INRODUCTION TO

PYTHON3 & DJANGO

Martina Pivarníková

Most used programming languages of 2019

1. Java
2. C
3. Python

Python is a widely used general-purpose, high level programming language. It was initially designed by Guido van Rossum in 1991 and developed by Python Software Foundation. It was mainly developed for emphasis on code readability, and its syntax allows programmers to express concepts in fewer lines of code.

It is used for:

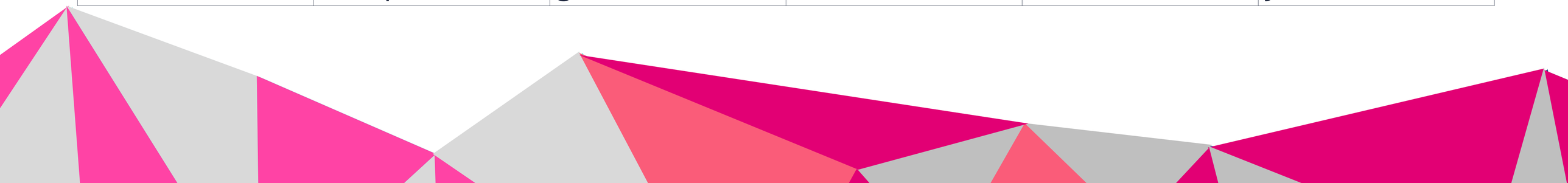
- web development (server-side),
- software development,
- mathematics,
- system scripting.

https://www.tutorialspoint.com/python3/index.htm

SYNTAX

- Extension .py
- Blocks of code – line indentation
- Single ('), double (") and triple (''' or """) quotes to denote string literals
- Hash sign (#) that is not inside a string literal begins a comment
- No semicolons at the end of line
- print(s) – output to console
- Python keywords

and	def	exec	if	not	return
assert	del	finally	import	or	try
break	elif	for	in	pass	while
class	else	from	is	print	with
continue	except	global	lambda	raise	yield



VARIABLES

- Variables - containers for storing data values.
- Unlike other programming languages, Python has **no** command for declaring a variable
- A variable is created the moment you first assign a value to it
- Python has five standard data types –
  - **Numbers**
    - int (10) , long (535633629843L), float (15.20), complex (3.14j)
  - **String**
    - str = 'Hello World!'

    - print(str) # Prints complete string
    - print(str[0]) # Prints first character of the string
    - print(str[2:5]) # Prints characters starting from 3rd to 5th
    - print(str[2:]) # Prints string starting from 3rd character
    - print(str * 2) # Prints string two times
    - print(str + "TEST“) # Prints concatenated string

VARIABLES

- **List (array)**

```
list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]
tinylist = [123, 'john']

print(list)            # Prints complete list
print(list[0])         # Prints first element of the list
print(list[1:3])       # Prints elements starting from 2nd till 3rd
print(list[2:])        # Prints elements starting from 3rd element
print(tinylist * 2)    # Prints list two times
print(list[0::2])      # Prints every 2nd element from 0
print(list + tinylist) # Prints concatenated lists
```

VARIABLES

- **Tuple**
  - The main differences between lists and tuples are:
    Lists are enclosed in brackets ([ ]) and their elements and size can be changed, while tuples are enclosed in parentheses (( )) and cannot be updated. Tuples can be thought of as read-only lists
  - `tuple = ( 'abcd', 786 , 2.23, 'john', 70.2 )`

VARIABLES

- **Dictionary**
  - They work like associative arrays and consist of key-value pairs. Dictionaries are enclosed by curly braces ({ }) and values can be assigned and accessed using square braces ([])
  - `dict = {}`
  - `dict['one'] = "This is one"`
  - `dict[2]     = "This is two"`

  - `tinydict = {'name': 'john','code':6734, 'dept': 'sales'}`

  - `print(dict['one'])` # Prints value for 'one' key
  - `print(dict[2])` # Prints value for 2 key
  - `print(tinydict)` # Prints complete dictionary
  - `print(tinydict.keys())` # Prints all the keys
  - `print(tinydict.values())` # Prints all the values

