Python for Beginners

Glimpse of last lecture

Introduction (an overview to course)

- ~~Getting started "Hello World"~~
- ~~Keywords and identifiers~~
- ~~Comments~~ and Statements
- ~~Variables and assignments~~
- Data types
- Flow control
- Methods and Functions
- Reading and Writing files
- Modules and import
- Object Oriented Programming

Data types

- ~~Numbers~~
- ~~List~~
- ~~Tuple~~
- ~~String~~
- ~~Dictionary~~
- Set

Program Flow Control in Python

- *If* statement
- Elif statement
- More on if, elif and else
- *For* loop
- *While* loop
- Useful operators

Methods and Functions

- Defining a function
- Flow when calling a function
- Parameters and arguments
- Global/local
- Functions calling functions

Reading/Writing Files

- Files and directories in Python
- Reading from a file
- Parsing data
- Printing data to external file

Modules and Import

- Standard Python library
- Datetime module
- Math and Random module
- Generators and decorators
- NumPy, Pandas, Matplotlib (basic uses)

Object Oriented Programming

- Introduction to OOPs
- Attributes and Class keywords
- Inheritance and Polymorphism

Statistical analysis of data with Python

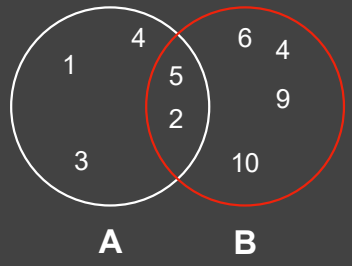
Sets

'Set' in python offers to store unordered collection of elements but without duplicity.

Sets

'Set' in python offers to store unordered collection of elements but without duplicity.

'Set' objects also support the basic mathematical functionalists :
union, intersection, difference.



	Python equivalent	
A ∪ B =	A | B	A ^ B (in either of them but not in both)
A ∩ B =	A & B	
A - B =	A - B	

'Set' can be constructed using two ways :

Using set( ) function

```
x = set('Mango', 'Banana', 'Apple')
y = set('tree', 'house', 'Banana')
```

Like dictionary (with keys only) using curly braces

```
SetA= {1,2,3,4,5}      Set1 = {'Mango', 'Banana', 'Apple'}
SetB= {2,5,6,9,10}     Set2 = {'tree', 'house', 'Banana'}
```

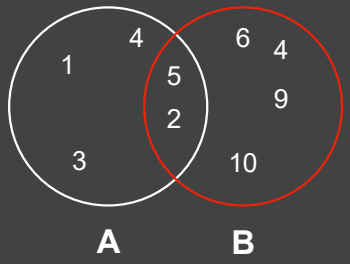
Hands-On 9

```
A = {'Mango', 'Banana', 'Apple'}
B = {'tree', 'house', 'Banana'}

A | B      A & B

A - B      A ^ B
```

Sets

'Set' in python offers to store unordered collection of elements but without duplicity.

'Set' objects also support the basic mathematical functionalists :
union, intersection, difference.



A ∪ B = 1, 3, 4, 5, 2, 6, 9, 10

A ∩ B = 5, 2

A - B = 1, 3, 4

Python equivalent	
A | B	
A & B	
A - B	

A ^ B
(in either of them but not in both)

'Set' can be constructed using two ways :

Using set( ) function

```
x = set(('Mango', 'Banana', 'Apple'))
y = set(('tree', 'house', 'Banana'))
```

Like dictionary (with keys only) using curly braces

```
SetA= {1,2,3,4,5}      Set1 = {'Mango', 'Banana', 'Apple'}
SetB= {2,5,6,9,10}     Set2 = {'tree', 'house', 'Banana'}
```

Hands-On 9

A = {'Mango', 'Banana', 'Apple'}
B = {'tree', 'house', 'Banana'}

A | B A & B

A - B A ^ B

Frequently used methods on Sets

add( )	Adds element to the set
clear( )	Remove all the elements from set
remove( )	Remove the specified element
difference( )	Difference b/w two or more sets
intersection( )	Intersection of two sets (common element)
union( )	Union of sets
update( )	Try to add list into Sets

