SKILL<S/ASH>

PYTHON CHEATSHEET

PART 1

SWIPE

SKILL<S/ASH>

Keywords

Keyword	Description	Code Example
False, True	Data values from the data type Boolean	**False** == (1 > 2), **True** == (2 > 1)
And, Or, Not	Logical operators: (x **and** y) → both x and y must be True (x **or** y) → either x or y must be True (**not** x) → x must be false	x, y = True, False (x **or** y) == True # True (x **and** y) == False # True (**not** y) == True # True

Break	Ends loop prematurely	**while**(**True**):
		break # no infinite loop
		print("hello world")

Continue	Finishes current loop iteration	**while**(**True**): **continue** print("43") # dead code
Class Def	Defines a new class → a real-world concept (object oriented programming) Defines a new function or class method. For latter, first parameter ("self") points to the class object. When calling class method, first parameter is implicit.	**class** Beer: **def** __init__(self): self.content = 1.0 **def** drink(self): self.content = 0.0 becks = Beer() # constructor - create class becks.drink() # beer empty: b.content == 0
In	Checks whether element is in sequence	42 **in** [2, 39, 42] # True

Keyword	Description	Code Example
If, Elif, Else	Conditional Program Execution: Program Starts With “If” Branch, Tries The “Elif” Branch, And Finishes With “Else” Branch (Until One Branch Evaluates To True).	X = Int(Input("Your Value: ")) **If** X > 3: Print("Big") **Elif** X == 3: Print("Medium") **Else**: Print("Small")

For, While	# For Loop Declaration
	For i **In** [0,1,2]:
	Print(I)
		# While Loop - Same Semantics
		J = 0
		While J < 3:
		Print(J)
		J = J + 1

Is	Checks Whether Both Elements Point To The Same Object	Y = X = 3 X **Is** Y # True [3] **Is** [3] # False
None	Empty Value Constant	**Def** F(): X = 2 F() **Is None** # True
Lambda	Function With No Name (Anonymous Function)	(**Lambda** X: X + 3)(3) # Returns 6
Return	Terminates Execution Of The Function And Passes The Flow Of Execution To The Caller. An Optional Value After The Return Keyword Specifies The Function Result.	**Def** Incrementor(X): **Return** X + 1 Incrementor(4) # Returns 5

Basic Data Types

Keyword	Description	Code Example
Boolean	The Boolean Data Type Is A Truth Value, Either **True** Or **False**. The Boolean Operators Ordered By Priority: **Not** X → “If X Is False, Then X, Else Y” X **And** Y → “If X Is False, Then X, Else Y” X **Or** Y → “If X Is False, Then Y, Else X” These Comparison Operators Evaluate To True: 1 < 2 **And** 0 <= 1 **And** 3 > 2 **And** 2 >=2 **And** 1 == 1 **And** 1 != 0 # True	## 1. Boolean Operations X, Y = **True**, **False** Print(X **And Not** Y) # True Print(**Not** X **And** Y **Or** X) # True ## 2. If Condition Evaluates To False **If None Or** 0 **Or** 0.0 **Or** '' **Or** [] **Or** {} **Or** Set(): # None, 0, 0.0, Empty Strings, Or Empty # Container Types Are Evaluated To False Print("Dead Code") # Not Reached

Integer, Float	An Integer Is A Positive Or Negative Number
	Without Floating Point (E.G. **3**). A Float Is A Positive Or Negative Number With Floating Point
	Precision (E.G. **3.14159265359**).
	
	The ‘//’ Operator Performs Integer Division. The Result Is An Integer Value That Is Rounded
	Towards The Smaller Integer Number (E.G. **3** // **2** == **1**).
		## 3. Arithmetic Operations
		X, Y = 3, 2
		Print(X + Y) # = 5
		Print(X - Y) # = 1
		Print(X * Y) # = 6
		Print(X / Y) # = 1.5
		Print(X // Y) # = 1
		Print(X % Y) # = 1s
		Print(-X) # = -3
		Print(Abs(-X)) # = 3
		Print(Int(3.9)) # = 3
		Print(Float(3)) # = 3.0
		Print(X ** Y) # = 9

