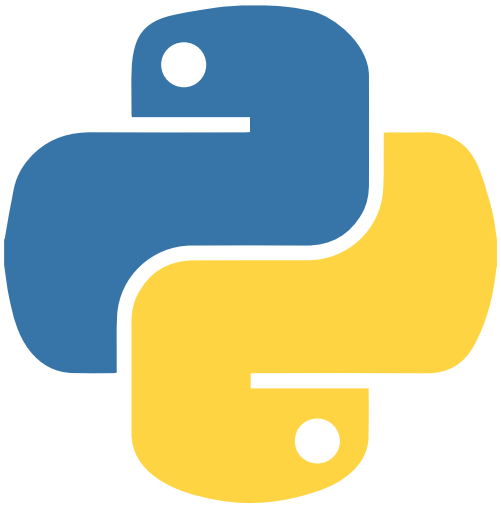
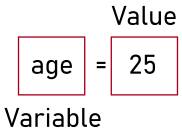


Python

A note by
Shah Md. Arshad Rahman Ziban

Chapter 00: Variables and Data types

Variables:



Example:

```
age = 25
first_name = "John"
is_student = True
print(age)  # 25
print(first_name)  # John
print(is_student)  # True
```

Variables hold values that can be used and modified throughout the program.

Data types:

Category	Data Types	Example
Text	str	"Hello, World!"
Numeric	int	45
	float	3.14
	complex	3 + 4j
Sequence	list	[1, 2, 3, 4, 5]
	tuple	(1, 2, 3, 4, 5)
	range	range(5) (which generates 0, 1, 2, 3, 4)
Mapping	dict	{"name": "Ziban", "age": 24}
Set	set	{1, 2, 3, 4, 5}
	frozenset	frozenset([1, 2, 3, 4, 5])
Boolean	bool	True or False
Binary	bytes	b"Hello"
	bytearray	bytearray(b"Hello")
	memoryview	memoryview(b"Hello")

Data types and their uses

Data Types	Uses
str	Storing and manipulating text data.
int	Representing whole numbers.
float	Representing decimal numbers.
complex	Representing complex numbers with real and imaginary parts.
list	Storing collections of items that can be changed.
tuple	Storing collections of items that cannot be changed (immutable).
range	Generating a sequence of numbers, often used in loops.
dict	Storing key-value pairs, often used for fast lookups and to represent structured data.
set	Storing unique items and performing set operations like union, intersection, etc.
frozenset	Similar to `set`, but immutable, making it hashable and usable as a dictionary key.
bool	Representing truth values, often used in conditions and logical operations.
bytes	Handling binary data, such as files or network packets.
bytearray	Similar to `bytes`, but mutable, allowing modification of binary data.
memoryview	Accessing the internal data of an object without copying, useful for handling large data sets.

Chapter 01: Operators

Operator Type	Operators	Description	Example
Arithmetic	+, -, *, /, %, **, //	Basic mathematical operations	x = 10 y = 3 (x / y) # 3.333... (x % y) # 1 (x ** y) # 1000 (x // y) # 3
Assignment	=, +=, -=, *=, /=, %=	Assign values to variables	x = 5 x += 3 print(x) # 8 x *= 2 print(x) # 16
Comparison	==, !=, >, <, >=, <=	Compare values and return a boolean result	x = 5 y = 3 (x == y) # False (x != y) # True (x >= y) # True (x <= y) # False
Logical	and, or, not	Logical operations combining boolean values	x = True y = False (x and y) # False (x or y) # True (not x) # False
Bitwise	&, |, ^, ~, <<, >>	Manipulate binary representations of integers directly, enabling efficient handling of bits within a number.	x = 5 # 0b0101 y = 3 # 0b0011 (x & y) # 1 (0b0001) (x | y) # 7 (0b0111) (x ^ y) # 6 (0b0110) (~x) # -6 (0b1010) (x << 2) # 20 (0b10100) (x >> 2) # 1 (0b0001)
Membership	in, not in	Test if a sequence contains an item	x = [1, 2, 3] (2 in x) # True (4 not in x) # True
Identity	is, is not	Test if two variables refer to the same object	x = [1, 2, 3] y = x z = [1, 2, 3] (x is y) # True (x is z) # False (x is not z) # True
Unary	-, +, ~, not	Unary operations on a single operand	x = -5 (-x) # 5 (+x) # -5 (~x) # 4 (not x) # False

