# PROGRAMMING SKILLS II

~ANURUDDHA ABEYSINGHE

*Lecture 02*

Foundation Certification in IT – Curtin batch

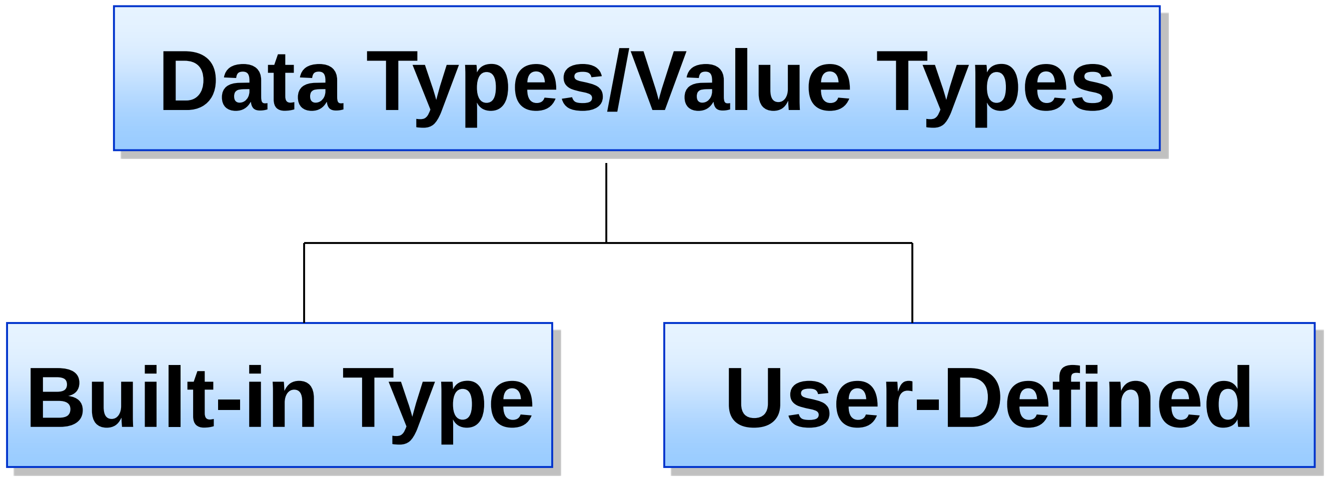
# C# LANGUAGE FUNDAMENTALS

LECTURE 02

## OVERVIEW

- Naming Variables
- Using Built-in Data Types
- Using Methods
- Arithmetic  Operator
- Operator Precedence

## COMPARING BUILT-IN AND USER-DEFINED VALUE TYPES



- **Examples of built-in value types:**
  - int
  - float
  - String

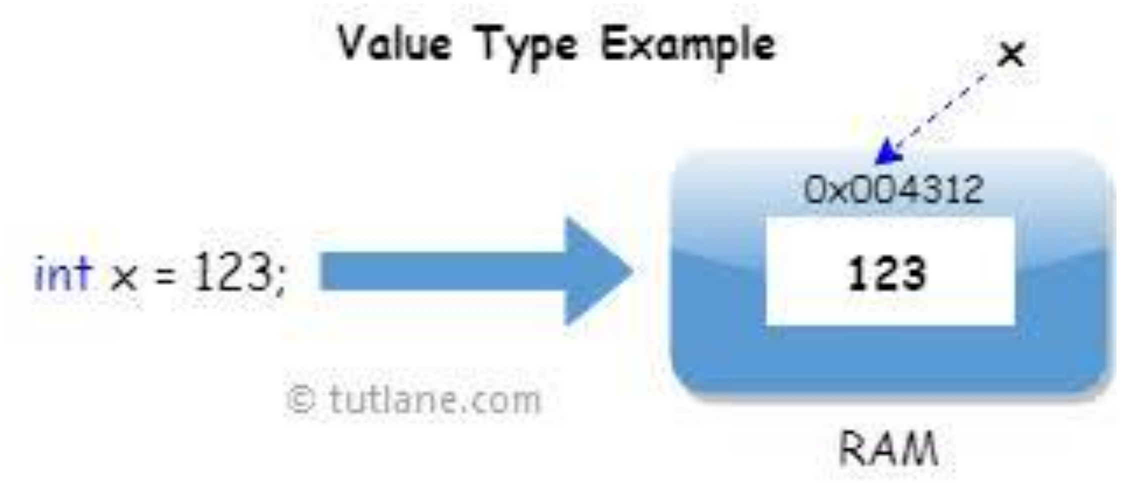
- **Examples of user-defined value types:**
  - enum
  - struct

