# PYTHON:

## SYLLABUS (0-1):

1. Computer system overview
2. Data representation
3. Insight into program execution
4. Computational thinking
5. Python fundamentals
6. Data handling
7. Conditional and interactive statement
8. Tinker
9. Calculator

## Computer system overview:

## Computing thinking and starting with python:

* Computing thinking is divided into 4 parts. They are-
  + Decomposition
  + Patterns recognization
  + Abstraction
  + Algorithms

### Decomposition:

* Decomposition means divide the whole program into small jobs.

### Patterns recognition:

* Pattern recognition defines as the classification of data based on knowledge already gained or on statistical information extracted from their representation.

### Abstraction:

* Abstraction is the process of hiding the real implementation of an application from the user and showing only that the user requires.

### Algorithm:

* Algorithm is step by step procedure, which defines a set of instructions to be executed in a certain order to get the desired output.

## Python style rules and convention:

1. Same block of code will have same place
2. Statement terminated
3. Max length 79 character
4. Use two block line between top-level definitions
5. One blank line between methods
6. White space around operator
7. Case sensitive
8. Doc string (‘’’)
9. Naming convention

## Data handling:

* Data type
* Mutable/immutable
* Operator
* Expression
* Working with standard python libraries

### Data type:

* Number
* String
* List
* Tuples
* Dictionaries
* Set

#### Number:

Numbers are

* Integer
* Float point
* Boolean

#### String:

* The string can be defined as the sequence of characters represented in the quotation marks. In python, we can use single, double, or triple quotes to define a string.
* Example:

A= ‘Binapani’

Print(A)

Output:

Binapani

### List:

* List created as the elements are separated by commas inside a square.
* List is mutable.
* Example:

list=[1,2,3,4,5]

### Tuples:

* Tuple is created as the elements are separated by commas inside a.
* Tuple is immutable.
* Example:

Tuple=(1,2,3,4,5)

### Dictionaries:

* Dictionary is a collection which works on a key-value pair.
* It works like an associated array where no two keys can be same.
* Dictionaries are enclosed by curly braces ({}) and values can be retrieved by square bracket ([]).
* Example:

### Set:

* Set is created as the elements are separated by commas inside curly braces.
* Example:

A={1,2,3,4,5}

Print(A)

Output:

{1,2,3,4,5}

### Mutable:

* If an object can be changed after it is created, is known as mutable.
* It means if we are able to change the value of an object within the same address is known as mutable.
* List and dictionary are mutable.

### Immutable:

* If an object can’t be changed after it is created is known as immutable.
* It means we can’t change value of an object within the same address.
* Integer, float point, Boolean, string, tuples and set are immutable type.

### Operator:

1. Arithmetic operator
2. Relational operator
3. Augmented operator
4. Identity operator
5. Logical operator
6. Bitwise Operator
7. Operator presidency

**Arithmetic operator:**

* Arithmetic operators are used to perform arithmetic operations between two operands. It includes +(addition), - (subtraction), \*(multiplication), /(divide), %(reminder), //(floor division), and exponent (\*\*).
* Consider the following table for a detailed explanation of arithmetic operators.

|  |  |
| --- | --- |
| **+ (Addition)** | It is used to add two operands. For example, if a = 20, b = 10 => a+b = 30 |
| **- (Subtraction)** | It is used to subtract the second operand from the first operand. If the first operand is less than the second operand, the value result negative. For example, if a = 20, b = 10 => a - b = 10 |
| **/ (divide)** | It returns the quotient after dividing the first operand by the second operand. For example, if a = 20, b = 10 => a/b = 2 |
| **\* (Multiplication)** | It is used to multiply one operand with the other. For example, if a = 20, b = 10 => a \* b = 200 |
| **% (reminder)** | It returns the reminder after dividing the first operand by the second operand. For example, if a = 20, b = 10 => a%b = 0 |
| **\*\* (Exponent)** | It is an exponent operator represented as it calculates the first operand power to second operand. |
| **// (Floor division)** | It gives the floor value of the quotient produced by dividing the two operands. |

**Relational operator:**

* Comparison operators are used to comparing the value of the two operands and returns Boolean true or false accordingly. The comparison operators are described in the following table

|  |  |
| --- | --- |
| **Operator** | **Description** |
| == | If the value of two operands is equal, then the condition becomes true. |
| != | If the value of two operands is not equal then the condition becomes true. |
| <= | If the first operand is less than or equal to the second operand, then the condition becomes true. |
| >= | If the first operand is greater than or equal to the second operand, then the condition becomes true. |
| > | If the first operand is greater than the second operand, then the condition becomes true. |
| **<** | If the first operand is less than the second operand, then the condition becomes true. |