Keyword	Description	Code Example

String	Python Strings Are Sequences Of Characters.
	The Four Main Ways To Create Strings Are The
	Following.
	1. Single Quotes
	'Yes'
	2. Double Quotes
	"Yes"
	3. Triple Quotes (Multi-Line)
	**"""Yes
	We Can"""**
	4. String Method
	Str(5) == **'5'** # True
	5. Concatenation
	"Ma" + **"Hatma"** # 'Mahatma'
	
	These Are Whitespace Characters In Strings.
	• Newline \N
	• Space \S
	• Tab \T
		## 4. Indexing And Slicing
		S = "The Youngest Pope Was 11 Years Old"
		Print(S[0]) # 'T'
		Print(S[1:3]) # 'he'
		Print(S[-3:-1]) # 'ol'
		Print(S[-3:]) # 'old'
		X = S.Split() # Creates String Array Of Words
		Print(X[-3] + " " + X[-1] + " " + X[2] + "S")
		# '11 Old Popes'
		
		## 5. Most Important String Methods
		Y = " This Is Lazy\T\N "
		Print(Y.Strip()) # Remove Whitespace: 'This Is Lazy'
		Print("DrDre".Lower()) # Lowercase: 'drdre'
		Print("Attention".Upper()) # Uppercase: 'ATTENTION'
		Print("Smartphone".Startswith("Smart")) # True
		Print("Smartphone".Endswith("Phone")) # True
		Print("Another".Find("Other")) # Match Index: 2
		Print("Cheat".Replace("Ch", "M")) # 'meat'
		Print(','.Join(["F", "B", "I"])) # 'F,B,I'
		Print(Len("Rumpelstiltskin")) # String Length: 15
		Print("Ear" In "Earth") # Contains: True

Complex Data Types

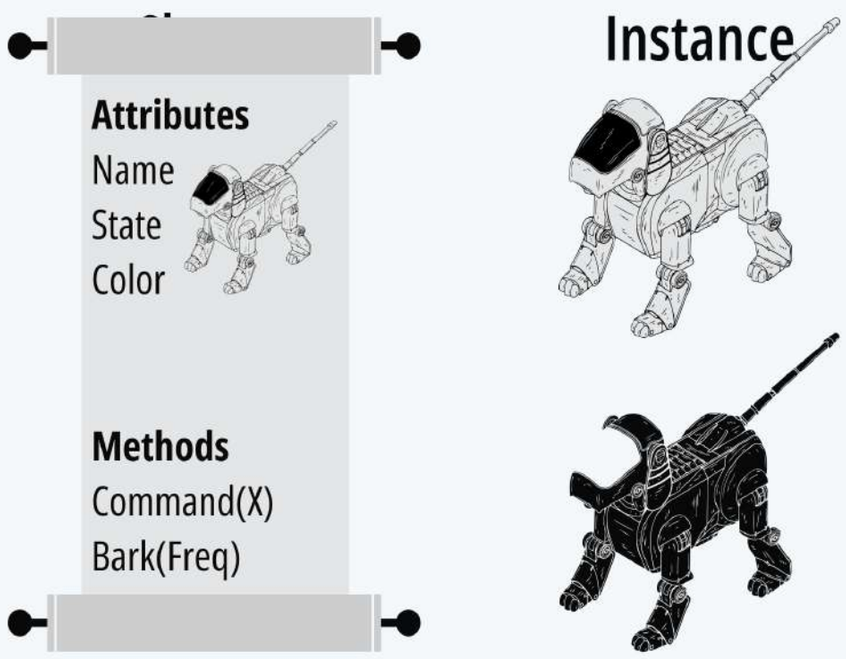
Keyword	Description	Code Example
List	A container data type that stores a sequence of elements. Unlike strings, lists are mutable: modification possible.	l = [1, 2, 2] print(len(l)) # 3
Adding Elements	Add elements to a list with (i) append, (ii) insert, or (iii) list concatenation. The append operation is very fast.	[1, 2, 2].append(4) # [1, 2, 2, 4] [1, 2, 4].insert(2,2) # [1, 2, 2, 4] [1, 2, 2] + [4] # [1, 2, 2, 4]
Removal	Removing an element can be slower.	[1, 2, 2, 4].remove(1) # [2, 2, 4]
Reversing	This reverses the order of list elements.	[1, 2, 3].reverse() # [3, 2, 1]
Sorting	Sorts a list. The computational complexity of sorting is O(n log n) for n list elements.	[2, 4, 2].sort() # [2, 2, 4]
Indexing	Finds the first occurence of an element in the list & returns its index. Can be slow as the whole list is traversed.	[2, 2, 4].index(2) # index of element 4 is "0" [2, 2, 4].index(2,1) # index of element 2 after pos 1 is "1"
Stack	Python lists can be used intuitively as stack via the two list operations append() and pop().	stack = [3] stack.append(42) # [3, 42] stack.pop() # 42 (stack: [3]) stack.pop() # 3 (stack: [])

