TURING

Python
Question 6 of 30

You are a developer writing a Python application to synchronize a set of drones to perform a procedure in real time. This needs synchronous I/O multiplexing, implemented by performing non-blocking I/O using threads, each of which calls the select() system call on a set of read/write I/O ports. Which of the following is/are true for this application? Select all that apply.
(Select all that apply.)

Quick Note
If the question is same as the last one or showing blank content, please click "Reload Content"

Quick Note
If the question is same as the last one or showing blank content, please click "Reload Content"

Reload content | Report a problem

Select all that apply

00:38:10

Next questio

A. The model discussed above is one of synchronous I/O multiplexing using select(), which is designed to work with blocking I/O. Thus, non-blocking I/O using several threads is not possible.

B. Non-blocking I/O using multiple threads is possible since each of those threads can poll a single read/write port, and perform reads/writes.

C. In general, non-blocking I/O using multiple threads results in better performance than trying to poll the ports serially, since the select() system calls can be executed in parallel, even though the threads are executed one at a time.

D. The only way to achieve synchronous I/O multiplexing using select() as above, is to use the Python multiprocessing feature provided by concurrency.futures.

concurrency.futures.

E. We cannot achieve better performance using non-blocking I/O with multiple threads, as compared to serial execution since the underlying CPython GIL prevents the threads from achieving true parallelism.

Python
Question 7 of 30

You are a developer, working on a Python 3 program. You are asked to do continuous integration using sockets, for sending data between applications. Consider the following code snippet from your own application:

```
# .. some code here
try:
   transport._extra['sockname'] = sock.getsockname()
except socket.error
   if transport.loop.get_debug():
     logger.warning("getsockname() failed on %r", sock,exc_info =
True)

if "peername" not in transport._extra:
   try:
     transport._extra['peername'] = sock.getpeername()
   except socket.error:
     transport._extra['peername'] = None
# .. some code here
```

Which of the following statements is true for the above code snippet?

Select one answer

00:36:50

Next quest

A. We can directly use sock.getpeername() instead of try except block, since getpeername returns a tuple and handles null if there is none.

B. Some of the UDP sockets may or may not have a peer name, and hence we are catching the exception using socket.error

C. Some of the TCP sockets may not have a peer name, and hence we are catching the exception using socket.error

D. All of the UDP sockets may not have peer names, and hence we are catching the exception for all UDP sockets.

TURING

Python
Question 8 of 30

You are a developer tasked with writing a Python application where 99% of the public API end-points that you expose works fine, but the one new endpoint you just added has a bug. You do not want that one new endpoint to crash-loop your deployed service. Which of the following non-blocking calls would you use?

Quick Note
If the question is same as the last one or showing blank content, please click "Reload Content"

Reload content | Report a problem

Select one answer 00:36:19 Next question

A asyncio.get_event_loop().run_forever()

B aiorun.run(main())

C asyncio.run(main())

D asyncio.run(broker.serve_forever())

E aiorun.run(main(), use_uvloop=True)

TURING

Python
Question 9 of 30

What is the difference between the implementations of the methods __repr__() and __str__(), within a user-defined class in Python? Select all that apply.
(Select all that apply.)

Quick Note
If the question is same as the last one or showing blank content, please click "Reload Content"

Reload content | Report a problem

Select all that apply 00:35:46 Next question

A A call to repr() invokes both __repr__() and __str__(), whereas a call to str() invokes just __str__()

B If we do not implement the __str__() function, then a call to str() on an object invokes __repr__()

C An invocation of __str__() returns a user-friendly printable string, and that can also be used by a debugger to reconstruct a representation of the original object

D If we do not implement __repr__(), then a call to repr() on an object invokes __str__()

Quick Note
If the question is same as the last one or showing blank content, please click "Reload Content"

Reload content | Report a problem

E An invocation of __repr__() returns a developer-friendly printable string, and that can also be used by a debugger to reconstruct a representation of the original object

TURING

Python
Question 10 of 30

What does the below code snippet do?

```
def some_func(array):
  array = iter(array)
  try:
   first = next(array)
  except StopIteration:
   return True

  return all(first == x for x in array)
```

Reload content | Report a problem

Select one answer 00:35:00 Next ques

A. Converts the input to a set and checking that it only has one or zero and returns true

B. Converts all the input to list and returns true

C. Returns true if all elements except the first element are evaluated as True

D. Converts the input to map without first item and returns true

E. Returns true if all elements are evaluated as True

Reload content | Report a problem

You are working on a Python 3 program. Consider the following code snippet from a distributed application.

```
# .. some code here
def run_threaded(job_func):
    job_thread = threading.Thread(target=job_func)
    job_thread.start()


schedule.every(10).seconds.do(run_threaded, job)
schedule.every(10).seconds.do(run_threaded, job)
schedule.every(10).seconds.do(run_threaded, job)
schedule.every(10).seconds.do(run_threaded, job)
schedule.every(10).seconds.do(run_threaded, job)



while 1:
    schedule.run_pending()
    time.sleep(1)
# .. some code here
```

Suppose you want a tighter control on the number of threads used, what would be your approach?