Hands-On with sets

- How to check if an element/item is in a given set : ~~index~~, ~~key~~

Using 'in' keyword

```
my_firstSet = {34, 22, 12, 100, 20000, 1, 2, 5, 8, 213, 35, 'banana', 'mango', 'orange'}
print('mango' in my_firstSet)
```

difference( ) : return the difference between sets
difference_update( ) : removes the same items in two sets

intersection( ) : return interaction (common)of sets
intersection_update( ): remove the *uncommon item* in sets

isdisjoint( ) : *return true if no-common item in sets*, otherwise false

issubset( ) : one set has all elements of other set with additional elements
issuperset( ) : opposite to issubset

Hands-On with sets

- How to check if an element/item is in a given set : ~~index~~, ~~key~~

Using 'in' keyword

```
my_firstSet = {34, 22, 12, 100, 20000, 1, 2, 5, 8, 213, 35, 'banana', 'mango', 'orange'}
my_secondSet = { 22, 100, 213, 'banana'}
my_thirdSet = { 34, 100, 213, 'grapes'}
```

difference( ) : return the difference between sets
difference_update( ) : removes the same items in two sets

intersection( ) : return interaction (common)of sets
intersection_update( ): remove the *uncommon item* in sets

isdisjoint( ) : *return true if no-common item in sets*, otherwise false

issubset( ) : one set has all elements of other set with additional elements
issuperset( ) : opposite to issubset

Comparison b/w data types

Based on quotes, braces and brackets : ‘ ’ or “ ”, ( ), { }, [ ]

	Mutability	Mixed data	Duplicacy	Ordered /Indexing
String **(single Or double quotes)**	No	No	Yes	Yes
List[ ]	Yes	Yes (list, set, tuple, int and dict.)	Yes	Yes
Tuple( )	No	Yes (list, set, tuple, int, dict.)	Yes	Yes
Dictionary{ }	Yes Keys are immutable type	Yes **Key can be int, str and tuple** but values can be of any data type	No	No
Set( )	No	Yes Only (int, str, tuple) Not (list, set,dict)	No	No

Python - statement

A statement can be understood as an instruction that the interpreter (in our case Python interpreter) can execute.

You have been working with two types of statements : assignment statement , print statement

"Remember: assignment statement doesn't produce any result"

On contrary, comments are not executed by interpreter.

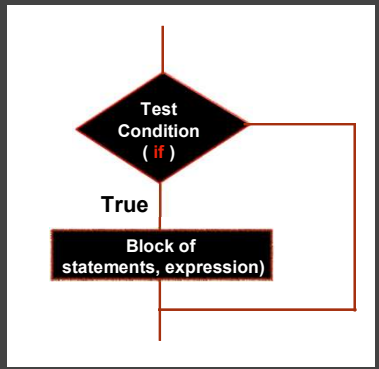
Python - Expressions

An expression may consists of ***variables***, ***operators*** and ***values***, which the Python interpreter evaluates and give the result.

If, elif, else statements (decision making building blocks in Python)

If, if-else, elif statements (decision making building blocks in Python)

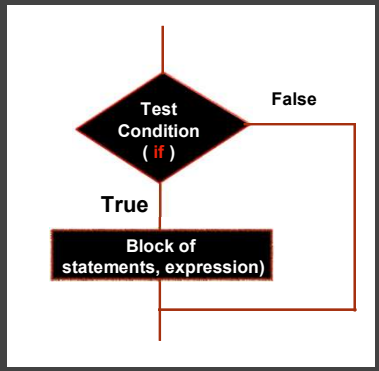
If Statement




```
x = 10
If x > 10:
    print(“ Number is Greater”)
```

