if x is equal to 13  else:  #this only runs if x is greater than 10 and is not 13 |

|  |  |
| --- | --- |
| **Accumulator Patterns** | |
| **Example:** Sum  Suppose I have a list of weights of some packages and I want to know how heavy it will be to carry all of them at once  package\_weights = [2, 6.5, 1, 10]  total = 0  for weight in package\_weights:  total += weight  print(total) #after this code runs #the total weight is printed | **Example:** List  Suppose I want to make a list of the squares of the integers 1 through 5  squares = []  for i in range(1,5):  squares.append(i\*\*2)  #after this code runs  #squares = [1, 4, 9, 16, 25] |
| **Example:** Pandas  Suppose I want to simulate flipping a coin 50 times and put the data into a dataframe  data = []  for i in range(50):  coin = randint(0,1) #simulate one coin flip as 0 or 1  d = {‘coin’ : coin} #create the row of data  data.append(d)  df = pandas.DataFrame(data) #creates a dataframe from data | |

|  |  |
| --- | --- |
| **Functions** | |
| **def** func(input1, input2, ... inputn = defaultn):  #code block that only runs when you call func()  #if inputn is not specified it is automatically set to default  **return** my\_answer #some functions **don’t** return anything! | |
| def f(x):  return x\*\*2  Warningy = f(3) #sets y = 9  z = f(x) #ERROR  Warning #x only exists inside f  a = f() #ERROR | def g(x=’World’):  print(‘Hello ‘ + x)  g() #prints ‘Hello World’  g(‘You’)#prints ‘Hello You’  Warninga= g() #a is NaN as g returns nothing  g(‘World’, ‘Us’) #ERROR |

|  |  |
| --- | --- |
| **For Loops** | |
| **for** i **in** iterable:  #code block to repeat  Repeats a block of code for every element of an iterable data type  Does **not** require you to advance the variable i | |
| **Example**: List  list = [‘CS101’,‘CS107’,‘ILL’]  **for** item **in** list:  #loops over every element  #of list  print(item)  This code prints:  CS101  CS107  ILL | **Example:** Range  **for** i **in** range(2,8,2):  #loops over every other  #integer starting at 2  #and less than 8  print(i \*\* 2)  This code prints:  4  16  36 |
| **range**(start, stop, step)  Generates a list of all integers from start to stop, jumping by step  start  The very first integer of the sequence. This defaults to 0 if not specified  stop  The boundary for the end of the sequence. This number is **not** included in the actual  sequence of number. Has no default value and must always be specified.  step  The spacing between numbers included in the sequence. This defaults to 1 | |

|  |  |
| --- | --- |
| **While Loops** | |
| **while** some\_condition\_is\_true:  #code block to repeat  Repeats a block of code while some condition is true  Often requires you to change the variables the condition relies on in the code block to get the loop to ever stop | |
| **Example:** Factorial  #This code calculates 5!  n = 5  result = 1  while n > 0:  result = result \* n  n = n - 1 | **Example:** User Input  #This code loops until the user  #inputs an integer greater than 5  a = ‘0’  while int(a) <= 5:  a = input(‘enter a number > 5’) |
| WarningWarning **Warning:** Infinite Loops  If some\_condition\_is\_true is never false then the code will never stop running!  So, if some\_condition\_is\_true is n>0 then I need to include a line where n decreases! | |