MODULES

```
pip install [package] --proxy 10.255.47.66:3128
```

```
import [package]
or
from [package] import [module]
or
import [package] as [name]
```

BASIC OPERATORS

- **Arithmetic Operators (a=10, b=20)**

Operator	Description	Example
+ Addition	Adds values on either side of the operator.	a + b = 30
- Subtraction	Subtracts right hand operand from left hand operand.	a – b = -10
* Multiplication	Multiplies values on either side of the operator	a * b = 200
/ Division	Divides left hand operand by right hand operand	b / a = 2
% Modulus	Divides left hand operand by right hand operand and returns remainder	b % a = 0
** Exponent	Performs exponential (power) calculation on operators	a**b =10 to the power 20
// Floor Division	The division of operands where the result is the quotient in which the digits after the decimal point are removed. But if one of the operands is negative, the result is floored, i.e., rounded away from zero (towards negative infinity) –	9//2 = 4 and 9.0//2.0 = 4.0, -11//3 = -4, -11.0//3 = -4.0

BASIC OPERATORS

- **Comparison Operators (a=10, b=20)**

Operator	Description	Example
==	If the values of two operands are equal, then the condition becomes true.	(a == b) is not true.
!=	If values of two operands are not equal, then condition becomes true.	(a != b) is true.
<>	If values of two operands are not equal, then condition becomes true.	(a <> b) is true. This is similar to != operator.
>	If the value of left operand is greater than the value of right operand, then condition becomes true.	(a > b) is not true.
<	If the value of left operand is less than the value of right operand, then condition becomes true.	(a < b) is true.
>=	If the value of left operand is greater than or equal to the value of right operand, then condition becomes true.	(a >= b) is not true.
<=	If the value of left operand is less than or equal to the value of right operand, then condition becomes true.	(a <= b) is true.

BASIC OPERATORS

- **Logical operators**

Operator	Description	Example
and Logical AND	If both the operands are true then condition becomes true.	(a and b) is true.
or Logical OR	If any of the two operands are non-zero then condition becomes true.	(a or b) is true.
not Logical NOT	Used to reverse the logical state of its operand.	Not(a and b) is false.

- **Membership operators**

Operator	Description	Example
in	Evaluates to true if it finds a variable in the specified sequence and false otherwise.	x in y, here in results in a 1 if x is a member of sequence y.
not in	Evaluates to true if it does not finds a variable in the specified sequence and false otherwise.	x not in y, here not in results in a 1 if x is not a member of sequence y.

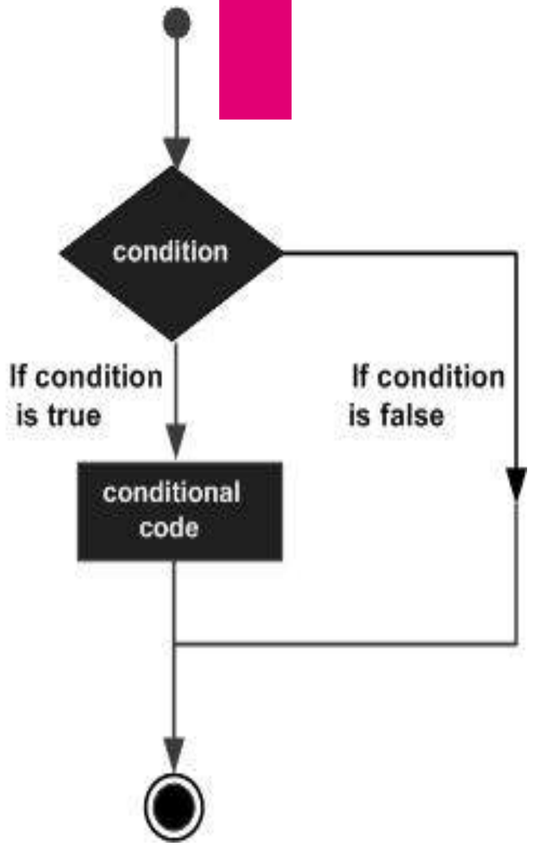
BASIC OPERATORS

- **Identity operators**

Operator	Description	Example
is	Evaluates to true if the variables on either side of the operator point to the same object and false otherwise.	x is y, here **is** results in 1 if id(x) equals id(y).
is not	Evaluates to false if the variables on either side of the operator point to the same object and true otherwise.	x is not y, here **is not** results in 1 if id(x) is not equal to id(y).