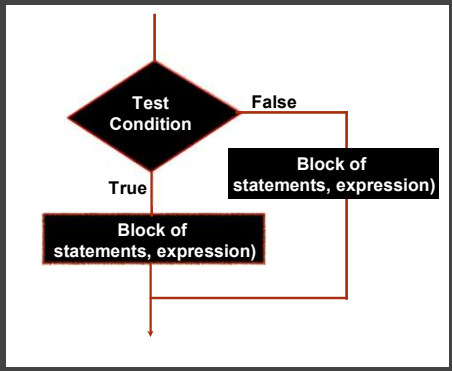
If, if-else, elif statements (decision making building blocks in Python)

If Statement




```
x = 10
If x > 10:
    print(" Number is Greater")
```

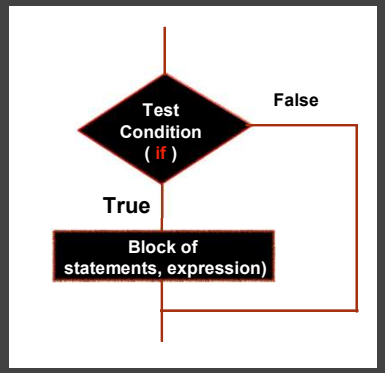
If-else Statement




```
If x > 10:
    print("Number is Greater")
else:
    print("Number is smaller")
```

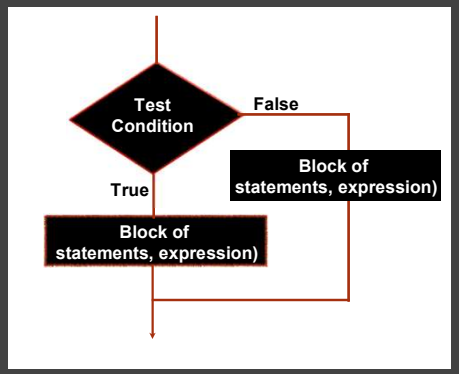
If, if-else, elif statements (decision making building blocks in Python)

If Statement



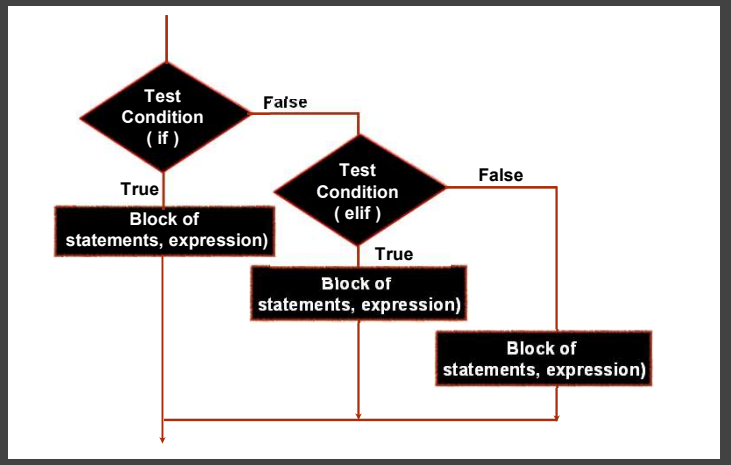

```
x = 10
If x > 10:
    print(" Number is Greater")
```

If-else Statement




```
If x > 10:
    print("Number is Greater")
else:
    print("Number is smaller")
```

If-elif Statement




```
If x > 10:
    print("Number is Greater")
elif x == 10:
    print("Number is smaller")
else:
    print("Number is smaller")
```

If, if-else, elif statements (decision making building blocks in Python)

Hands-On

Try to **define a variable**
with a number and
with a string

and apply,
if, elif, else statements

Hint -

x = 100; (use if, elif, else)

Name = 'Krakow' (use if-else)

If, if-else, elif statements (decision making building blocks in Python)

Hands-On

Try to **define a variable**
with a number and
with a string

and apply,
if, elif, else statements

Hint -

x = 100; (use if, elif, else)

Name = 'Krakow' (use if-else)

Using if-else statement with lists

```
Days = ['Monday', 'Tuesday', 'Wednesday']

If 'Tuesday' in Days:

    print('yes')

else:

    print('no')
```

Modify list:
Hint: Use append(item), insert(index,item)

If, if-else, elif statements (decision making building blocks in Python)

