
Python for Everybody

Course 1:

Getting started with python

Welcome to python programming

- Python was named after a play.
- Python is a programming language.
- Python is an easy to understand programming language.
- Python is an lightweight and a versatile language.
- Vocabulary/words in python:variables and reserved words
- Sentence structure:valid syntax patterns
- Story structure:constructing a program for a purpose.

Reserved words

Reserved Words

- You cannot use **reserved words** as variable names / identifiers

False	await	else	import	pass
None	break	except	in	raise
True	class	finally	is	return
and	continue	<u>for</u>	lambda	try
as	def	from	nonlocal	while
<u>assert</u>	del	global	not	with
async	elif	<u>if</u>	or	yield

Sentences and lines

Sentences or Lines

x = 2



Assignment statement

x = **x** + 2



Assignment with expression

print(**x**)



Print function

Variable

Operator

Constant

Function

- Interactive:
- You type directly to python one line at a time and it responds that is CMD.
- Scripts:
- You write sequence of statements into a file using text editor and tell python to execute the statements in the file.

EXPRESSIONS and VARIABLES:(CHAPTER 2)

- **Constants:**

- Fixed values such as numbers, letters, and strings, are called “constants” because their values do not change.
- Numeric constants are as you expect.
- String constants use single quote(') or double quote(“).

- **Variables:**

- A variable is a named place in a memory where a programmer can store data and later retrieve the data using the variable name.
- Programmers get to choose the names of the variables.
- We can update the variables later as well.
- Naming rules of variables:
- Must start with letter or underscores
- Must consist of letter, number, and an underscore.
- Case sensitive.

Good:	spam	eggs	spam23	_speed
Bad:	23spam	#sign	var.12	
Different:	spam	Spam	SPAM	

- **Assignments statements:**
- We assign a value to a variable using the assignment statement(=)
- An assignment statement statements consists of an expression on the right hand side and a variable to store the result.
- For variables name we generally use mnemonic names (the name that make sense with the relevance with the context)
- The numeric expression uses different operators for solving the expressions.
- The operators are used in accordance of the precedence and associativity.

Operator	Operation
+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Power
%	Remainder

Operator Precedence Rules

Highest precedence rule to lowest precedence rule:

- Parentheses are always respected
- Exponentiation (raise to a power)
- Multiplication, Division, and Remainder
- Addition and Subtraction
- Left to right

Parenthesis
Power
Multiplication
Addition
Left to Right



- “Type” is used for checking the type of the variables.

- Type of data value matters a lot as we can't add 1 to a string.
- If we want to get the type of the data value by using type function we can do it.

```
>>> eee = 'hello ' + 'there'
>>> eee = eee + 1
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: Can't convert 'int' object to str implicitly
>>> type(eee)
```

- **Types of numbers:** Numbers have two main type:
 - Integers: are either negative or positive number without the fractional part.
 - Floating point: have fractional parts.
- **Type conversion:**
 - When we put an integer and floating point in an expression an integer is explicitly converted into a floating point.
 - We can do type of data conversion by int() and float()
 - By using this you can convert your integer to floating point and vice versa.

```
>>> print(float(99) + 100)
199.0
>>> i = 42
>>> type(i)
<class 'int'>
>>> f = float(i)
>>> print(f)
42.0
>>> type(f)
<class 'float'>
```

- Integer division produces floating point division.
- Integer division is differently done in Python2 and Python3.
- In python2 ,the integer division returns floor value instead of the decimal value
- In python3 it tends to return floating point value.
- **String conversion:**
- You can also convert string data into an int and float by using int(),float()
- But it is only possible when your contains numeric value.