## BUILT IN DATA TYPES USED IN C#

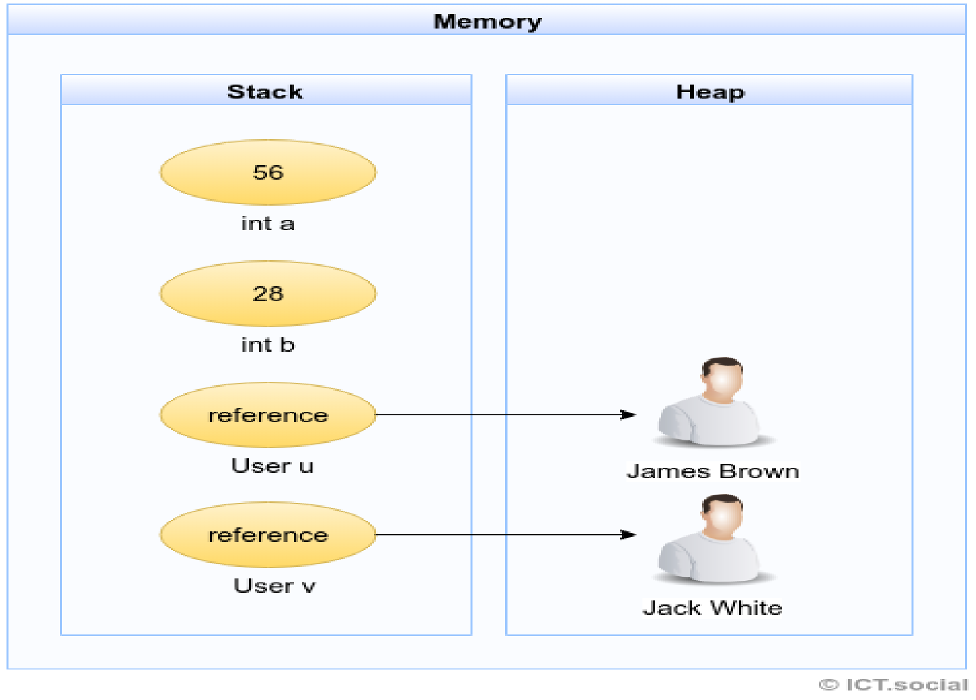
| Data Type | Memory | Used for | Examples |
|---|---|---|---|
| int | 4 bytes | Stores whole numbers | 2, 856974, 0 |
| float | 4 bytes | Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits | 2.365, 3.0 |
| double | 8 bytes | Stores fractional numbers. Sufficient for storing 15 decimal digits | 2.36589, 3.00001 |
| char | 2 bytes | Stores a single character/letter | ‘a’, ‘*’ |
| string | 2 bytes per character | A sequence of Unicode characters | “Hello”, “C# Programming” |
| bool | 8-bit | Logical True / False | TRUE , FALSE |