Select one answer 00:33:51 Next ques

A. Use only jobQueue.queue that will queue relevant threads as and then it is added.

B. Use multiple schedulers and make them run serially.

C. Use a thread pool

D. None is correct.

TURING

Python
Question 12 of 30

You are working on a Python 3 program that is doing a sequential for loop through a list of input strings on some operation f, where f is CPU-bound. f is embarrassingly parallel. As follows:

```
# some code
for line in lines:
 f(lines)
# some code
```

Given f is taking a long time, you would like to take advantage of your multicore CPU to parallelize the operation. In this case, which of the following approaches would work?
(Select all that apply.)

to parallelize the operation. In this case, which of the following approaches would work?
(Select all that apply.)

Select all that apply

00:33:15

Next que

A. We can start a new separate process for each f while iterating the for loop.

B. We can write code at Cython level and explicitly release the GIL (Global Interpreter Lock) to run f concurrently with multiple threads.

C. We can use the asyncio library to run f in parallel.

D. We can parallelize f on CPU level by implementing and manipulating the low level threading interface directly, working around the GIL (Global Interpreter Lock)

E. We can use a thread pool to spawn the number of threads equal to the number of CPU cores to run f.

TURING

Python
Question 13 of 30

Given a Python 3 string of type Unicode ZÃ¼rcher, we would like to store it as a utf-8 string on persistent storage. When we try to decode the string as UTF-8, it fails to be converted and raises an error. What could have been the reason?

Quick Note
If the question is same as the last one or showing blank content, please click "Reload Content"

Select one answer

00:32:32

Next ques

A. Python 3 does not support unicode or utf-8 conversion.

B. The string ZÃ¼rcher is correctly formatted, but we should not be calling decode but should be calling encode.

C. Python 3 does not have the concept of unicode/utf-8 strings, all is just "strings".

D. The string ZÃ¼rcher is incorrectly formatted, so it is not a valid unicode string or utf-8 string.

Question 14 of 30

Select one answer

00:31:29

Next que

Consider the following Python 3 code:

```
class Square:
  def draw(self):
    print(f'Inside Square::draw()')

  def resize(self):
    print(f'Inside Square::resize()')

class Circle:
  def draw(self):
    print(f'Inside Circle::draw()')

  def resize(self):
    print(f'Inside Circle::resize()')

class ShapeManager:
  def __init__(self, shapes):
    self._shapes = shapes

  def manage(self):
    for shape in self._shapes:
      shape.draw()
      shape.resize()
```

```
if __name__ == '__main__':
  shapes = [Square(), Square(), Circle(), Square(), Circle(),
Circle(), Square(), Circle()]
  shape_manager = ShapeManager(shapes)
  shape_manager.manage()
```

Which of the following statements is true?

A. The above code works, because Duck Typing is supported in Python, and the Python runtime only looks for available definitions of draw() and resize() within the Square and Circle objects

B. The above code works, because both Square and Circle classes inherit from the built-in Python class object by default, and the above definitions add the abstract methods draw() and resize() to the definition of the object

C. The above code does not work, because both Square and Circle classes do not inherit from any common base class or abstract class, and polymorphism cannot be implemented

D. The above code works, because both Shape and Circle classes inherit from the built-in Python abstract class ABC by default, and the decorator @abstractmethod is implicitly made available in the Square and Circle classes.

E. The above code works, because Duck Typing is supported in Python, and the Python runtime automatically creates an implicit inheritance hierarchy from the built-in abstract base class ABC, and the decorator @abstractmethod is made available in the Square and Circle classes.

Python

Question 15 of 30

Consider the following snippet of Python code:

```
#show list of squares of numbers in list
L=[1,2,3,4,5,6,7,8,9,10]
```

Which of the following expressions in Python code will deliver the desired result - to obtain the list of squares of numbers in L?

Quick Note

If the question is same as the last one or showing blank content, please click "Reload Content"

A. map(L, lambda x: x**2)

B. list(map(lambda x: x**2, L))

C. L**2

D. map(x: x**2, L)

TURING

Python
Question 16 of 30

You are a Python developer tasked with writing an application to control a swarm of 10,000 drones, to set up a dazzling laser show at the FIFA Football World Cup. This needs large-scale synchronous I/O multiplexing using select() system calls. You have two approaches: use multithreading or use coroutines. Which approach is preferable, and why?
(Select all that apply.)

