WELCOME

Python Series

AGENDA

Character Sets
Tokens
Types of Tokens
Punctuators and Delimiters
Escape Sequence

CHARACTER SETS

- Set of valid characters recognized by programming language, in this case, python.
- It includes:
  - **Alphabets**: (A-Z) and (a-z)
  - **Digits**: 0-9
  - **Special Symbols**: “ ‘ : ; ~ ! @ # $ % ^ & * ( ) _ - + = { } [ ] | \
  - **White Spaces**: tab space, blank space, new line and carriage return
  - **Others**: All ASCII and UNICODE characters are supported by python

TOKENS

- Definition: Tokens are smallest individual AND meaningful unit of code.

- Example:
  - date = 12, in this case, tokens are 'date', '=', '12', because python interpreter understands that there is a user defined variable (date) that store something (12).
  - It is wrong to say that tokens are 'd', 'a', 't', 'e' etc,

- Role: They are building blocks of python syntax, used by python interpreter to understand and execute code.

- Tokens includes Keywords, Identifiers, Literals, Operators, Punctuators (Delimiters)

TYPES OF TOKENS - KEYWORDS

- Keywords are words that has some special meaning or significance.
- Keywords cannot be used to arbitrary purposes, like naming variables or functions
- They are used because of their unique properties.
- Python has 33 keywords.
- Some of them are: try, catch, break, continue, except, False, True, if, in etc

TYPES OF TOKENS - IDENTIFIERS

- Identifiers are names given to entities such as variables, functions, classes, lists or methods for identification.

- Python is case sensitive, hence affecting how identifiers are named. So apple and Apple are considered different identifiers.

- It starts with either alphabets (including A-Z and a-z) or underscore (_). Everything else is unacceptable. Valid examples are abc, _abc, Abc, but not $abc or =abc.

- Digits can be part of identifiers but cannot be the first character. Example includes greet4, my_1st_name, but not 1st_place.

TYPES OF TOKENS - IDENTIFIERS

- Special Characters other than underscore (_) is not allowed in identifiers.
- Identifiers cannot be python keywords. Hence if, for, in, try etc cannot be named as identifiers.
- Valid Examples:
  - myVar, __init__, data2
- Invalid Examples:
  - 2be0rNot2Be (starts with digit), high-five (contains a hyphen), class (keyword), 'hello world' (contains white space).

TYPES OF TOKENS - LITERALS

- Any **Fixed** or **Constant values** of a program are known as **literals**.

- **Types of Literals:**
  - **String Literals**: Represent text enclosed in single, double, or triple quotes. Example: ‘Python’, “Learning”, “““Multi-line String”””

  - **Character Literals**: A single character enclosed in single or double quotes. Example: ‘A’, “z”

  - **Boolean Literals**: Represent one of two values: True or False.

TYPES OF TOKENS - LITERALS

- **Types of Literals:**
  - **Numeric Literals**: These are literals written in the form of numbers. They are of following types:
    - **Integer Literals**: Positive or negative whole numbers without a fractional part. Examples: 42, -99, 0b1010 (binary), 0x1A (hexadecimal), 0o123 (octal).
    - **Float Literal**: Real numbers with fractional parts. Examples: 3.14, -0.001
    - **Complex Literal**: Numbers with a real and imaginary part. Examples: 3+4j, -5-6j

TYPES OF TOKENS - LITERALS

- **Types of Literals:**
  - **Special Literals**: Python uses 'None' to represent absence of value.
  - **Literals Collection**: It includes list, tuples, dictionary and sets.
    - **List**: An ordered collection of elements in square brackets, mutable (changeable). Example: [1, 'apple', 3.14]
    - **Tuple**: An ordered collection of elements in parentheses, immutable (not changeable). Example: (1, 'banana', 7.89)
    - **Dictionary**: An unordered collection of key-value pairs in curly braces. Example: {'name': 'Alice', 'age': 30}
    - **Set**: An unordered collection of unique elements in curly braces. Example: {3, 5, 7, 5} (will result in {3, 5, 7} since sets contain unique elements)

TYPES OF TOKENS - OPERATORS

- **Definition**: Operators are tokens that perform operations on variables and values in an expression

- **Operands**: The variables or values on which operators act are called operands.
- Example: 5+6=11, (here, 5,6 are operands, + are operators, 5+6 is expression)

- **Types of Operators:**
  - **Unary Operator**: Act on a single operand.
  - **Example**:
    - **Negation (-):** if x = 5 then –x evaluates to -5.
    - **Logical NOT (not):** = if flag = True then not flag evaluates to False.

