Luigi python pdf

A complete overview of Python features:I our previous tutorial we discussed the types and uses of control statements in detail. In this guide, we'll discuss Python features along with simple examples. Don't fail to read through our complete range of Python Tutorials in this series. Watch video tutorials function arguments in Python: Video #1Functions, calling a Function & Return Statement in Python: Video #2FunctionsA feature is a block of code used to perform certain specific actions. A feature provides higher modularity and code reusability. Features help break a large code into smaller modules. Syntax:def function_name(parameters): #Block of code or clausesDefine a FunctionFunction block should always start with the keyword 'def,' followed by function names and parentheses. We can pass any number of parameters or arguments inside the parentheses. The block of a code for each function should start with a colon (:)An optional 'return' statement to return a value from the function. Example: def my_function(): print(Hello Python) Simply defining a function is useless if you don't call it. Calling a FunctionOnce the structure of a function is closed, you can execute it by calling the function by using the function name. For example, def my_function(): print(Hello Python) my_function() Output:Hello PythonCalling a Function using ParametersWe can define any number of parameters while while defining a defining a function. Syntax:def my_function(parameters): #Block of code or statementsExample: def my_function(fname): print(Current language is: , fname) my_function(Python) my_function(Java) Output:Python Current language is: Python Current language is: JavaReturn StatementA return clause is used to return a value from the function. Example: def additions(a, b): sum = a+b return sum print(Sum is: , addition(2, 3)) Output:Sum is: 5Output:Function ArgumentsIn python, we can call a function using 4 types of arguments:Required argumentKeyworddefault argumentVariable length argument #1) Required arguments:Required arguments are the arguments passed to a function in a sequential order, the number of arguments defined in a function should match with the function definition. Example: def addition(a, b): sum = a+b print(The sum of two numbers is:, sum) addition(5, 6) Output:The sum of two numbers is: 11Output:#2) Keyworded Arguments:When we use keyword arguments in a function call, the caller identifies the arguments with the argument name. Example: def language(lname): print(Current language is:, lname) language(lname = Python) Output:Current language is: PythonOutput:#3) Default arguments:When a function is called without any arguments, that default argument is used. Example: def country(cName = India): print(Current country is:, cName) country(New York) country(London) country() Output:Current country is: New York Current country is: London Current country is: India It's time again: new version of Python is imminent. Toward. in beta (3.9.0b3), we will soon see the full release of Python 3.9. Some of the newest features are incredibly exciting, and it will be great to see them used after release. We will cover the following: Dictionary Union Operators Type Hinting Two new string methods New Python parser Let's take a first look at these new features and how to use them. One of my favorite new features with an elegant syntax. If we have two dictionaries a and b that we need to merge, we can now use union operators. We have the merge operator: And the update operator |=, which updates the original dictionary: If our dictionaries share a common key, the key value pair in the other dictionary will be used: Dictionary Update with Iterables Another cool behavior of the operator |= is the ability to update the dictionary with new key value pairs using an itecable item – as a list or generator : If we try the same thing with the default tray operator | we will get a TypeError because it will only allow unions between dict types. Type tip Python is dynamically written, which means we don't have to enter data types in our code. This is okay, but sometimes it can be confusing, and suddenly Python's flexibility becomes more of a nuisance than anything else. Since 3.5, we were able to specify types, but it was quite cumbersome. This update has really changed that, let's use an example: No type suggests (left) v type suggests with 3.9 (right) In our add_int function, we clearly want to add the same number to itself (for some mysterious undefined reasons). But our editor doesn't know it, and it's perfectly okay to add two strings along with the help of + — so no warning is given. What we can now do is set the expected input type as int. Using this, our editor picks up on the problem immediately. We can get pretty specific about the types included as well, for example: Type suggests can be used everywhere — and thanks to the new syntax, it now looks much cleaner: We enter sum_dict arguments as a dict and the returned value as an int. During the test definition, we also determine its type. Not as glamorous as the other new features, but it's still worth a mention as it's particularly useful. Two new string methods for removing prefixes and suffixes have been added: New parser This is more of an out of visual impairment change but has the potential to be one of the most significant changes for the future development of Python. Python currently uses a predominantly LL(1) based grammar, which in turn can be parsed by an LL(1) interpreter – who interprets code from top to bottom, left to right, with a lookahead of just one token. Now, I have almost no idea how this works — but I can give you some of the current issues in Python because of the use of this method: Python contains non-LL(1) grammar; because of this, some parts of the grammar uses workarounds, creating unnecessary complexity. LL(1) creates limitations in the Python syntax (without possible solutions). This issue highlights the fact that the code simply cannot be executed using the current parser (raising a SyntaxError): LL(1) breaks with left-recursion in parses. Meaning special recursive syntax can cause an infinite loop in the parse tree. Guido van Rossum, the creator of Python, explains this here. All these factors (and many more that I simply can't understand) have a big impact on Python; they limit the development of language. The new parser, based on PEG, will allow python developers to be much more flexible – something we will start to notice from Python 3.10 onwards. That's all we can look forward to with the upcoming Python 3.9. If you really can't wait, the latest beta version – 3.9.0b3 – is here. This article originally appeared on Towards Data Science by James Briggs, an AI consultant based in London. He is fascinated by the phenomenal advances made in tech eco-systems