Quick Note
If the question is same as the last one or showing blank content, please click "Reload Content"

Quick Note
If the question is same as the last one or showing blank content, please click "Reload Content"

Select all that apply

00:28:13

Next que

A
Multithreading is preferred since it can achieve synchronous I/O multiplexing with non-blocking I/O. Coroutines cannot be used to implement non-blocking I/O.

B
Coroutines are preferred since this mechanism uses the event loop that is free from the overhead of stack allocation and context switching that we associate with multithreading.

C
Coroutines are preferred since their subsequent invocations are managed by the event loop through the fan-out and fan-in mechanisms. This incurs much less synchronization overhead as compared to multithreading

D
Coroutines are preferred, since they are not limited by the underlying CPython GIL, and can achieve true application-level parallelism.

E
Multithreading is preferred since it can achieve synchronous I/O multiplexing with non-blocking I/O. Coroutines cannot be used to implement synchronous multiplexing involving system calls.

TURING

Python
Question 17 of 30

You have a Python 3 program P that is being tested by its corresponding unit test file TP. You introduced 5 test cases (ta, tb, tc, td, te). Each of these tests pass individually when run alone. However, when you run all the tests using default parameters, some of the tests fail to pass, and every time the failing test cases are different. What do you think is the most likely reason?

Quick Note
If the question is same as the last one or showing blank content, please click "Reload Content"

Select one answer

00:27:18

Next que

A
The Python unittest standard library was running tests in parallel by default, which breaks the assumptions in P thus causing TP to return failure randomly.

B
The design and implementation of P or TP have shared states across tests, either in P or TP.

C
The tests failures were expected to fail, because the developers did not run the tests enough number of times to counter the randomness. e.g. running the test 50 times instead of 5 times.

D
The tests in TP were run in parallel without synchronization primitives, thus causing race conditions, leading to tests to fail randomly.

monitors a library of books. If any overdue books are found, they need to be displayed at the top of the dashboard, with an alert. The underlying code looks similar to this:

```
book_ids = [1001, 1002, 1003, 1004, 1005, 1006, 1007, 1008, 1009]

overdue_ids = {1002, 1006}

def display_sorted_books(book_ids, overdue_ids):

  found = False

  def helper(id):

    if id in overdue_ids:

      found = True

      return (0, id)

    return (1, id)

  book_ids.sort(key=helper)

  return found
```

We see that even when elements from the set overdue_ids are found in the list book_ids, the function display_sorted_books() always returns False. What can we do to fix this error. and why?

Reload content | Report a problem

Select one answer 00:26:11 Next quest

A. Declare found as nonlocal within the scope of helper(), to enable scope traversal out of a nested function.

B. Define found within the scope of helper(), instead of in display_sorted_books(), since closures cannot work with variable assignment within a nested function.

C. Declare found as nonlocal at the place of its definition within the scope of display_sorted_books(), to enable scope traversal out of a nested function.

D. Declare found as nonlocal within the scope of helper(), since the closure of a nested function does not include its enclosing scopes.

TURING

Python
Question 19 of 30

You have a Python 3 program that exposes an API function that has a type specified for the input parameters and returns values, as follows:

```
def user_facing_api(text: str, amount: float) -> bool:
  # uses text
  # uses amount
  # returns boolean results
```

During runtime, you noticed that the callers were actually passing in string type for both input parameters, despite expecting a string and float parameter. However, the program did not raise any errors or warnings. What happened?

Select one answer 00:25:18 Next que

A. The Python runtime type check caught the amount having a type string and not float, but the TypeError exception was suppressed.

B. There is no runtime type check. The user code inside the function user_facing_api happens to just work.

C. The code was actually in an undefined state, so it may raise an error the next time we run it.

D. The Python runtime type check was disabled by default, so the type violation was not enforced.

Python
Question 20 of 30

Consider a Python 3 function log_message() that implements logging functionality (internal to the application), that logs information into files in a filesystem. In addition, the code for log_message() is organized as follows (to be treated as pseudocode, as parts of the code are incomplete):

```
def log_message(filename, message):
   try:
      handle = open(filename, 'a')
      write(handle, message)
   except Exception as e:
       # Some code
   else:
       # Some code
   finally:
       # Some code
```

Now, where is it recommended we place the code to close the file handle that was opened in the try clause?

Select one answer 00:24:39 Next que

A In the finally clause, because the code in this clause is always executed, whether an error scenario occurs or not

B In the try clause itself, because each call to open() must be matched by a symmetric call to close()

C In the except clause, because the opened file remains open only when an error scenario occurs

D In the else clause, because the code in this clause is always executed, whether an error scenario occurs or not

E In the else clause, because the opened file remains open only when a regular, non-error scenario occurs

Now, where is it recommended we place the code to close the file handle that was opened in the try clause?