DECISION MAKING

- If statements

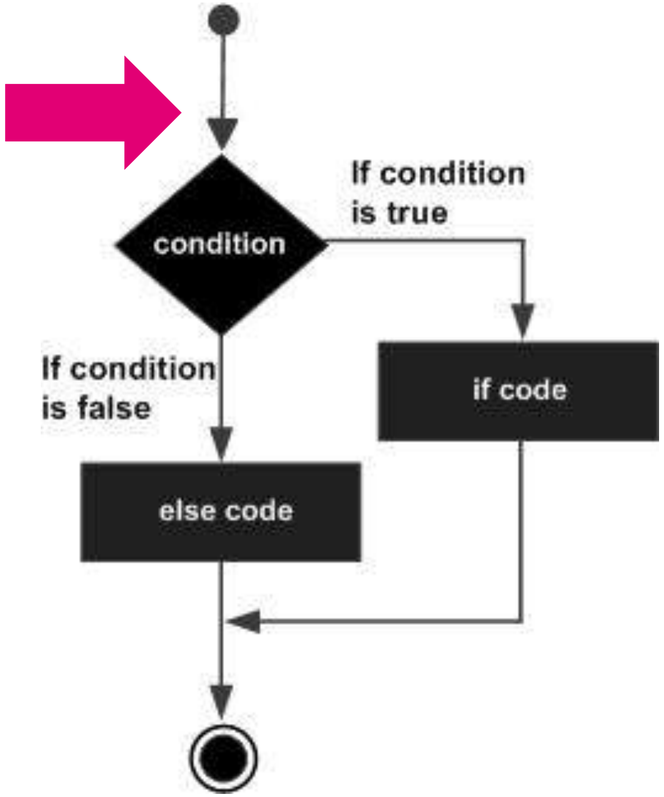


Statement & Description

if statements
An if statement consists of a boolean expression followed by one or more statements.
`if expression:`
`  statement(s)`

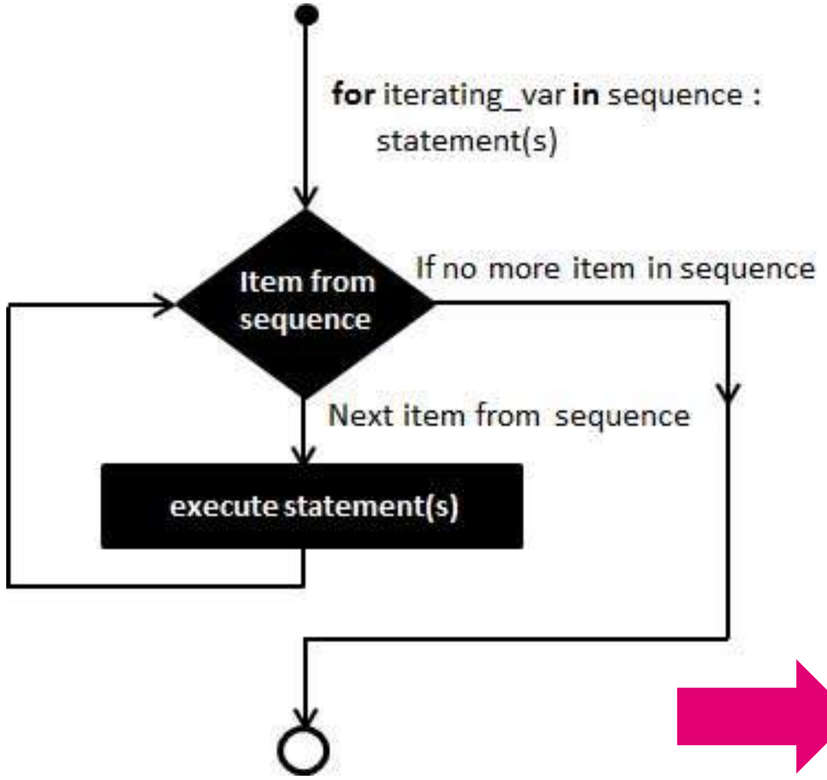
if...else statements
An if statement can be followed by an optional else statement, which executes when the boolean expression is FALSE.
`if expression:`
`  statement(s)`
`else:`
`  statement(s)`