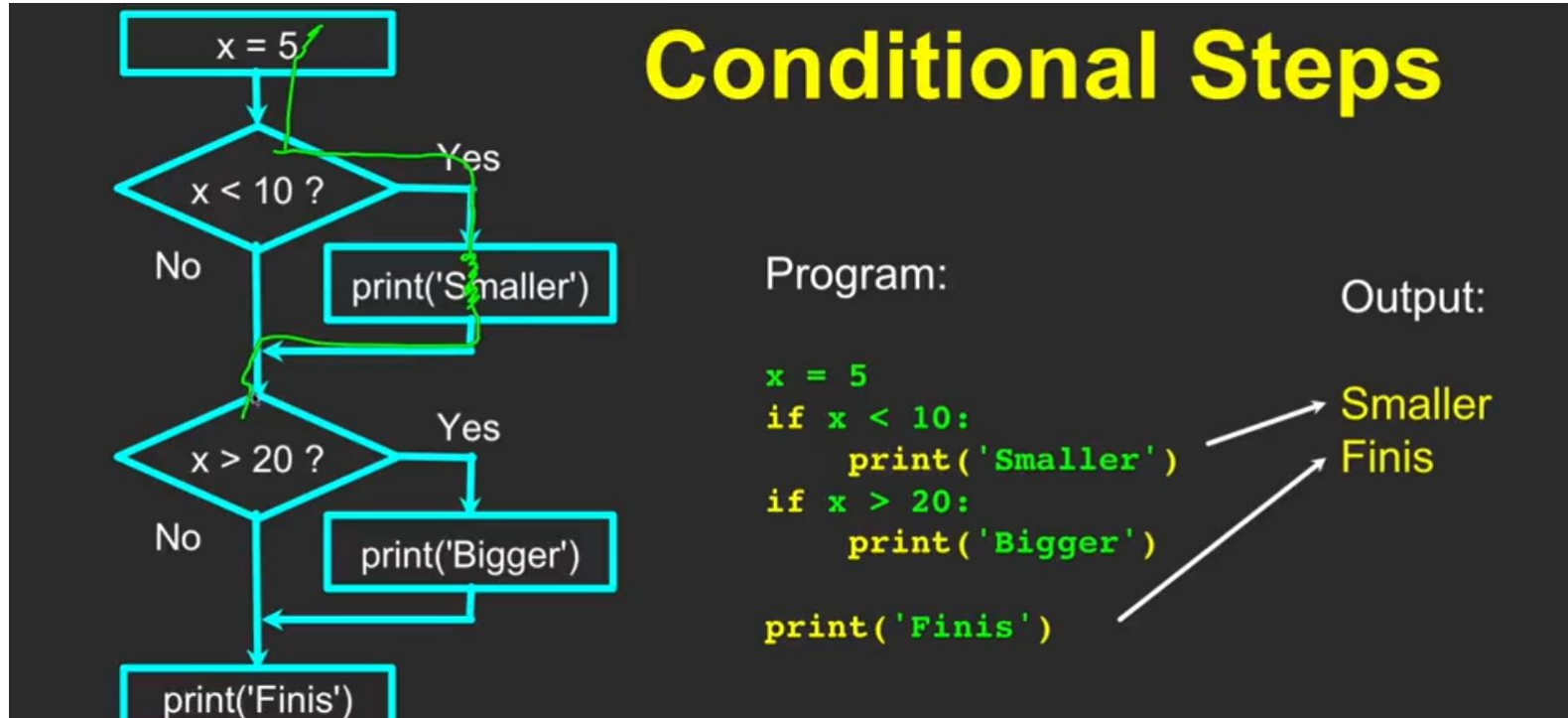
```
>>> sval = '123'
>>> type(sval)
<class 'str'>
>>> print(sval + 1)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: Can't convert 'int' object
to str implicitly
>>> ival = int(sval)
>>> type(ival)
<class 'int'>
>>> print(ival + 1)
124
>>> nsv = 'hello bob'
>>> niv = int(nsv)
Traceback (most recent call last):
```

- **Input function:**
- We can instruct python to pause and read the data from the user using the input() function
- Input function is used for taking input data from user.
- Input function always return string.

- **Comments:**
- Comments are a way to add text in the code that is ignored by python during execution.
- Comments are added by using **#** symbol in your python code.
- Comments are used for documentation purpose for future use of the code.
- Comments are used for giving an idea about the sequence of code.

CONDITIONAL STATEMENTS(CHAPTER 3)