  - **Binary Operator:** Requires two operands to operate.
  - **Example**:
    - **Addition (+):** 5 + 3 = 8
    - **Subtraction (-):** 5 – 3 = 2
    - **Multiplication (*):** 5 * 3 = 15
    - **Division (/):** 6 / 3 = 2

Python Series

TYPES OF TOKENS - PUNCTUATORS

- **Definition**: Punctuators include symbols that help in structuring the program.

- **Types of Punctuators:**
  - **Parentheses (( )):** Enclose expressions and parameters in function calls.
  - **Brackets ([ ]):** Define lists, list comprehensions, and indexing.
  - **Braces ({ }):** Define sets and dictionaries.
  - **Commas (,):** Separate items in lists, tuples, function arguments, and multiple variable assignments.
  - **Colons (:):** Define the start of an indented block (e.g., after function definitions, loops, conditions) and separate keys from values in dictionaries or Used in slice notation.
  - **Semicolons (;):** Optionally separate multiple statements on a single line. Overuse is discouraged as it can reduce code readability.
  - **Quotes (' ', " "):** Denote string literals.
  - **Period (.):** Access attributes of objects or to indicate floating-point numbers.
  - **Backslash (\ ):** Used in escape characters and line continuation.

Python Series

PUNCTUATORS | DELIMITERS

	Punctuators	Delimiters
Use	Structure and operate on the code.	Separate and enclose code elements.
Scope	All punctuators are **NOT** delimiters	All delimiters are punctuators. (Subset of punctuators).
Example	`=`, `:`, `;`, `.`	`,`, `( )`, `[ ]`, `{ }`
Use Case	**Assignment Operator (=)**: Used to assign a value to a variable, e.g., x = 10. **Colon (:)**: Introduces a block of code, like in function definitions or loops, e.g., for i in range(5):. **Semicolon (;)**: Used to separate multiple statements on a single line, e.g., a = 5; b = 10;. **Period (.)**: Used for object attribute access, e.g., object.method().	**Commas (,):** Separate elements in a list, tuple, function arguments, or multiple assignments, e.g., my_list = [1, 2, 3]. **Parentheses (()):** Enclose tuples or function parameters and arguments, e.g., my_function(arg1, arg2). **Brackets ([]):** Define lists or index/slice arrays and strings, e.g., my_list = [1, 2, 3]. **Braces ({}):** Enclose sets and dictionaries, e.g., my_dict = {'key': 'value'}.

ESCAPE SEQUENCE

- **Definition**: Sequences of characters that have a special meaning when used inside a string or character.
- **Syntax** : Characters need to be preceded by a backslash (\) character.
- **Use**: To insert characters that are illegal in a string, use an escape character.
- **Illegal Characters**: Characters that cannot be directly inserted into a string are termed as Illegal characters
- **Example of Illegal character:**
  - Text = 'this is my cat's hat.'
- **Correction**:
  - Text = 'this is my cat\'s hat.'

ESCAPE SEQUENCE

Escape Sequence	**Meaning**	**Code**	**Result**
\n	New line	print("hello\nworld")	hello world
\t	Horizontal tab	print("hello\tworld")	hello world
\	Backslash	print("hello\\world")	hello\world
\"	Double quote	print("hello\"world\"")	hello"world"
\'	Single quote	print('hello\'world\'')	hello'world'
\r	Carriage return	print("hello\rworld")	world (overwrites hello)
\b	Backspace	print("hello\bworld")	hellworld (removes o)

Upcoming…

Data Types

Python Series