Hands-On

Multiple conditional statement using elif :

```
if today == 'Monday':
    print("Office day")

elif today == 'Tuesday':
    print("Work from home day")

elif today == 'Wednesday':
    print("Find excuse for not going to office")

elif today == 'Thursday':
    print("Work from home day")

elif today == 'Friday':
    print("Half day office")

else:
    print("Weekend, official-OFF day")
```

Single line conditional statement

```
Day = 'Friday'

if Day == 'Monday': print("Yes, today is Friday")
```

Nested if-else statements

If, if-else, elif statements (decision making building blocks in Python)

Hands-On

Nested if-else conditions

Initial inputs

```
Shop = 'Fruits'
Type = ‘sweet'
color = 'Yellow'
```

Change inputs values and try to guess the expected answers

```
if Shop == 'Fruits':
    print(" Plan to buy Fruits today")

    if Type == 'sweet':
        print(" Fruits are sweet")
    elif Type == 'good':
        print("Not sweet, but still good fruits")

                                        Hint: Nested to last elif, indentation gives the clue
        if color == 'Yellow':
            print("Fruits are either sweet or good and color is yellow")
        else:
            print("Fruits are niether sweet, nor yellow")


    else:
        print(" Fruits are not sweet")

else:
    print(" Not buying fruits today")
```

Python - Operators

Operators allow to perform operations on operands

Variables or values of variables

In Python,

Operators are divided in the following groups

- Arithmetic operators
- Assignment operators
- Comparison operators
- Logical operators
- Identity operators
- Membership operator
- Bitwise operator

Operators

In Python,

Operators are divided in the following groups

- Arithmetic operators
- Assignment operators
- Comparison operators
- Logical operators
- Identity operators
- Membership operator
- Bitwise operator

Allow to apply mathematical operations with num. values : x = 9 , y = 2

Addition (+)	Adding the operands	x + y
Subtraction (-)	Subtracts R.H.S operand from L.H.S	x - y
Multiplication (*)	Multiply the operands	x * y
Division (/)	Divide L.H.S by R.H.S operand	x / y
Modulus (%)	Divide L.H.S by R.H.S, return remainder	x % y
Exponent (**)	Divide L.H.S by R.H.S, return remainder	x ** y
Floor division (//)	Divide L.H.S by R.H.S, round-up the quotient to nearest integer	x // y

Try to execute these Statements with x = 13, y = 3

Operators

In Python,

Operators are divided in the following groups

- Arithmetic operators
- Assignment operators
- Comparison operators
- Logical operators
- Identity operators
- Membership operator
- Bitwise operator

Allow to assign variables to variables; lets x = 9 , y = 2

(=)	**x = y**
(+=)	**x += y**
(-=)	**x -= y**
(*=)	**x *= y**
(/=)	**x /= y**
(%/=)	**x %= y**
(**=)	**x **= y**
(//=)	**x //= y**

Try to answer —

Operators

In Python,

Operators are divided in the following groups

- Arithmetic operators
- Assignment operators
- Comparison operators
- Logical operators
- Identity operators
- Membership operator
- Bitwise operator

Comparison operators allow to compare two operands; lets x = 9 , y = 2

Equal to (==)	**x == y**
Not equal to (!=)	**x != y**
Greater than (>)	**x > y**
Smaller than (<)	**x < y**
Greater or equal (>=)	**x >= y**
Smaller or equal (<=)	**x <= y**

Try to answer —

Operators

In Python,

Operators are divided in the following groups

- Arithmetic operators
- Assignment operators
- Comparison operators
- Logical operators
- Identity operators
- Membership operator
- Bitwise operator

Allow to combine multiple conditional statement : x = 9 , y = 2

and	**x > 5 and y < 4**	**Both statements should true *to return True***
or	**x > 5 or y < 2**	**If one of the statements is true, *return True***
not	**not (result) not(x > 5 and y < 4)**	**Reverse the state of result: False->Ture, True->False**

Operators

In Python,

Operators are divided in the following groups

- Arithmetic operators
- Assignment operators
- Comparison operators
- Logical operators
- Identity operators
- Membership operator
- Bitwise operator