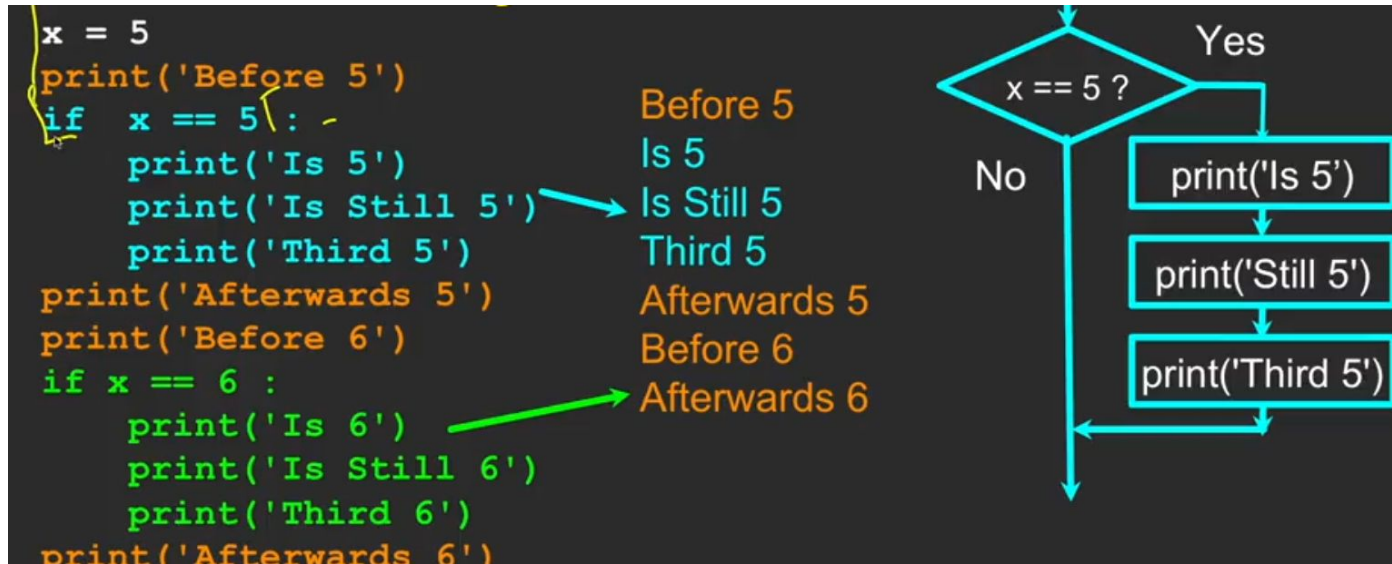
- Conditional statements are the statements that executes conditionally.
- if , else ,elif are the conditional statements that are used for checking conditions.
- If statements checks for a conditions and depending on the output we decide whether the line independent next to it is going to be executed or not.



- During conditional statements we tend to use indenting .
- Indenting notation is used for specifying the piece of code to be executed if some condition is true,otherwise we tend to ignore the indented code.
- Along with conditional statements we tend to use comparison operators.

Python	Meaning
<	Less than
<=	Less than or Equal to
==	Equal to
>=	Greater than or Equal to
>	Greater than
!=	Not equal

- **Indentation:**
- In python indentation is very meaningful and is used while using conditional statements, looping statements and function definition
- Indentation in python defines the scope of the statements.
- In other programming languages we generally tend to use '{ }' for defining scope but in python we use indentation.
- We tend to de-indent once the scope of the statement ends.

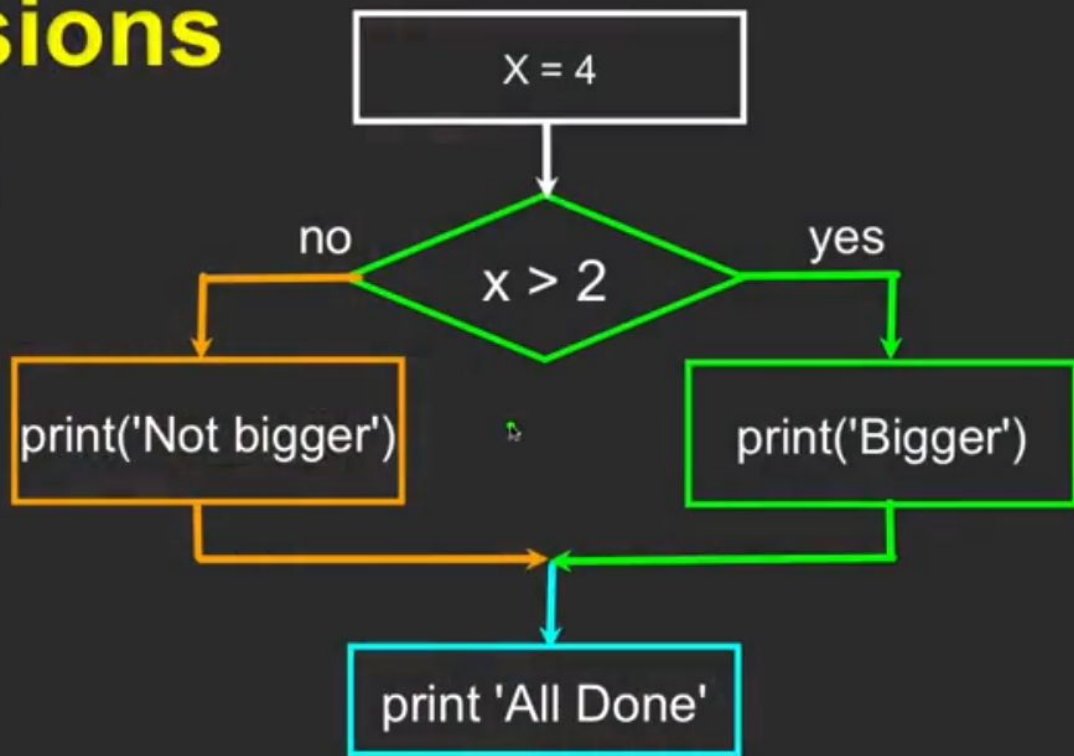


Two-way Decisions with else:

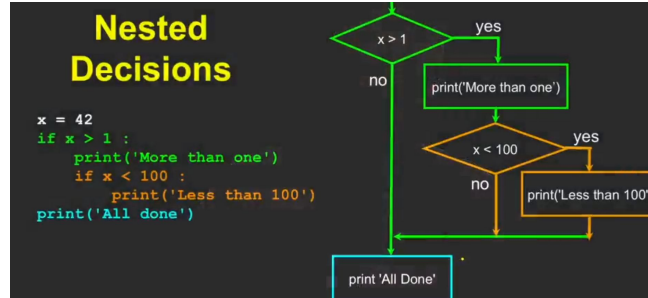
```
x = 4
```

```
if x > 2 :  
    print('Bigger')  
else :  
    print('Smaller')
```

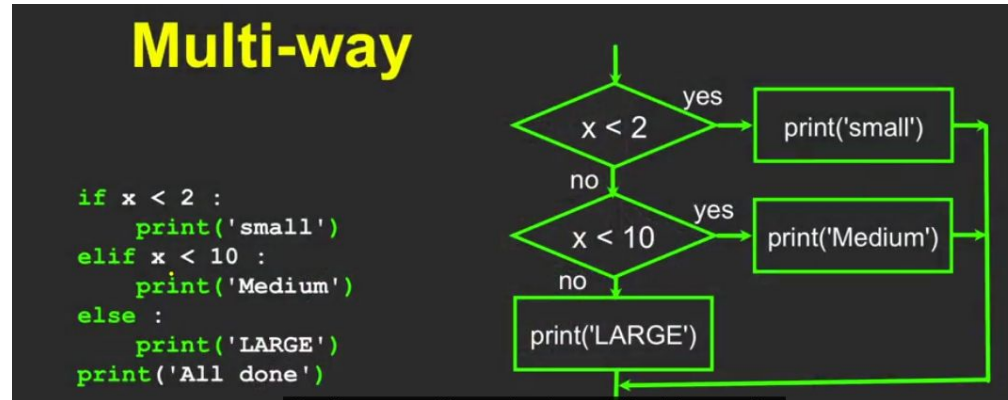
```
print 'All done'
```



- **Nested conditional statements:**
- Nested statements are the where we have condition initiated within a conditional statement.



- **Elif:**
- It is term stands for “else if” .
- It is used for multiway conditional statement.
- Elif allows us to chain multiple conditions after initial if statement.



- Try-except method:
- Try except method is an exception handling method .
- Try and except method is used for catching and eliminating traceback error.
- If an exception occurs in try method then we execute the except method but otherwise if there is no exception python tends to skip except .

```
$ cat notry.py
→ astr = 'Hello Bob'
  istr = int(astr)
  print('First', istr)
```

```
astr = 'Hello Bob'
try:
    istr = int(astr)
except:
    istr = -1

print('First', istr)
```

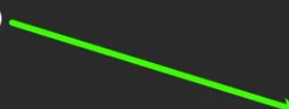
FUNCTIONS(CHAPTER-4)

- Functions are used as store and reuse .
- “Def “ keyword is used for defining a function within python script.
- Function is a named block of code ,consisting of reusable code and used for performing some specific tasks.
- Python has provided us with some built-in functions,ex-input(),max(),print(),that are used for performing specific task.
- **Building a function:**
- Def keyword is used for building a local function.
- Def keyword just remembers the code and store it.
- For executing or running the code stored as function we will have to invoke the code by invoking it.

```
x = 5
print('Hello')

def print_lyrics():
    print("I'm a lumberjack, and I'm okay.")
    print('I sleep all night and I work all day.')

print('Yo')
print_lyrics()
x = x + 2
print(x)
```



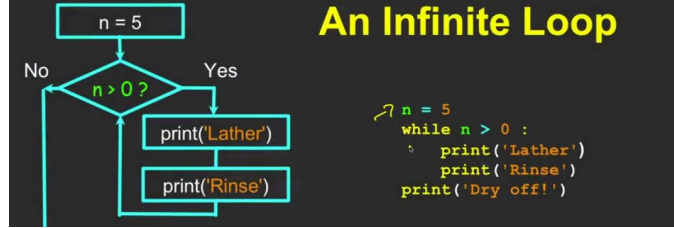
Hello
Yo
I'm a lumberjack, and I'm okay.