Chapter 02: Conditionals

In Python, conditionals allow you to execute code based on whether certain conditions are true or false. The primary conditional statements in Python are if, elif, and else. Here's a quick overview of how to use them:

Condition Type	Example
if Statement	x = 10 if x > 5: print("x is greater than 5")

elif Statement	x = 10
	if x > 15:
	  print("x is greater than 15")
	elif x > 5:
	  print("x is greater than 5 but less than or equal to 15")

else Statement	x = 2
	if x > 15:
	  print("x is greater than 15")
	elif x > 5:
	  print("x is greater than 5 but less than or equal to 15")
	else:
	  print("x is 5 or less")

Combining Conditions (combine multiple conditions using logical operators like and, or, and not.)	x = 7 if x > 5 and x < 10: print("x is between 5 and 10")

Nested Conditionals	x = 10
	if x > 5:
	  if x < 15:
	  print("x is between 5 and 15")
	  else:
	  print("x is 15 or more")
	else:
	  print("x is 5 or less")

Chapter 03: Loops

For Loop:

Section	Code Example	Output
Iterating Over Lists	for fruit in ["apple", "banana", "cherry"]: print(fruit)	apple banana cherry
Iterating Over Strings	for char in "hello": print(char)	h e l l o
Range of Numbers	for i in range(2): print(i)	0 1 2
Range with Start and End	for i in range(1, 3): print(i)	1 2
Range with Step	for i in range(1, 10,3): print(i)	1 4 7
Iterating Over Dicts	for k, v in {"name": "John", "age": 30}.items(): print(k, v)	name John age 30

Nested Loops	for x in ["red", "big"]:
	for y in ["apple", "banana"]:
	print(x, y)
		red apple
		red banana
		big apple
		big banana

Loop Control	for i in range(5): if i == 3: break print(i)	0 1 2
	for i in range(5): if i % 2 == 0: continue print(i)	1 3
else Clause	for i in range(3): print(i) else: print("Done")	0 1 2 Done
List Comprehensions	squares = [x**2 for x in range(4)] print(squares)	[0, 1, 4, 9]

While Loop:

Section	Code Example	Output
Simple while Loop	i = 1 while i < 6: print(i) i += 1	1 2 3 4 5
break to Exit Early	i = 1 while i < 10: print(i) if i == 5: break i += 1	1 2 3 4 5
continue to Skip Iteration	i = 0 while i < 10: i += 1 if i % 2 == 0: continue print(i)	1 3 5 7 9
else Clause	i = 1 while i < 6: print(i) i += 1 else: print("Loop is complete")	1 2 3 4 5 Loop is complete
Infinite Loop	i = 1 while True: print(i)	

Nested while Loops	i = 1
	while i <= 3:
	j = 1
	while j <= 3:
	print(i, j)
	j += 1
	i += 1
		1 1
		1 2
		1 3
		2 1
		2 2
		2 3
		3 1
		3 2
		3 3

Chapter 04: Functions

Concept	Example	Output
Simple Function	def greet(): print("Hello, World!") greet()	Hello, World!
Function with Parameters	def greet(name): print(f"Hello, {name}!") greet("Alice")	Hello, Alice!
Default Parameters	def greet(name="World"): print(f"Hello, {name}!") greet()	Hello, World!
Default Arguments	def greet(name, message="Hello"): print(f"{message}, {name}!") greet("Alice")	Hello, Alice!
Return Value	def add(a, b): return a + b result = add(3, 4) print(result)	7
Multiple Return Values	def calculate(a, b): return a+b, a-b, a*b s, d, p = calculate(10, 5)	Sum: 15 Diff: 5 Prod: 50
Lambda Function	add = lambda a, b: a + b print(add(3, 4))	7

Nested Functions	def outer(text):
	  def inner():
	  print(text)
	  inner()
	outer('Hello')
		Hello

Variable Positional Args (Accepts any number of positional arguments using *args.)	def sum_all(*args): return sum(args) print(sum_all(1, 2, 3))	6