Allow to *compare the memory location* of **objects** :

string, list[ ],…

is	**x is y**	**True, if both variables/object are same**
is not	**x is not y**	**True, if both variables/object are not same (not values, but locations)**

```
x = "'little','big'"
y = "'little','big'"
z = x
x == y
```

Try to answer —

Operators

In Python,

Operators are divided in the following groups

- Arithmetic operators
- Assignment operators
- Comparison operators
- Logical operators
- Identity operators
- Membership operator
- Bitwise operator

Allows to test if a strings, lists or tuples are members of other sequence (string, list or tuples)

in	Day in x	True, if Day(sequence) is in x
not in	Day2 in x	True, if Day2(sequence) is in x, otherwise false

x = ['Sunday', 'Monday', 'Tuesday', 'Wednesday']
Day = 'Monday'
Day2 = ['Thursday', 'Friday', 'Saturday']

Operators

..	32	16	8	4	2	1	
	..	..	0	0	0	0	For 0
			0	1	1	1	For 7
			1	1	1	1	For 15

In Python,

Operators are divided in the following groups

- Arithmetic operators
- Assignment operators
- Comparison operators
- Logical operators
- Identity operators
- Membership operator
- Bitwise operator

a & b AND	Bitwise and
a | b OR	Bitwise or
a ^ b XOR	Bitwise exclusive
~ b NOT	Bitwise not
a << n left shift	shift a left by n(2) bits
	shift a right by n(2) bits

Operators

..	32	16	8	4	2	1	
	..	..	0	0	0	0	For 0
			0	1	1	1	For 7
			1	1	1	1	For 15

Lets take few variables a = 60, b = 13

In Python,

Operators are divided in the following groups

- Arithmetic operators
- Assignment operators
- Comparison operators
- Logical operators
- Identity operators
- Membership operator
- Bitwise operator

a & b AND	Bitwise and
a | b OR	Bitwise or
a ^ b XOR	Bitwise exclusive
~ b NOT	Bitwise not
a << n left shift	shift a left by n(2) bits
	shift a right by n(2) bits

Operators

In Python,

Operators are divided in the following groups

- Arithmetic operators
- Assignment operators
- Comparison operators
- Logical operators
- Identity operators
- Membership operator
- Bitwise operator

	a & b	AND	Bitwise **and**
	a | b	OR	Bitwise **or**
	a ^ b	XOR	Bitwise **exclusive**
	~ b	NOT	Bitwise **not**
	a << n	left shift	shift a left by n(2) bits
			shift a right by n(2) bits

..	32	16	8	4	2	1	
..	..	0	0	0	0	For 0	
		0	1	1	1	For 7	
		1	1	1	1	For 15	

Lets take few variables a = 60, b = 13

$a = (1\,1\,1\,1\,0\,0)_2$

$b = (0\,0\,1\,1\,0\,1)_2$

Operators

In Python,

Operators are divided in the following groups

- Arithmetic operators
- Assignment operators
- Comparison operators
- Logical operators
- Identity operators
- Membership operator
- Bitwise operator

..	32	16	8	4	2	1
	..	..	0	0	0	0 For 0
			0	1	1	1 For 7
			1	1	1	1 For 15

Lets take few variables a = 60, b = 13

$a = (1\,1\,1\,1\,0\,0)_2$

$b = (0\,0\,1\,1\,0\,1)_2$

- **a & b** AND — Bitwise and: 0 0 1 1 0 0 = 12
- **a | b** OR — Bitwise or: 1 1 1 1 0 1 = 61
- **a ^ b** XOR — Bitwise exclusive: 1 1 0 0 0 1 = 49
- **~ b** NOT — Bitwise not: 1 1 0 0 1 0
- **a << n** left shift — shift a left by n(2) bits
- shift a right by n(2) bits

Operators precedence follow in Python

**	Exponential or raise to power
~, +, -	Complement, unary plus, unary minus
*, /, %, //	Multiply, divide, modulus, floor division
+, -	Addition and subtraction
>>, <<	Bitwise : right , left
&	Bitwise : AND
^, |	Bitwise : exclusive OR , OR
<=, <, >, >=	Comparison operators
=, %=, /=, //=, -=, +=, *=, **=	Assignment operators
is, is not	Identity operators
in, not in	Membership operator
not, or, and	Logical operators

A = 20, B = 10, C = 15, D = 2

((A + B) * (C / D)) ** D ?