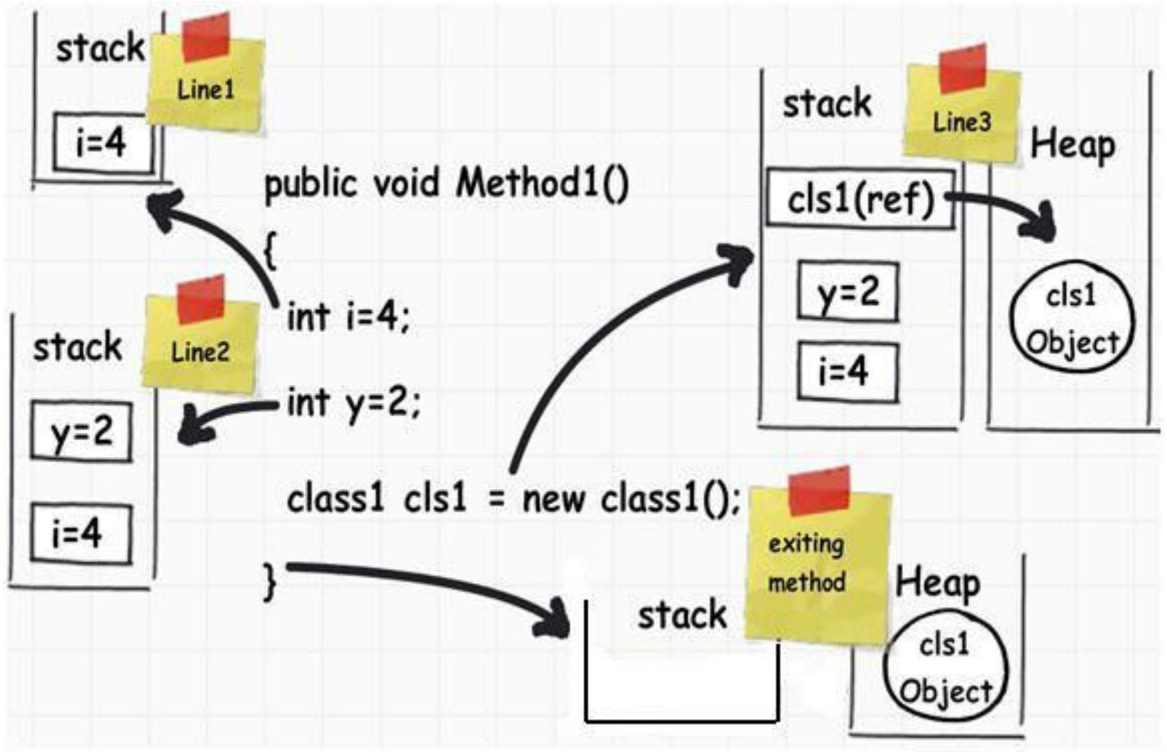
## VARIABLES

- Variable is a Label/identification Name for data in programming.
- Use to store information inside your Ram.
- Each variable has Data Type and a Name



## WHAT'S HAPPEN INSIDE YOUR RAM?





## DIFFERENCE BETWEEN STACK AND HEAP MEMORY

| | Stack Memory | Heap Memory |
|---|---|---|
| ***Execution of Memory*** | Used only by one thread of execution | Used until the application exit. |
| ***Access Memory*** | Can't access by other threads | Objects can access throughout the application |
| ***Life Time*** | Keep until end of the thread execution | Keep until end of the whole application execution |
| ***Usage*** | To contain,<br>• local primitive variables<br>• reference variable to objects in heap area | To contain object details |

## RULES AND RECOMMENDATIONS FOR NAMING VARIABLES

Rules

1. Must start with a letter or  the underscore character.
2. After the first character use letters, digits or the underscore character.
3. Do not used reserved keywords.

Recommendations

- Avoid using all uppercase letters.
- Avoid starting with an underscore.
- Avoid using abbreviations (shortened form of a word or phrase).
- Use PascalCasing naming in multiple-word names.

  (PascalCasing : capitalize the first character of each word.)