Variable Keyword Args
(Accepts any number of keyword arguments using **kwargs.)
	def print_details(**kwargs):
	for k,v in kwargs.items():
	print(f"{k}: {v}")
	print_details(name="Alice", age=30)
		name: Alice
		age: 30

Recursive Function	def factorial(n): return 1 if n==1 else n * factorial(n-1) print(factorial(5)	120

Chapter 05: List (Array)

List create and length:

Creating	# Empty list
	my_list = []
	
	# List of integers
	my_list = [1, 2, 3, 4, 5]
	
	# List with mixed data types
	my_list = [1, "Hello", 3.14]

Lenght	my_list = [10, 20, 30, 40] length = len(my_list) # Returns 4

Additional List Operations :

Method	Description	Example Code	Output
append()	Adds a single element to the end of the list.	my_list = [1, 2, 3] my_list.append(4) print(my_list)	[1, 2, 3, 4]
insert()	Inserts an element at a specified index.	my_list.insert(1, 'a')	[1, 'a', 2, 3]
extend()	Adds multiple elements to the end of the list.	my_list.extend([4, 5, 6])	[1, 2, 3, 4, 5, 6]
List Concatenation	Adds elements by concatenating another list.	my_list = my_list + [4, 5]	[1, 2, 3, 4, 5]
List Comprehension	Adds elements generated by a comprehension.	my_list.extend([x + 10 for x in my_list]	[1, 2, 3, 11, 12, 13]

List create and length :

Method	Description	Example Code	Output
remove()	Removes the first occurrenceof a specified value from the list.	my_list = [1, 2, 3, 2, 4] my_list.remove(2) print(my_list)	[1, 3, 2, 4] Removing an item not index
pop()	Removes and returns an item by index (last item if no index is given).	last_item = my_list.pop() second_item = my_list.pop(1)	last_item = 4 my_list = [1, 3, 2] second_item = 2 [1, 3, 2, 4]
del	Deletes an item at a specified index or a slice of items.	del my_list[2] Deleting the items from index 1 to 3 (2, 3, 2) del my_list[1:4]	[1, 2, 2, 4] [1, 4]
List Concatenation	Creates a new list excluding items that match a condition.	my_list = [item for item in my_list if item != 2]	[1, 3, 4]
clear()	Removes all items from the list.	my_list.clear()	[ ]

Arrange (sort) a list :

Method	Description	Example Code	Output
sort()	Sorts the list in-place in ascending or descending order.	num = [4, 2, 9] num.sort() num.sort(reverse=True)	[2,4,9] [9,4,2]
sorted()	Returns a new sorted list from the elements of any iterable. (original list remains unchanged)	x = sorted(num) y= sorted (num, reverse=True)	[2,4,9] [9,4,2]
Sorting Strings	Sorts a list of strings alphabetically or in reverse alphabetical order.	words = ["banana", "apple"] x = words.sort() y = words.sort(reverse=True)	['apple', 'banana'] ['banana', 'apple']
Custom Key Sorting	Sorts using a custom key function, like sorting strings by length.	x = words.sort(key=len) y = sorted(words, key=len)	['apple', 'banana'] ['apple', 'banana']

Key Differences Between sort() and sorted():

Modification : sort() modifies the list in place, while sorted() creates a new sorted list.
Return Value : sort() returns None, while sorted() returns a new sorted list.
Applicability : sort() can only be used with lists, whereas sorted() works with any iterable, including dictionaries and tuples.

Generate a list :

Method	Description	Example Code	Output
Filtering Odd Numbers with filter and lambda	Filters odd numbers from a range.	odd = list(filter(lambda x: x % 2 == 1, range(1, 11)))	[1, 3, 5, 7, 9]
Squaring Odd Numbers with List Comprehension	Squares odd numbers from 1 to 10.	sq = [x ** 2 for x in range(1, 11) if x % 2 == 1]	[1, 9, 25, 49, 81]
Filtering Numbers Greater Than 5 with List Comprehension	Filters numbers greater than 5 from a list.	X = [x for x in [3, 4, 5, 6, 7] if x > 5]	[6, 7]
Filtering Numbers Greater Than 5 with filter and lambda	Filters numbers greater than 5 from a list using filter.	X = list(filter(lambda x: x > 5, [3, 4, 5, 6, 7]))	[6, 7]