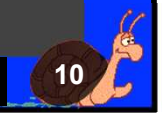
Looping in Python

After decision making,

LOOPs are another fundamental part of any fundamental language, that allows to execute the the same action several times.

Print integer numbers between 1 to 10 ?

```
print(1)
1
print(2)
2
print(3)
3
.
.
.print(10)
10
```

Looping in Python

After decision making,

LOOPs are another fundamental part of any fundamental language, that allows to execute the the same action several times.

Print integer numbers between 1 to 10 ?

```
print(1)
1
print(2)
2
print(3)
3
.
.
.print(10)
10
```

Using LOOP :

```
for x in range(1,11):
    print(x)
```

Print only odd / Even numbers ?

Looping in Python

After decision making,

LOOPs are another fundamental part of any fundamental language, that allows to execute the the same action several times.

Print integer numbers between 1 to 10 ?

```
print(1)
1
print(2)
2
print(3)
3
.
.
.print(10)
10
```

Using LOOP :

```
for x in range(1,11):
    print(x)
```

Print only odd / Even numbers ?

```
for letter in "coffee":
    print(letter*10)
```

What it does ?

Looping in Python

After decision making,

LOOPs are another fundamental part of any fundamental language, that allows to execute the the same action several times.

Print integer numbers between 1 to 10 ?

```
print(1)
1
print(2)
2
print(3)
3
.
.
.print(10)
10
```

Using LOOP :

```
for x in range(1,11):
    print(x)
```

Print only odd / Even numbers ?

```
for letter in "coffee":
    print(letter*10)
```

What it does ?

Range in for loops

As per documentation, ‘ Range’ represents an immutable sequence of numbers and is commonly used for looping a specific number of times

Python ranges can be used in multiple forms

ranges(final) for x in range(11):

ranges(initial,final) for x in range(4,11):

ranges(initial,final,step)

- for x in range(4,11,1):
- for x in range(4,11,2):
- for x in range(11,1,-2):

Steps up / down

Hands - On

What are expected numbers generated by the following range ?

range(2,19) ?

What are expected numbers generated by the following range ?

range(5) ?

What are expected numbers generated by the following range ?

```
for x in range(18,0,-3):
print(x)
```

- Write a code using for loop to add odd/even numbers from range 1-100
- Modify code to print only odd numbers divisible by 3 from range 1-100
- Modify code to print only number which give remainder 1 after dividing by 4 from range 1-100

while loops

With while loop, one can do almost all actions which for loops can do, **with additional features**.

As the name suggests, it invoke the conditional statement

Syntax:

while *conditional_statement*:
Block_of_statements,
unless conditional statement become false

Difference between *while* and *for* loop

```
x = 1
While x <11:
    print(x)
    x+=1
```

```
for x in range(1,10):
    print(x)
```