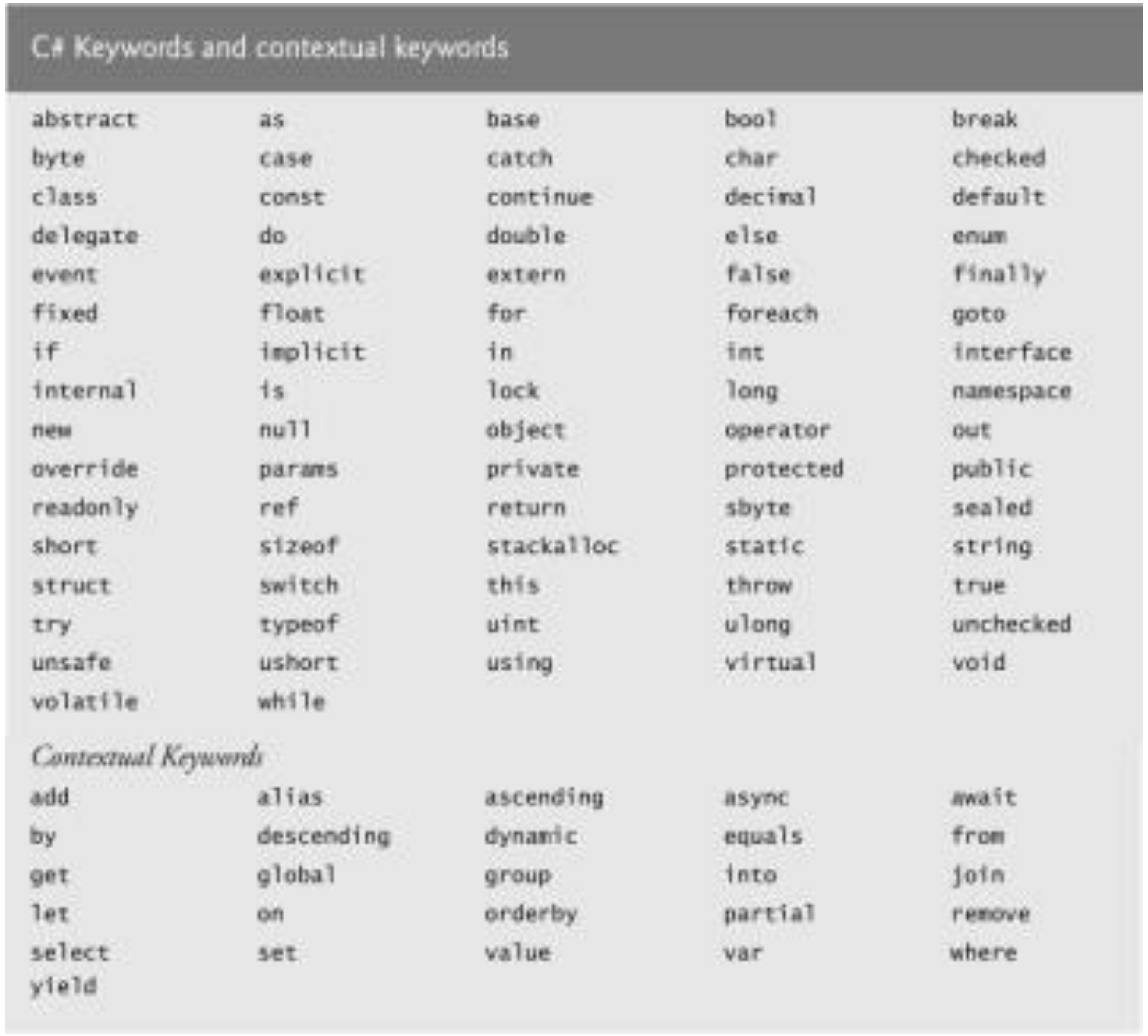
## C# KEYWORDS

```
abstract, base, bool, default, if, finally
```

- Keywords are reserved identifiers
- Do not use keywords as variable names
  - Results in a compile-time error
- Avoid using keywords by changing their case sensitivity

```
int INT;  // Poor style
```

## KEYWORDS IN C#



C# Keywords and contextual keywords

| | | | | |
|---|---|---|---|---|
| abstract | as | base | bool | break |
| byte | case | catch | char | checked |
| class | const | continue | decimal | default |
| delegate | do | double | else | enum |
| event | explicit | extern | false | finally |
| fixed | float | for | foreach | goto |
| if | implicit | in | int | interface |
| internal | is | lock | long | namespace |
| new | null | object | operator | out |
| override | params | private | protected | public |
| readonly | ref | return | sbyte | sealed |
| short | sizeof | stackalloc | static | string |
| struct | switch | this | throw | true |
| try | typeof | uint | ulong | unchecked |
| unsafe | ushort | using | virtual | void |
| volatile | while | | | |