Keyword	Description	Code Example
Set	A set is an unordered collection of elements. Each can exist only once.	basket = {'apple', 'eggs', 'banana', 'orange'} same = set(['apple', 'eggs', 'banana', 'orange'])
Dictionary	The dictionary is a useful data structure for storing (key, value) pairs.	calories = {'apple' : 52, 'banana' : 89, 'choco' : 546}
Reading And Writing Elements	Read and write elements by specifying the key within the brackets. Use the keys() and values() functions to access all keys and values of the dictionary.	print(calories['apple'] < calories['choco']) # True calories['cappu'] = 74 print(calories['banana'] < calories['cappu']) # False print('apple' in calories.keys()) # True print(52 in calories.values()) # True
Dictionary Looping	You can loop over the (key, value) pairs of a dictionary with the items() method.	**for k, v in calories.items():** **print(k) if v > 500 else None** # 'chocolate'
Membership Operator	Check with the 'in' keyword whether the set, list, or dictionary contains an element. Set containment is faster than list containment.	basket = {'apple', 'eggs', 'banana', 'orange'} print('eggs' in basket} # True print('mushroom' in basket} # False

List And Set Comprehension	List comprehension is the concise Python way to create lists. Use brackets plus an expression, followed by a for clause. Close with zero or more for or if clauses.
	Set comprehension is similar to list comprehension.
		# List comprehension
		l = [('Hi ' + x) for x in ['Alice', 'Bob', 'Pete']]
		print(l) # ['Hi Alice', 'Hi Bob', 'Hi Pete']
		l2 = [x * y for x in range(3) for y in range(3) if x>y]
		print(l2) # [0, 0, 2]
		# Set comprehension
		squares = { x**2 for x in [0,2,4] if x < 4 } # {0, 4}

Classes

Keyword	Description	Code Example

Classes	A class encapsulates data and functionality - data as attributes, and functionality as methods. It is a blueprint to create concrete instances in the memory.
	 
		`class Dog:`
		`""" Blueprint of a dog """`
		`# class variable shared by all instances`
		`species = ["canis lupus"]`
		
		`def __init__(self, name, color):`
		`  self.name = name`
		`  self.state = "sleeping"`
		`  self.color = color`
		
		`def command(self, x):`
		`  if x == self.name:`
		`  self.bark(2)`
		`  elif x == "sit":`
		`  self.state = "sit"`
		`  else:`
		`  self.state = "wag tail"`
		
		`def bark(self, freq):`
		`  for i in range(freq):`
		`  print("[" + self.name`
		`  + "]: Woof!")`
		
		`bello = Dog("bello", "black")`
		`alice = Dog("alice", "white")`
		`print(bello.color) # black`
		`print(alice.color) # white`
		`bello.bark(1) # [bello]: Woof!`