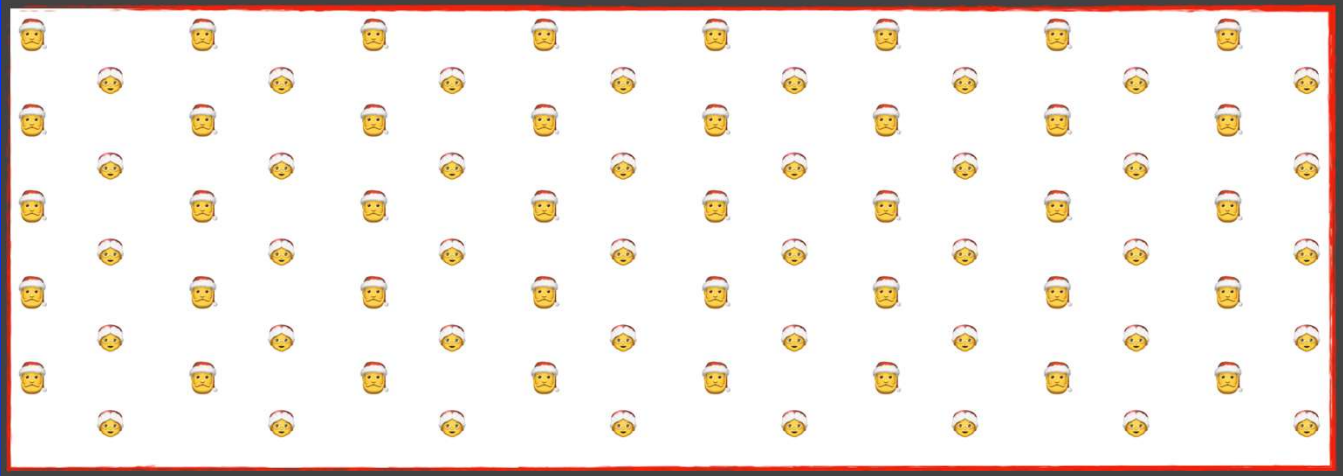
Hands-On

```
for num in range(1,11):
    print("\U0001f332")
```

- Print only, when num is even
- Print only, when num is odd
- Print both, even and odd

Print

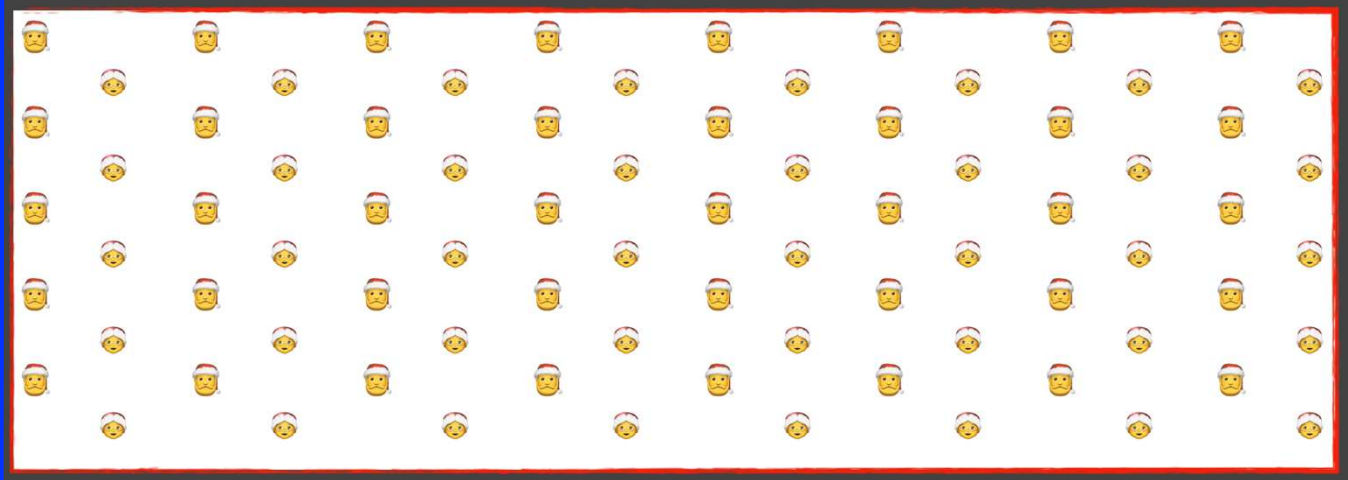
Hands-On - print



\U0001f385 

\U0001f936 

Hands-On - print



\U0001f385

\U0001f936

```
for num in range(1,11):

    if num % 2 == 0:

        print("    \U0001f936\t\t\U0001f936\t\t\U0001f936\t\t\U0001f936\t\t".expandtabs(4))

    elif num%2 ==1: print("\U0001f385\t\U0001f385\t\U0001f385\t\U0001f385\t".expandtabs())
```

Thank you !!!