nested if statements
You can use one if or else if statement inside another if or else if statement(s).
`if expression1:`
`  statement(s)`
`  if expression2:`
`  statement(s)`
`  elif expression3:`
`  statement(s)`
`  elif expression4:`
`  statement(s)`
`  else:`
`  statement(s)`
`else:`
`  statement(s)`



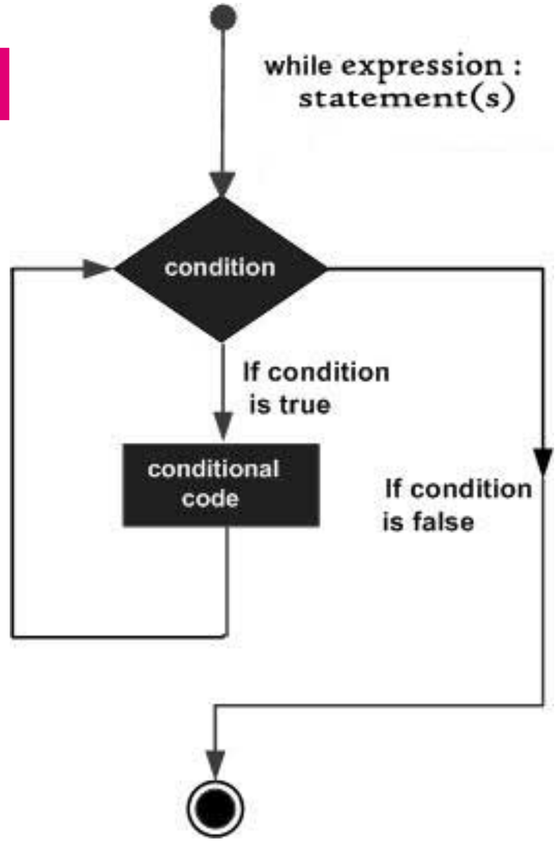
LOOPS

- Identity operators



Operator	Syntax
while	while expression: statement(s)

for	for iterating_var in sequence:
	  statements(s)
	
	for letter in 'Python': # First Example
	  print('Current Letter :', letter)
	
	fruits = ['banana', 'apple', 'mango']
	for fruit in fruits: # Second Example
	  print('Current fruit :', fruit)
	
	Iterating by Sequence index
	
	fruits = ['banana', 'apple', 'mango']
	for index in range(len(fruits)):
	  print('Current fruit :', fruits[index])



REGULAR EXPRESSIONS

A regular expression is a special sequence of characters that helps you match or find other strings or sets of strings, using a specialized syntax held in a pattern. Using module **re**.

a, X, 9, < ordinary characters just match themselves exactly.	**\S** matches any non-whitespace character.
. (a period) matches any single character except newline '\n'	**\t, \n, \r** tab, newline, return
\w matches a "word" character: a letter or digit or underbar [a-zA-Z0-9_].	**\d** decimal digit [0-9]
\W matches any non-word character.	**^** matches start of the string
\b boundary between word and non-word	**$** match the end of the string
\s matches a single whitespace character -- space, newline, return, tab	**** inhibit the "specialness" of a character.

Example.

```
 pattern = re.compile("(201[0-9]|202[0-9]|203[0-9]|204[0-9])(1[0-2]|0[1-9])(3[0-1]|[0-2][1-9]|[1-2]0)")
if pattern.match(item):…
```

EXCEPTIONS

```
try:
   You do your operations here
   ......................
except [ExceptionI]:
   If there is ExceptionI, then execute this block.
except [ExceptionII]:
   If there is ExceptionII, then execute this block.
except:
   If there is any exception, then execute this block.
   ......................
finally:
   This part will be always executed
```

```
raise [Exception [, args [, traceback]]]
```

FUNCTIONS

- A function is a block of organized, reusable code that is used to perform a single, related action. Functions provide better modularity for your application and a high degree of code reusing.