**Logical operator:**

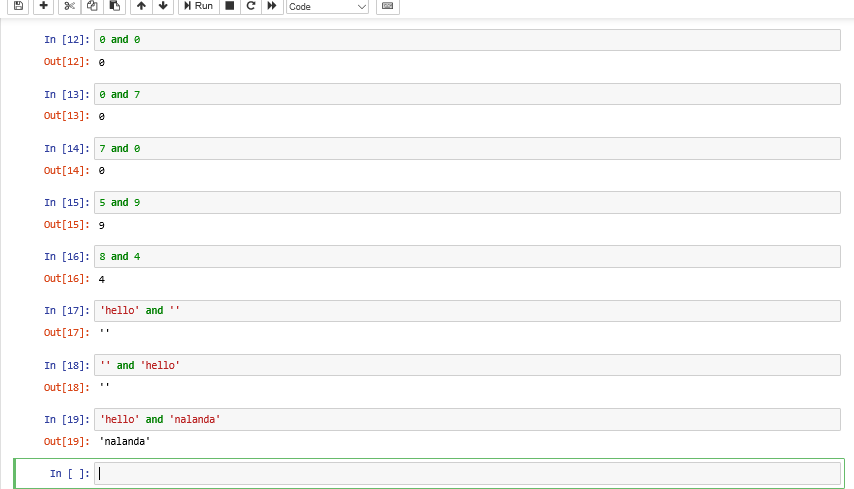
* Logical operators allows to make decision on multiple conditions.
* Logical operators are AND, OR & NOT.

**OR OPERATOR:**

* In an expression x OR y, if first operator has false value, then return second operator as result, otherwise first.
* Example:

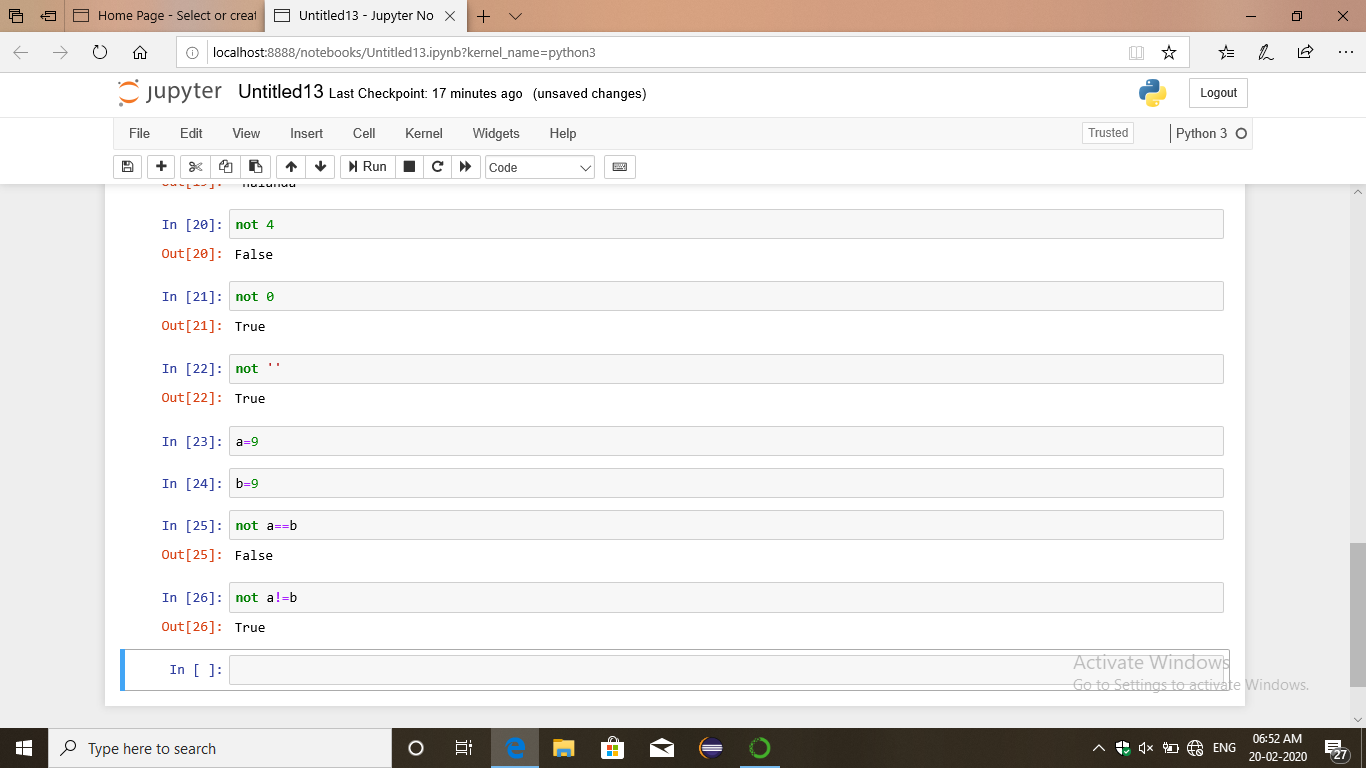


**AND OPERATOR:**

* If first operator is false, then return first as result otherwise 2nd operator.
* Example: 

**NOT OPERATOR:**

* Example:



**NOTE:**

## NOT, OR, AND has lesser priority than other operator.

## Identity Operators

|  |  |
| --- | --- |
| **Operator** | **Description** |
| is | It is evaluated to be true if the reference present at both sides point to the same object. |
| is not | It is evaluated to be true if the reference present at both side do not point to the same object. |

