Особенности использования и ограничения Python

Общие требования для работы узлов Python в Loginom:

- В системе должен быть установлен Python одной разрядности с приложением/сервером Loginom.
- В Loginom может потребоваться предварительная настройка в Параметрах компонента: Python в Администрировании.
- Минимальная поддерживаемая версия Python — 3.5.
- Максимальная протестированная версия Python — 3.12.

Для настольных редакций выполнение узла *Python* по умолчанию разрешено, для серверных — запрещено. При необходимости эти настройки можно изменить в «Параметрах компонента: Python».

Loginom поддерживает два режима работы Python:

- внутри процесса Loginom (только на Windows);
- в отдельном процессе (на Windows и Linux).

Работа узлов Python внутри процесса Loginom

Особенности и ограничения:

- Python из дистрибутива Anaconda не поддерживается.
- Не поддерживаются модули пакета multiprocessing.
- Создание GUI-интерфейса из кода Python не является целевым назначением компонента *Python* и в некоторых случаях может приводить к неверной работе приложения. Например:
  - для работы `matplotlib` с бэкэндом Qt и в целом с PyQt требуется наличие файла qt.conf с путями к библиотекам/плагинам Qt. При невозможности их найти, приложение Loginom будет принудительно завершено с сообщением об ошибке.
- Оставленные незавершенными фоновые потоки могут приводить к аварийному завершению приложения.
- В случае критических ошибок *python3x.dll* может аварийно завершить приложение. Рекомендуется использовать предварительно отлаженный код Python.
- Одновременно может выполняться только один узел *Python*.
- Остановить выполнение узла *Python* можно только между инструкциями интерпретатора.
- Виртуальное окружение можно использовать при задании в Администрировании параметра компонента Python *Путь интерпретатора*. Для этого путь интерпретатора должен указывать на python.exe внутри виртуального окружения (подробнее о поиске модулей Python внутри процесса Loginom). Для корректной работы версия python.exe в виртуальном окружении и версия *python3x.dll*, через которую выполняется код узла, должны совпадать.

> **Примечание:** под Linux не поддерживается режим выполнения Python внутри процесса Loginom. Узлы *Python*, настроенные на выполнение внутри процесса, на Linux будут выполняться в отдельном процессе.

Работа узла Python в отдельном процессе

Особенности и ограничения:

- При запуске исполняемого кода на Python в отдельном процессе может выполняться сразу несколько узлов *Python* параллельно.
- Тайм-аут ожидания запуска (мс) не поддерживается.

Работа узлов Python в изолированном окружении

На Linux возможна работа узлов Python в изолированном окружении пользователя. Для этого необходимо:

- В Администрировании, в «Параметрах компонента: Python» включить опцию *Передавать переменные окружения узла*.
- В параметре *Путь интерпретатора* указать полный путь к bash-скрипту — *python_run.sh* (или аналогичному).
- Установить *Docker* или *Podman*. В *python_run.sh* имеется параметр `DOCKER_OR_PODMAN`, отвечающий за способ контейнеризации, значение по умолчанию — `podman`.
- Подготовить базовый docker образ с Python и нужными Python пакетами внутри. В *python_run.sh* имя образа прописывается в параметре `IMAGE_NAME`, значение по умолчанию — `python`.

В *python_run.sh* имеется параметр `CONTAINER_OPTIONS`, в который можно передавать настройки контейнера, например, значение переменной `oom-score-adj`. По умолчанию устанавливается значение `oom-score-adj=1000`. То есть если OOM Killer будет необходимо завершить какой-либо процесс, то контейнер будет первым кандидатом для этого.

Поиск модулей Python внутри процесса Loginom

Код узла выполняется в модуле `__main__`. Таким образом выполняется условие `__name__ == '__main__'`, которое обычно используется для запуска скриптов.

Поддерживается импорт модулей из файлового хранилища. Путь к папке, в которой расположен интерпретатор, добавляется как ещё один путь в `sys.path` — список путей, по которым ищутся модули.

- Если пакет сохранен, то относительный путь к импортируемому модулю указывается от расположения пакета. sys.path (`sys.path[0]`) содержит путь к директории пакета, в котором находится узел *Python*.

- Если пакет не сохранен, то относительный путь к импортируемому модулю указывается от каталога пользователя. sys.path (`sys.path[0]`) содержит путь к каталогу пользователя.

Если на один каталог выше каталога *Путь интерпретатора* существует файл с именем `pyvenv.cfg` , то `sys.prefix` и `sys.exec_prefix` устанавливаются равными этому каталогу, и проверяется наличие `site-packages` . Найденный `site-packages` добавляется в `sys.path` . Если `pyvenv.cfg` содержит ключ `include-system-site-packages` и его значение не равно true, то базовый путь `site-packages` (относительно `sys.base_prefix`) не включается в `sys.path` .

В модулях относительный путь указывается от расположения модуля, осуществляющего импорт.

▼ Пример:

Относительно сохраненного пакета Loginom расположены следующие модули:

- модуль `./foo/__init__.py` :

```
__all__ = ["module"]
from .foo import cube
```

- модуль `./foo/foo.py` :

```
def cube(x):
    return x * x * x
```

- модуль `./foo/module/__init__.py` :

```
from .module import say_hello
```

- модуль `./foo/module/module.py` :

```
def say_hello():
    return "Hello"
```

При таком расположении модулей импорт и вызов функций этих модулей может осуществляться в узле *Python* следующим образом:

```
from foo import cube
from foo.module import say_hello

print(say_hello())
print("3 ** 3 =", cube(3))
```

В режиме Предпросмотра результатов значение атрибута `__file__` модуля `__main__` равно `<preview>`, при выполнении узла — `<main>`.

▼ **Пример:**

```
from builtin_data import OutputTable

if __file__ == "<preview>":
    print(__file__)
else:
    OutputTable.Append()
    OutputTable.Set(0, __file__) #__file__ == "<main>"
```