daily and loves writing about AI, Python, and programming in general. Follow him on Twitter. Read next: Satoshi Nakaboto: Grayscale exceeds $5B in Bitcoin under management TechPython (programming language)Codelanguage Python 3.9 expected to be released on Monday, October 5, 2020. Before releasing the official version, the developers had planned to release six alpha, five beta preview, and two release candidates. At the time of writing, the first candidate was recently released on August 11. Now we are eagerly awaiting the second release candidate, which will probably be available from 14 September. So, you might be wondering what's new in Python 3.9. Right? There are some significant changes that will dictate how python programs work. Most importantly, in this latest version, you will get a new parser that is based on interpretation expression grammar (PEG). Similarly, it merges | and update |= Union Operators is added to dict . So, let's take a closer look at all the upcoming features and improvements to Python 3.9. New parser based on PEG Unlike the older LL(1) parser, the updated version has some key differences that make it more flexible and future-proof. Basically in LL(1), python developers had used several hacks to avoid its limitations. In turn, it affects the flexibility to add new language features. The big difference between PEG and a contextless-grammar based parsers (e.g. LL(1)) is that in the PEG the selection operator is ordered. Let's assume we write this rule: A | B | C. Now, in the case of LL(1) parser, it will generate designs to conclude which from A, B, or C needs to be expanded. On the other hand, PEG will try to check whether the first option (e.g. A) succeeds or not. It will continue to the next option only when A does not succeed. In simple words, PEG will check the options in the order in which they are written. Support for IANA time zone In real applications, users typically require only three types of UTC The local time zone for System IANA Time Zones Now, if you are already familiar with previous versions of Python then you may know that Python 3.2 introduced a class datetime.timezone. Basically, its main objective was to provide support to UTC. In true sense, the local time zone is still not available. However, in version 3.0 of Python, developers changed the semantics of naïve time zones to support local time operations. In Python 3.9, they will add support for the IANA time zone database. For the most part, this database is also called tz or the Olson database. So, don't get confused with these terms. All IANA time zone functionality is packed inside the zoneinfo module. This database is very popular and widely distributed in Unix-like operating systems. However, remember that Windows uses a completely different method of managing time zones. Added union operators In earlier versions of Python, merging or updating two dicts has not been very effective. That is why developers are now before Union Operators who | for merging and |= for updating the poets. For example, earlier when we use d1.update(d2) then it also changes d1. So, to fix it, we need to implement a small hack something like e = d1.copy(); e.update(d2). Here we create a new temporary variable to keep the value. However, this solution is not very effective. That is the main reason for adding the new union operators. Introducing delete prefix() and removesuffix() Have you ever felt the need for certain features that can easily remove prefixes or suffixes from a given string? Now, you can say that there are already some features like str.lstrip([chars]) and str.rstrip([chars]) that can do this. But this is where the confusion begins. Actually, these functions work with a set of characters instead of a substring. So, there is definitely a need for some separate functions that can remove the substring from the beginning or end of the string. Another reason to provide built-in support for removeprefix() and removesuffix() is that software developers usually write this functionality on their own to increase their productivity. But in most cases, they make mistakes while handling empty strings. So, a built-in solution can be very helpful for real apps. Type suggests generics in default collections Do you ever notice the duplicate collection hierarchy in the write module? For example, you can either use typing. List or built-in list. So, in Python 3.9, the core development team has decided to add support for generic syntax to the write module. The syntax can now be used in all standard collections contained in this module. The bigger plus of this feature is now users can easily comment on their code. It also helps instructors learn Python in a better way. Added graphlib module In graphs, a topological order plays an important role in identifying the flow of jobs. Which means it follows a linear order to tell task that will run before the second. Other. the graphlib module enables us to perform a topological sort or order on a graph. It is mostly used with hashable nodes. Modules reinforced in Python 3.9 In my opinion, the major effort took place while improving the existing modules. You can evaluate this with the fact that a massive list of 35 modules is updated to optimize the Python programming language. Some of the most significant changes happened inside GC, http, imaplib, ipaddress, math, os, pydoc, random, signal, socket, time, and sewn modules. Inprecated Python features Around 16 features are anesthetized in Python version 3.9. You can get detailed information from the official Python 3.9 announcement. Here I will try to give you a brief overview of the most important things that are clouded. If you have ever worked with random module then you probably know that it can accept any hashable type as a seed value. This can have unintended consequences because there is no guarantee whether the hash is deterministic or not. That's why the developers decided to just accept No, int, float, str, byte and byte array as the seed value. Also, from now on, you must set the state argument to open a GzipFile file for writing. A total of 21 features that were pre-decorated in previous versions of Python have now been completely dropped from the language. You can take a look at the complete list on python's website. This article originally appeared on the Live Code Stream by Juan Cruz Martinez (Twitter: @bajcmartinez), founder and publisher of Live Code Stream, entrepreneur, developer, author, speaker and doer of things. Live Code Stream is also available as a free weekly newsletter. Sign up for updates on everything related to programming, AI and computer science in general. Read Next: This Company Claims It Can Use Nuclear Waste to Make Batteries That Last 28,000 Years Without Charging TechdevelopersPython (Programming Language)Computer Science