**BITEWISE OPERATOR:**

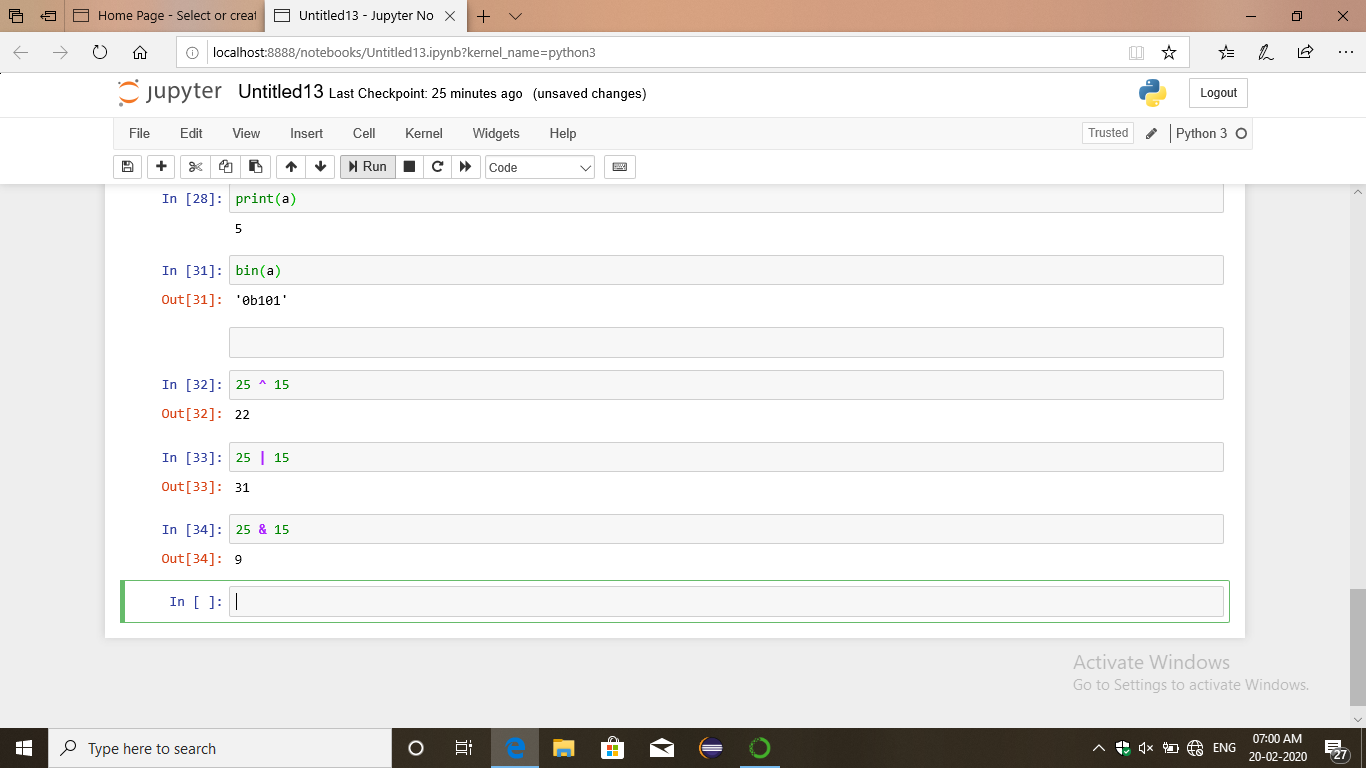
& =AND

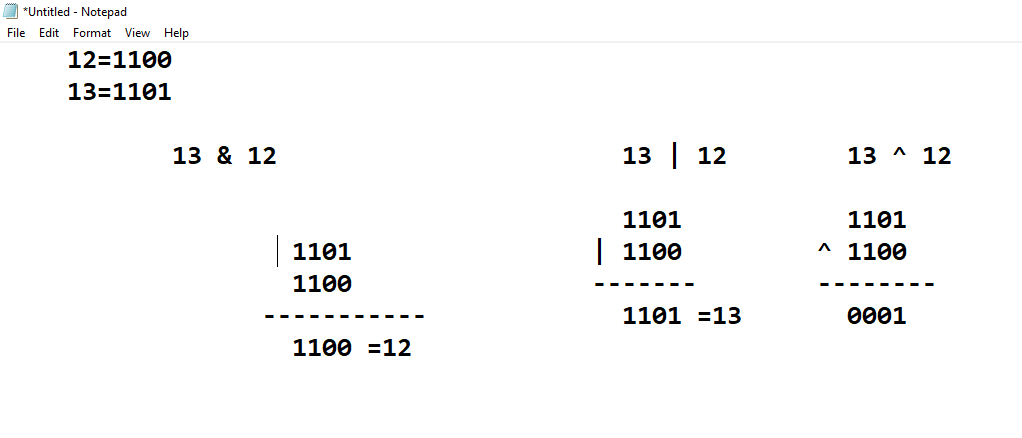
| = OR

^ =XOR

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | **Operator** | **Description** | | & (binary and) | If both the bits at the same place in two operands are 1, then 1 is copied to the result. Otherwise, 0 is copied. | | | (binary or) | The resulting bit will be 0 if both the bits are zero otherwise the resulting bit will be 1. | | ^ (binary xor) | The resulting bit will be 1 if both the bits are different otherwise the resulting bit will be 0. | | ~ (negation) | It calculates the negation of each bit of the operand, i.e., if the bit is 0, the resulting bit will be 1 and vice versa. | | << (left shift) | The left operand value is moved left by the number of bits present in the right operand. | | >> (right shift) | The left operand is moved right by the number of bits present in the right operand. | |

* Example:





**OPERATOR PRESEDENCE:**

* ()
* \*\*
* ~x
* fx,-x
* \*, /, //, %
* +
* &
* ^
* /
* <, <=, >, >=
* Logical operator
* The precedence decreases top to bottom.

**Type casting:**

* Erg

### Standard Libraries:

## Conditional interactive:

* If:
* If:

else:

* If:

elif:

else:

* If:

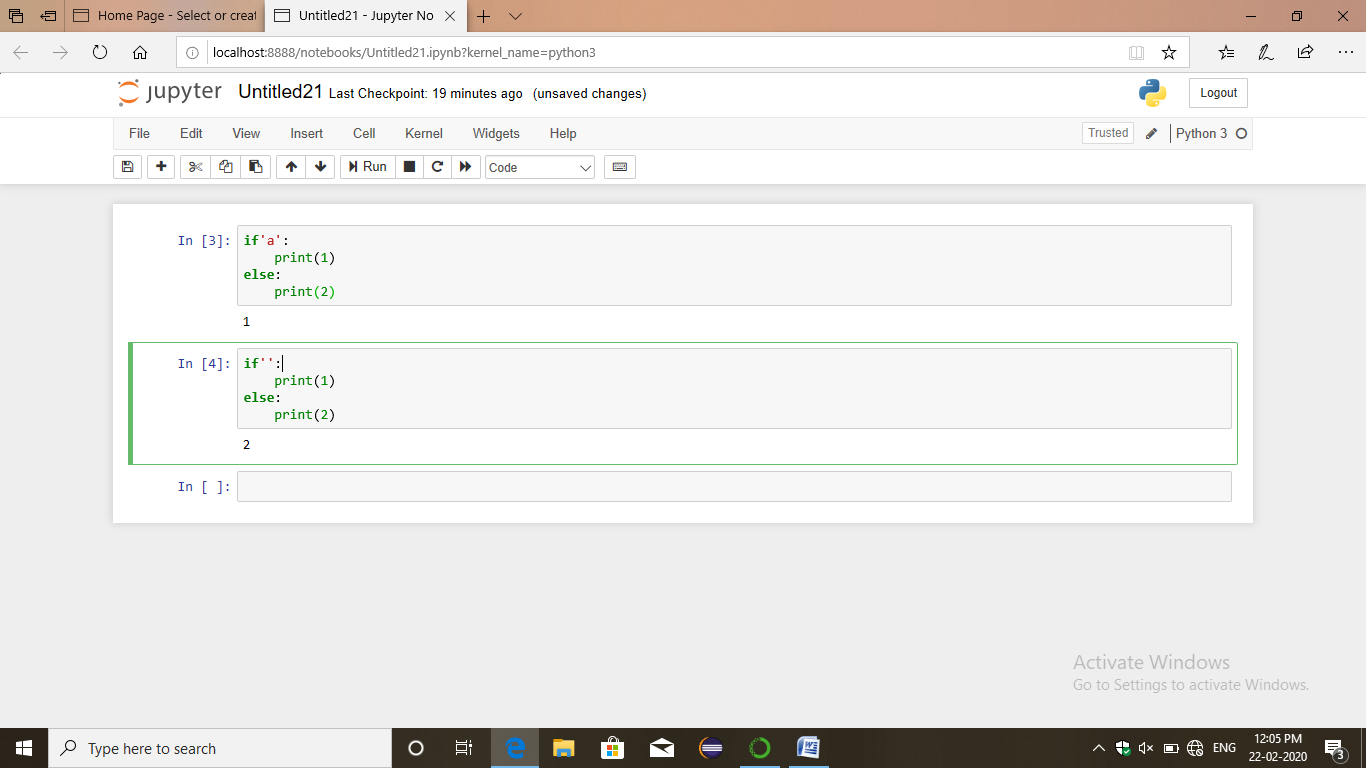
If:

else:

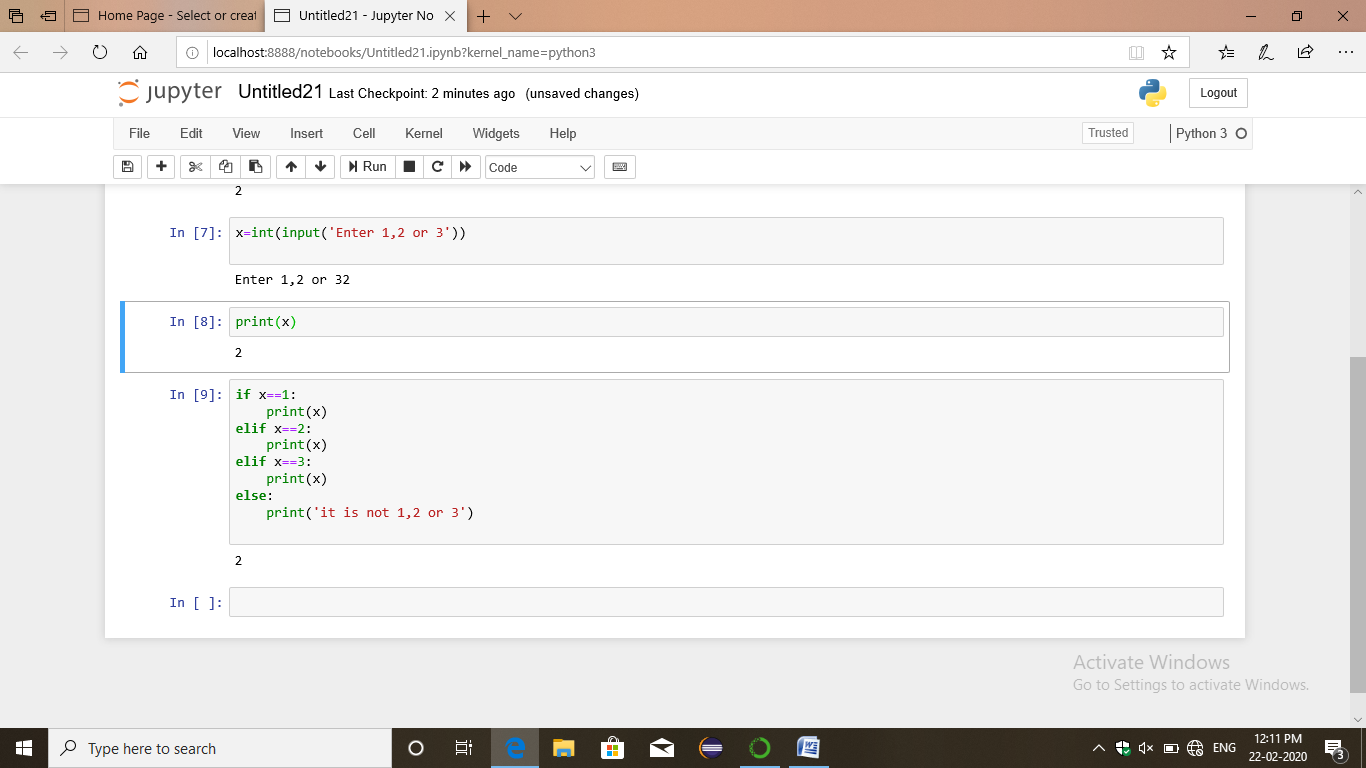
else:

* Range()
* Loops
* Loop else
* Break vs. continue

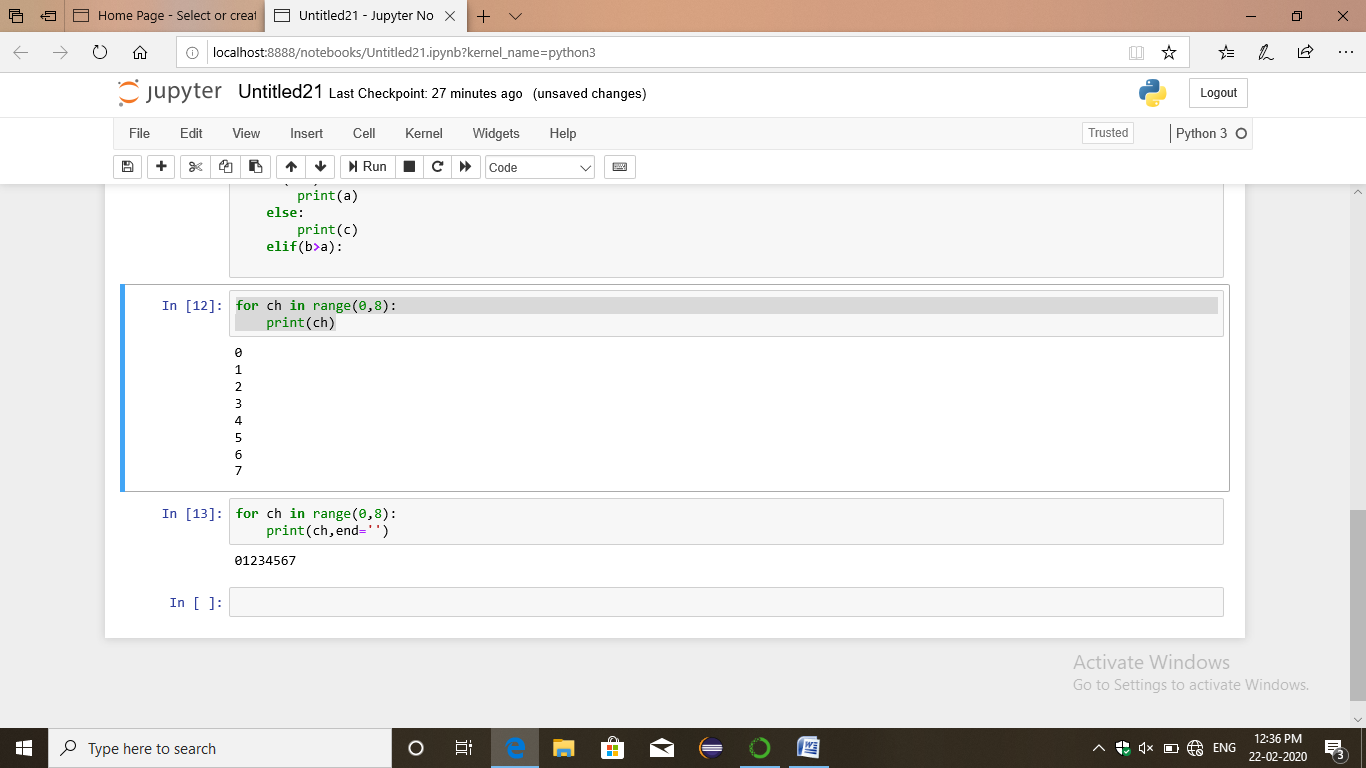
### If:



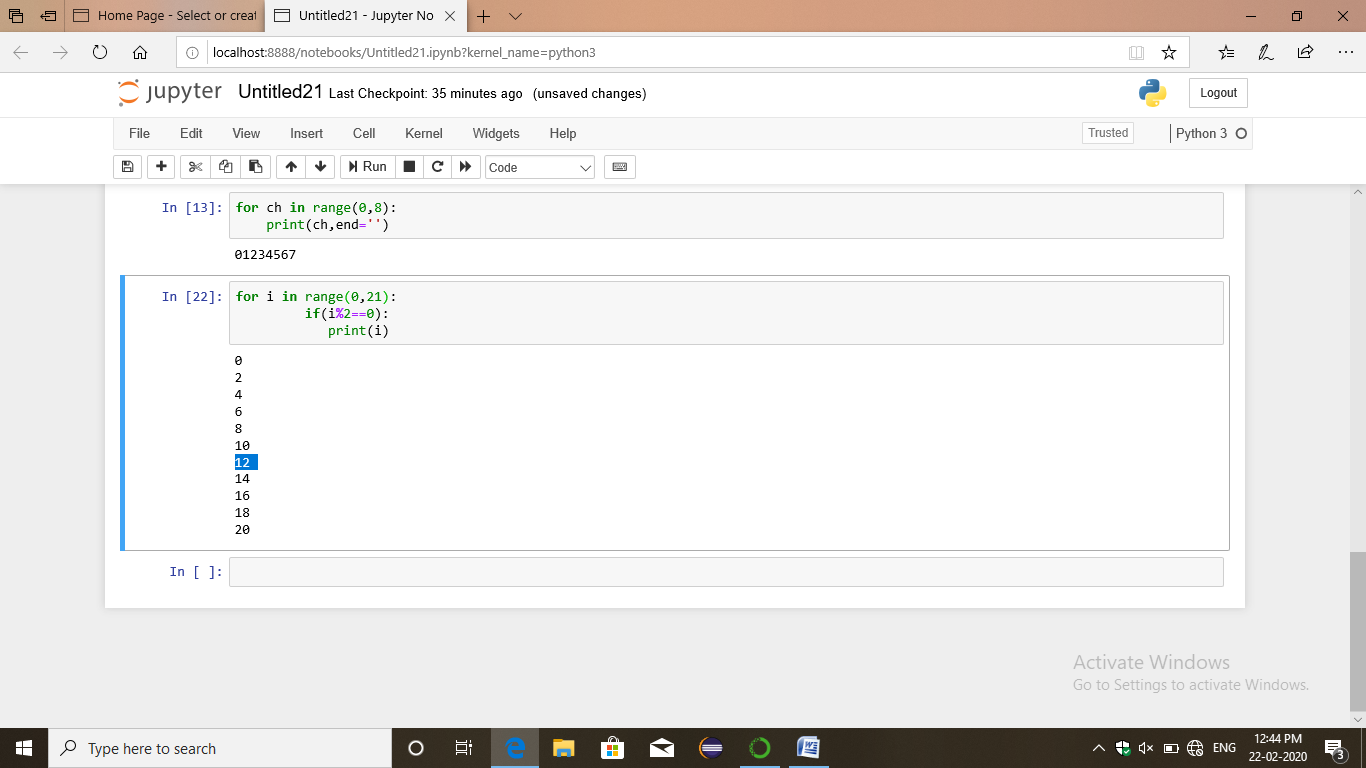
### If else:



### Range():



### Loop:



### Loop else:

### Break statement:

* The break statement terminates the loop containing it. Control of the program flows to the statement immediately after the body of the loop.
* If break statement is inside a nested loop, break will terminate the innermost loop.

### Continue:

* The continue statement is used to skip the rest of the code inside a loop for the current interaction only.
* Loop does not terminate but continue on with the next iteration.