- Defining a Function
  - Function blocks begin with the keyword def followed by
  - the function name and parentheses (( )).
  - Any input parameters or arguments should be placed within these parentheses. You can also define parameters inside these parentheses.
  - The code block within every function starts with a colon (:) and is indented.
  - The statement return [expression] exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as return None.

```
def functionname( parameters ): #default parameters – e.g (name,age=35)
        "function_docstring“
        function_suite
        return [expression]

_____________________________
[result =] functionname(param)
```

GLOBAL VS LOCAL VARIABLES

- Variables that are defined inside a function body have a local scope, and those defined outside have a global scope.

```
total = 0   # This is global variable.
# Function definition is here
def sum( arg1, arg2 ):
    # Add both the parameters and return them."
    total = arg1 + arg2; # Here total is local variable.
    print ("Inside the function local total : ", total)
    return total

# Now you can call sum function
sum( 10, 20 )
print ("Outside the function global total : ", total )
```

Result:

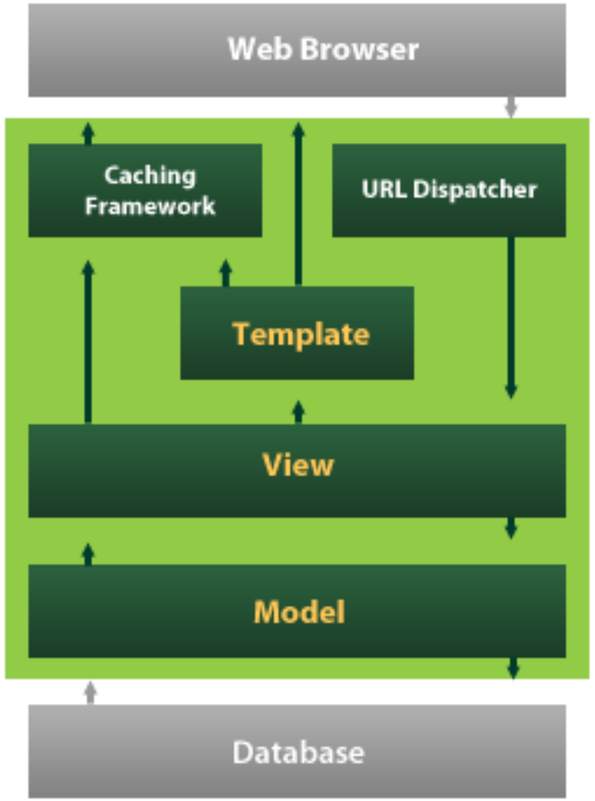
```
Inside the function local total :  30
Outside the function global total :  0
```

DJANGO

- open source web application framework
- Architecture Model-view-controller
- Originally: to manage several news-oriented sites
- In June 2005, it was released publicly under the open source BSD license

- Main advantage: easy creation of complex, database-driven web applications
- "Do Not Repeat Yourself” - DRY
- The core of the framework contains an object-relational mapper, which is a mediator between the data model (defined by the Python class) and the relational database
- built-in template system

django



1. The URL dispatcher (urls.py) maps the requested URL to a view function and calls it. If caching is enabled, the view function can check to see if a cached version of the page exists and bypass all further steps, returning the cached version, instead. Note that this page-level caching is only one available caching option in Django. You can cache more granularly, as well.

2. The view function (usually in views.py) performs the requested action, which typically involves reading or writing to the database. It may include other tasks, as well.

3. The model (usually in models.py) defines the data in Python and interacts with it. Although typically contained in a relational database (MySQL, PostgreSQL, SQLite, etc.), other data storage mechanisims are possible as well (XML, text files, LDAP, etc.).

4. After performing any requested tasks, the view returns an HTTP response object (usually after passing the data through a template) to the web browser. Optionally, the view can save a version of the HTTP response object in the caching system for a specified length of time.

5. Templates typically return HTML pages. The Django template language offers HTML authors a simple-to-learn syntax while providing all the power needed for presentation logic.

Thanks!

Any questions?