List Slicing : a = ['spam', 'egg', 'bacon', 'tomato', 'ham', 'lobster']

Operation	Code	Output
Basic Slicing	print(a[2:5])	['bacon', 'tomato', 'ham']
Negative Index Slicing	print(a[-5:-2])	['egg', 'bacon', 'tomato']
Positive Index Slicing	print(a[1:4])	['egg', 'bacon', 'tomato']
Omitting Start Index	print(a[:4])	['spam', 'egg', 'bacon', 'tomato']
Omitting Start Index with 0	print(a[0:4])	['spam', 'egg', 'bacon', 'tomato']
Omitting End Index	print(a[2:])	['bacon', 'tomato', 'ham', 'lobster']
Omitting End Index with len(a)	print(a[2:len(a)])	['bacon', 'tomato', 'ham', 'lobster']
Omitting Both Indices	print(a[:])	['spam', 'egg', 'bacon', 'tomato', 'ham', 'lobster']
Slicing with a Stride (Step)	print(a[0:6:2])	['spam', 'bacon', 'ham']
Slicing with a Stride (Step) from Index 1	print(a[1:6:2])	['egg', 'tomato', 'lobster']
Slicing with a Negative Stride	print(a[6:0:-2])	['lobster', 'tomato', 'egg']
Reversing the List with Slicing	print(a[::-1])	['lobster', 'ham', 'tomato', 'bacon', 'egg', 'spam']

Count & Repeating :

Method	Description	Example Code	Output
count()	Counts the number of occurrences of a specified element in a list.	li = [3, 1, 3, 2, 5] print(li.count(3))	2
List Repetition	Repeats elements in a list a specified number of times.	li = ["re"] * 3 print(li)	['re', 're', 're']

Chapter 06: String

Strings in Python are sequences of characters. They can be created by enclosing characters in single quotes, double quotes, or triple quotes for multi-line strings.

Accessing Characters : str = "Hello"

Example Code	Output	Description
print(str[0])	H	Prints the first character of the string str, which is "H".
print(str[1])	e	Prints the second character of the string str, which is "e".
print(str[-1])	o	Prints the last character of the string str, which is "o".

Slicing Strings : str = "Hello"

Example Code	Output	Description
print(str[1:4])	ell	Slices the string str from index 1 to 3 (4 is excluded), giving "ell".
print(str[:2])	He	Slices the string from the start up to index 1 (2 is excluded), giving "He".
print(str[2:])	llo	Slices the string from index 2 to the end, giving "llo".
print(str[:])	Hello	Slices the entire string, giving the whole string "Hello".
print(str[::2])	Hlo	Slices the string from start to end, stepping by 2, giving "Hlo".

String Concatenation and Repetition :

Example Code	Output	Description
str1 = "Hello" str2 = "World" print(str1 + " " + str2)	Hello World	Concatenates str1, a space " ", and str2, resulting in "Hello World".
print(str1 * 2)	HelloHello	Repeats str1 three times, resulting in "HelloHello".

String Methods : str = "Hello World"

Method	Description	Example Code	Output
lower()	Converts all characters in the string to lowercase.	print(str.lower())	hello world
upper()	Converts all characters in the string to uppercase.	print(str.upper())	HELLO WORLD
title()	Converts the first character of each word to uppercase.	print(str.title())	Hello World
strip()	Removes leading and trailing whitespace.	print(" Hello ".strip())	Hello
split()	Splits the string into a list of words based on whitespace.	print(str.split())	['Hello', 'World']
isalpha()	Checks if all characters in the string are alphabetic.	print(str.isalpha())	False because the space is not an alphabetic character.
isdigit()	Checks if all characters in the string are digits.	print("123".isdigit())	True

Chapter 06: User input

Description	Example
Single input (string)	name = input("Enter your name: ")
Multiple inputs (space-separated)	num1, num2, num3 = input("Enter three numbers separated by space: ").split()
Float input	height = float(input("Enter your height in meters: "))
Integer input	age = int(input("Enter your age: "))
Yes/No input	is_student = input("Are you a student? (yes/no): ")
Convert space-separated string input to a list of integers	arr = list(map(int, input("Enter the list of numbers separated by space: ").split())) Output : arr = [1, 2, 3, 4, 5]