*Contextual Keywords*

| | | | | |
|---|---|---|---|---|
| add | alias | ascending | async | await |
| by | descending | dynamic | equals | from |
| get | global | group | into | join |
| let | on | orderby | partial | remove |
| select | set | value | var | where |
| yield | | | | |

## QUIZ: CAN YOU SPOT THE DISALLOWED VARIABLE NAMES?

- `int 12count;`
- `char $diskPrice;`
- `char middleInitial;`
- `float this;`
- `int __identifier;`

## TYPE CASTING

| Data Type | Memory |
|---|---|
| int | 4 bytes |
| float | 4 bytes |
| double | 8 bytes |
| char | 2 bytes |
| string | 2 bytes per character |
| bool | 8-bit |

- Convert from one data type to another data type.

Type Casting

| Implicit Casting | Explicit Casting |
|---|---|
| Smaller size convert to Large size | Large size to Smaller size |
| No loss data | Might be data loss |
| Type safe casting | Unsafe casing |
| Ex:-<br>• Int → float<br>• Int → double<br>• Float → double<br>• Char → int | Ex:-<br>• String → char<br>• Float → int<br>• Double → int<br>• Double → float |

## TYPE CASTING – CONT.

- There are many Built-in methods to do the type casting such as,

  - **Convert.ToInt32()**
  - **Convert.ToChar()**
  - **Convert.ToDouble()**
  - **Convert.ToString()**
  - **int.Parse()**
  - **double.Parse()**

Try this -

```
int a = 5;
double b = 3.25;
bool c = true;
char x = 'x';

Console.WriteLine("Convert int to String = "+Convert.ToString(a));
Console.WriteLine("Convert int to double = "+Convert.ToDouble(a));
Console.WriteLine("Convert double to int = "+Convert.ToInt32(b));
Console.WriteLine("Convert bool to string = "+ Convert.ToString(c));
Console.WriteLine("Convert char to string = "+ Convert.ToString(x));
```

## ARITHMETIC EXPRESSIONS

- An *expression* is a combination of operators and operands
- *Arithmetic expressions* (we will see logical expressions later) compute numeric results and make use of the arithmetic operators:

| | |
|---|---|
| Addition | + |
| Subtraction | - |
| Multiplication | * |
| Division | / |
| Remainder | % |

Find the answer : 2+6/2*5+2-1*3

## DIVISION AND REMAINDER

- If both operands to the division operator (/) are integers, the result is an integer (the fractional part is discarded)

```
14 / 3
```
equals?

```
8 / 12
```
equals?

- The remainder operator (%) returns the remainder after dividing the second operand into the first

```
14 % 3
```
equals?

```
8 % 12
```
equals?

## OPERATOR PRECEDENCE

- Operators can be combined into complex expressions

```
result  =  total + count / max - offset;
```

- Operators have a well-defined precedence which determines the order in which they are evaluated

- *Precedence rules*
  - Parenthesis are done first
  - Division, multiplication and modulus are done second
    - Left to right if same precedence (this is called associativity)
  - Addition and subtraction are done last
    - Left to right if same precedence

## PRECEDENCE OF ARITHMETIC OPERATIONS

| Operator(s) | Operation | Order of evaluation (precedence) |
|---|---|---|
| `( )` | Parentheses | Evaluated first. If the parentheses are nested, the expression in the innermost pair is evaluated first. If there are several pairs of parentheses "on the same level" (i.e., not nested), they are evaluated left to right. |
| `*`, `/` or `%` | Multiplication<br>Division<br>Modulus | Evaluated second. If there are several such operators, they are evaluated left to right. |
| `+` or `-` | Addition<br>Subtraction | Evaluated last. If there are several such operators, they are evaluated left to right. |

Precedence of arithmetic operators.

## OPERATOR PRECEDENCE: EXAMPLES

- Identify the order of evaluation in the following expressions and Find the answer.

```
1.   2 + 3 + 4 + 5 + 6

2.   2 + 3 * 4 - 12 / 6

3.   12 / (3 * (2 + (5 - 3)))

4.   14 / (1 + 6) - 7 % 2
```

# THANK YOU

SEE YOU NEXT WEEK