Team 5 Project 1: Tokens

- Jeevana Shruti K
- Parikshith Saraswathi
- Raaj Atulkumar Patel

Procedural Language

- Program is organized into procedures/functions.

- Focus is on **step-by-step instructions** (how to do).

- Data and functions are **separate**.

Object-Oriented Language (OOP)

- Program is organized into objects
- Focus is on modeling real-world entities
- Supports concepts like encapsulation, inheritance, polymorphism

C Keywords

Definition: Reserved words with fixed meaning in the C language.

Purpose: Form the building blocks for data types, control flow, and program structure.

Data types: int, float, char, double, void

Control flow: if, else, for, while, switch, goto, break, continue

Memory management / storage: static, extern, register, auto, sizeof

Function & structure: return, struct, union, enum, typedef

Count: ~32 standard keywords (C90)

More added in later standards (C99, C11, C23, etc.)

C keywords are low-level (data types, control flow).

Note: Keywords cannot be used as identifiers (e.g., variable names).

Keywords in Python

- Reserved words, case-sensitive
- Examples: def, class, if, elif, else, while, for, return, import, try, except, with, yield, lambda
- Around 35 keywords (from Python 3.12v)

Identifiers in C

Definition: Names given to variables, functions, arrays, structures, etc.

Rules:

- Must begin with a letter or underscore (_)
- Can include letters, digits, underscores afterward
- Case-sensitive (sum ≠ Sum)
- Cannot be a keyword

Examples: sum, main, _count

Identifiers in Python

- Names for variables, functions, classes, modules, etc.
- Rules: Similar to C, cannot use keywords
- Example: total, MyClass, _hidden

Literals in C

Definition: Fixed values used directly in a program.

- Types of Literals:
    1. Numeric → 10, 3.14, 0x1F, 075
    2. Character → 'a', '\n'
    3. String → "hello"
- Constants in C:
    1. const keyword
    2. #define macros

Literals in Python

- Numeric: 10, 3.14, 0b1010, 0o12, 0x1F
- String: "hello", 'world', triple quotes ("""...""")
- Boolean: True, False
- Special: None, complex numbers (3+4j), lists ([1,2,3]), dicts ({"a":1}), sets

Operators in C

Category	Symbols	Purpose
Arithmetic	+ - * / %	Math operations
Relational	< <= > >= == !=	Compare values
Logical	&& || !	Logical and/or/not
Bitwise	& | ^ ~ << >>	Bitwise and/or/Xor/not/left shift/right shift
Assignment	= += -= *= /= %= ...	Assign/update values
Ternary (Conditional)	?:	Inline condition
Increment/Decrement	++ --	Step values

Operators in Python

- Arithmetic: + - * / % ** //
- Relational: < <= > >= == !=
- Logical: and or not (keywords instead of symbols)
- Bitwise: & | ^ ~ << >>
- Assignment: = += -= *= /= %= **= //= <<= >>= &= ^= |=
- Membership: in, not in
- Identity: is, is not

Separators/Delimiters in C

Delimiter	Purpose
;	End of statement
{ }	Code block
( )	Functions, precedence
[ ]	Arrays
,	Separator

Separators/Delimiters in Python

- : (block start, loops, if, functions, classes)
- ( ) (grouping, functions)
- [ ] (lists, indexing)
- { } (dicts, sets)
- , (comma)
- No semicolons required (but optional)

Comments in C

- **Purpose**:
  - Add explanations, notes, or reminders in code
  - Ignored by the compiler (do not affect execution)

- **Types of Comments**:
  - Single-line → // comment
  - Multi-line → /* comment */

Comments in Python

- Single-line: # comment
- Multi-line: technically no native syntax, but triple quotes (""" … """) often used as docstrings

Whitespace in C

Definition: Spaces, tabs, and newlines used in code.