- Arguments:
- Arguments are the values that we pass in the function as an input when we call the function.
- We give arguments within the parenthesis.
- Parameters:
- Parameters are the variables which we use in the function definition within in the function.
- Parameters are used as an alias for whatever arguments function is going to use later when function is invoked.
- Return statements:
- We use return statements within function for two things:
- 1. It sends the residual value back to the part of the code that called the function .
- 2. It stops the function execution.

LOOPS AND ITERATION(CHAPTER-5)

- Loops are the piece of code that is repeated again and again for indefinite or definite times.
- **Indefinite loops:**
- Loops that are executed unknown number of times.
- example:While loop
- **While:**
- While is an indefinite loop that is executed until and unless the given condition is true.
- After every iteration in while loop it tends to check the condition,if the condition is :
- True: loop executed for one more time.
- False: loop execution stops.
- **Iteration variable:**
- It is an temporary variable used within the loop to hold the current item of the iterable object during execution.
- This variable tends to change it value after every iteration in order to stop the execution of the loop after some time ,if the value of the loop doesn't change then the program might end up in a infinite loop.

An Infinite Loop



- Zero trip loop:

- The loop that executes for zero times.

```
n = 0
while n > 0 :
    print('Lather')
    print('Rinse')
    print('Dry off!')
```

- Break:

- Break statement helps us to end the current loop and jumps to the statement followed by the loop immediately.
- Continue:
- Continue statement end the current loop and jumps back to the top of the loop and starts the next iteration.

```
while True:
    line = input('> ')
    if line[0] == '#' :
        continue
    if line == 'done' :
        break
    print(line)
print('Done!')
```

- **Definite loop:**
- Loops that are executed for some known number of times.
- Ex- for
- **For:**
- For is an definite loop that is executed for some known number of times.
- For loop is usually used with “in” keyword for looping through some character set or,etc.

```
for i in [5, 4, 3, 2, 1] :  
    print(i)  
print('Blastoff!')
```

- **Finding largest number:**

```
largest_so_far = -1  
print('Before', largest_so_far)  
for the_num in [9, 41, 12, 3, 74, 15] :  
    if the_num > largest_so_far :  
        largest_so_far = the_num  
    print(largest_so_far, the_num)  
  
print('After', largest_so_far)
```

\$ python largest.py

Before -1

9 9

41 41

41 12

41 3

74 74

74 15

After 74

We make a variable that contains the largest value we have seen so far. If the current number we are looking at is larger, it is the new largest value we have seen so far.

- Counting the number of elements:

```
zork = 0  
print('Before', zork)  
for thing in [9, 41, 12, 3, 74, 15] :  
    zork = zork + 1  
    print(zork, thing)  
print('After', zork)
```

```
$ python countloop.py  
Before 0  
1 9  
2 41  
3 12  
4 3  
5 74  
6 15  
After 6
```

To **count** how many times we execute a loop, we introduce a **counter variable** that starts at 0 and we add **one** to it each time through the loop.

- Summing the elements :

```
zork = 0  
print('Before', zork)  
for thing in [9, 41, 12, 3, 74, 15] :  
    zork = zork + thing  
    print(zork, thing)  
print('After', zork)
```

```
$ python countloop.py  
Before 0  
9 9  
50 41  
62 12  
65 3  
139 74  
154 15  
After 154
```

To **add up** a **value** we encounter in a loop, we introduce a **sum variable** that starts at 0 and we add the **value** to the sum each time through the loop.

- Search by using boolean variable:

```
found = False
print('Before', found)
for value in [9, 41, 12, 3, 74, 15] :
    if value == 3 :
        found = True
        print(found, value)
print('After', found)
```

```
$ python search1.py
Before False
False 9
False 41
False 12
True 3
True 74
True 15
After True
```

If we just want to search and know if a value was found, we use a variable that starts at **False** and is set to **True** as soon as we find what we are looking for.

- Finding smallest number:

```
smallest = None
print('Before')
for value in [9, 41, 12, 3, 74, 15] :
    if smallest is None :
        smallest = value
    elif value < smallest :
        smallest = value
    print(smallest, value)
print('After', smallest)
```

```
$ python smallest.py
Before
9 9
9 41
9 12
3 3
3 74
3 15
After 3
```

We still have a variable that is the **smallest** so far. The first time through the loop **smallest** is **None**, so we take the first **value** to be the **smallest**.