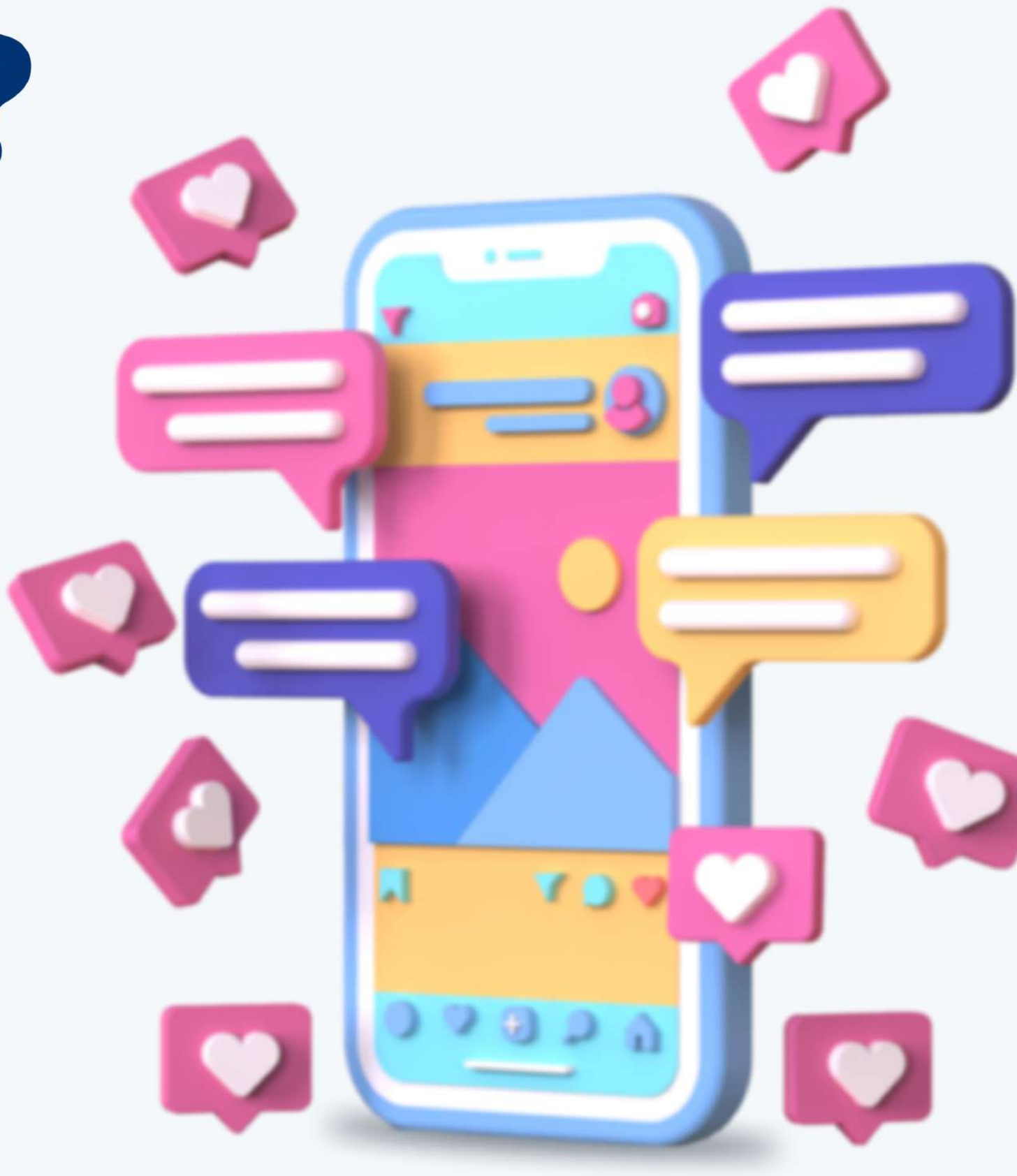
Instance	You are an instance of the class human. An instance is a concrete implementation of a class: all attributes of an instance have a fixed value. Your hair is blond, brown, or black - but never unspecified. Each instance has its own attributes independent of other instances. Yet, class variables are different. These are data values associated with the class, not the instances. Hence, all instance share the same class variable **species** in the example.	

Keyword	Description	Code Example
Self	The first argument when defining any method is always the **self** argument. This argument specifies the instance on which you call the method. **self** gives the Python interpreter the information about the concrete instance. To define a method, you use **self** to modify the instance attributes. But to call an instance method, you do not need to specify **self**.	alice.command("sit") print("[alice]: " + alice.state) # [alice]: sit bello.command("no") print("[bello]: " + bello.state) # [bello]: wag tail alice.command("alice") # [alice]: Woof! # [alice]: Woof!

Creation	You can create classes “on the fly” and use them aslogical units to store complex data types.
	
	`class Employee():`
	`  pass`
	`employee = Employee()`
	`employee.salary = 122000`
	`employee.firstname = "alice"`
	`employee.lastname = "wonderland"`
	
	`print(employee.firstname + " "`
	`+ employee.lastname + " "`
	`+ str(employee.salary) + "$")`
	`# alice wonderland 122000$`
		bello.species += ["wulf"]
		print(len(bello.species)
		== len(alice.species)) # True (!)

SKILL<S/ASH>

DID YOU LIKE THIS POST?



TELL US IN THE COMMENTS BELOW!

DROP A FOLLOW FOR MORE SUCH VALUABLE CONTENT!