Behavior:

- Ignored by the compiler except inside strings or character literals
- Used for code readability and organization
- Indentation: Improves clarity but has no effect on execution

Indentation in Python

- Whitespace (indentation) is syntactically significant
- Indentation defines code blocks

Special/Extra Operators C vs Python

C Only	Python Only	Points
++ / --	*(none)*	Increment / decrement operators exist in C, not in Python
?: (ternary conditional)	x if cond else y	C uses ternary operator, Python uses keyword-based syntax
->	*(none)*	Member access through pointers in C. Not needed in Python
sizeof	len(), sys.getsizeof()	C uses sizeof operator; Python uses built-in functions
(none)	**	Exponentiation in Python
(none)	//	Floor division in Python
(none)	in, not in	Membership testing (lists, sets, dicts)
(none)	is, is not	Identity comparison (object reference check)
(none)	and, or, not	Python uses keywords instead of &&, `

C vs Python Tokens Comparison

Category	C Example	Python Example	Notes

Keywords	`int x = 5;`
	`if (x > 0) {`
	` return 1;`
	`} else { return 0;}`
		`x = 5`
		`if x > 0:`
		`  return_value = 1`
		`else:`
		`  return_value = 0`
			C has `int`, explicit types;
			Python does not require type declarations.

Identifiers	`int totalSum = 100; // identifier`	`total_sum = 100 # identifier\n`	Python follows **snake_case** (PEP8) C often uses **camelCase**.

Literals	`int age = 25;`
	`char grade = 'A’;`
	`char name[] = "John";`
		`age = 25`
		`grade = 'A’`
		`name = "John"`
			Python has no `char` type;
			`C – 'A'` is a **string of length 1**.

Operators	`int a = 10, b = 3;`
	`int result = a / b;`
	`int mod = a % b;`
	`a++;`
		`a, b = 10, 3`
		`result = a / b`
		`mod = a % b`
		`power = a ** b`
			Python has `**` (power), `//` (floor division);
			C has `++`, `--`, ternary `?:`.

Separators / Delimiters	`if (a > b) {`
	` printf("A is bigger \\n");}`
		`if a > b:`
		`  print("A is bigger")`
			C uses `{ }` + `;`
			Python uses **:** + **indentation**.

Comments	`// Single-line`
	`/* Multi-line */`
		`# Single-line`
		`\"\"\"\nMulti-line`
		`(docstring)\n\"\"\"\n`
			C supports `/* ... */`;
			Python uses `#` and docstrings.

Whitespace	`if (a > 0) {`
	` printf("Positive \\n");}`
		`if a > 0:`
		`  print("Positive")`
			In C, whitespace is ignored (except in strings);
			in Python, **indentation is syntax-defining**.

Execution Time / Performance	`// Compiled to machine code`
	`// Runs very fast`
		`# Interpreted (slower execution)`	C is **compiled**, very fast, low-level control. Python is **interpreted**, slower due to dynamic typing & garbage collection, though libraries use C under the hood.

Memory Management	`int *arr = malloc(5 *`
	`sizeof(int));`
	`free(arr);`
		`arr = [1, 2, 3, 4, 5]`
		`# auto memory mangement`
			In C, programmer must manually allocate (`malloc`) and free memory. In Python, memory is managed automatically by the **garbage collector**.

C vs Python Summary Table

Category	C	Python
Keywords	Low-level (32+)	High-level (35+)
Identifiers	Case-sensitive, follow C rules	Case-sensitive, same rules, PEP8 naming style
Literals	Numbers, chars, strings	Richer: numbers, strings, bools, `None`, lists, dicts
Operators	Includes `++`, `--`, `?:`	Includes `**`, `//`, `is`, `in`
Separators	`; { } ( ) [ ] ,`	`: ( ) [ ] { } ,` + **indentation**
Comments	`//`, `/* */`	`#`, docstrings (`"""`)
Whitespace	Ignored (except in strings/chars)	**Syntax-defining** (indentation matters)
Programming Paradigm	Structural / Procedural	Object-Oriented, multi-paradigm
Execution Time	Very Fast (compiled)	Slower (interpreted, dynamic typing)
Ease of Use	Verbose, manual memory	Concise, automatic memory management
Readability	Lower (explicit, technical)	Higher (clean, beginner-friendly)

Overall: C vs Python

Aspect	C (Structural)	Python (OOP)
Readability	Lower (manual types, extra code)	Higher (clean, minimal syntax)
Ease of Use	Harder (manual memory & string handling)	Easier (automatic memory, built-ins)
Verbosity	More code for same logic	Less code, concise
Paradigm	Procedure-driven	Object-driven
Execution Time	Very Fast (compiled)	Slower (interpreted, dynamic)

Thank You