TURING

Python
Question 21 of 30

Consider the following snippet of Python code:

```
for sentence in paragraph:
  for word in sentence.split():
    single_word_list.append(word)
```

What will the equivalent of the above code be?

Select one answer 00:23:41 Next qu

A
```
single_word_list = [word for sentence in paragraph for word in sentence.split()]
```

B
```
single_word_list = [word for sentence in paragraph for word in sentence]
```

C
```
single_word_list = [word for word in sentence for sentence in paragraph]
```

D
```
single_word_list = [word for word in sentence.split() for sentence in paragraph]
```

TURING

Python
Question 22 of 30

Consider the following snippet of Python code:

```
def found_it(nums, looking_for):
  for i, x in enumerate(nums):
    if x == looking_for:
      return True

nums = [x for x in range(1000)]
looking_for = 500

print(found_it(nums, looking_for))
```

What is/are the best code review comment(s), for the above code?

(Select all that apply.)

Select all that apply

00:22:11

Next ques

A. The run-time performance of the above code can be improved since a better algorithm to search for the value looking_for in the found_it() function exists.

B. The function found_it() adds to the technical debt since there are unused variables in its scope.

C. We definitely need to add text comments to the definition of the function found_it(), since the code readability is very poor.

D. It is not possible to unit test the found_it() function using the PyTest unit testing module.

Python
Question 23 of 30

Consider a Python module arbit.py that imports symbols from the built-in random module in two different ways:

```
# IMPORT-1
import random
# IMPORT-2
from random import choice
```

Suppose the code in arbit.py accesses symbols from the random module as follows:

```
numbers = [num for num in range(0, 100)]
# CLIENT-CODE-1
shuffled =  shuffle(numbers)
# CLIENT-CODE-2
pick = choice(numbers)
```

Which of the above snippets of client code will work / not work, and why?

Which of the above snippets of client code will work / not work, and why?

Select one answer

00:21:43

Next ques

A. CLIENT-CODE-2 will work and CLIENT-CODE-1 will not because IMPORT-1 does not place shuffle into the symbol table of arbit.py, whereas IMPORT-2 places choice into it

B. CLIENT-CODE-1 will work and CLIENT-CODE-2 will not work because IMPORT-2 does not place choice into the symbol table of arbit.py, whereas IMPORT-2 places both shuffle and choice into it

C. Neither CLIENT-CODE-1 nor CLIENT-CODE-2 will work because both IMPORT-1 and IMPORT-2 do not place the symbols shuffle and choice from the random module, into the symbol table for arbit.py

D. Both CLIENT-CODE-1 and CLIENT-CODE-2 will work because IMPORT-2 is sufficient to place all the symbols from the random module directly into the symbol table for arbit.py

E. Both CLIENT-CODE-1 and CLIENT-CODE-2 will work because IMPORT-1 is sufficient to place all the symbols from the random module directly into the symbol table for arbit.py

TURING

Python
Question 24 of 30

You are working on a Python 3 program that is trying to compute some naive integer factorization, i.e. some compute intensive tasks. Which of the following changes will actually help improve the runtime?

Quick Note
If the question is same as the last one or showing blank content, please click "Reload Content"

Select one answer 00:20:57 Next que

A. Use of threading to run multiple OS level threads to run each task. This helps because OS level synchronization is faster.

B. Use of multiple system processes to run each task. This helps because all CPU cores are properly and effectively used.

C. Use of asyncio to run multiple application level threads to run each task. This helps because this reduces the need to do any kind of kernel context switch, and still leverages multiple CPU cores.

D. All are correct does not help.

TURING

Python
Question 25 of 30

Consider the following snippet of Python code, which is part of a larger function implementation:

```
for i in range(n//2, n):
   for j in range(1, (n//2)+1):
      for k in range(2, n, pow(2,k)):
        count = count + 1
```

The time complexity of the above code is:

Reload content Report a problem

Select one answer 00:20:25 Next questio

A. O(n^2)

B. O(n^2 log^2 n)

C. O(n^3)

D. O(n^2 log n)

E. O(n log n)

TURING

Python
Question 26 of 30

You need to define a Python class named Widget in your application with an attribute named value, a fairly common name. In which of the scenarios below, is it better to rename self.value to self.__value (with a double underscore)?

Quick Note
If the question is same as the last one or showing blank content, please click "Reload Content"

Quick Note
If the question is same as the last one or showing blank content, please click "Reload Content"

Select one answer 00:16:48 Next questi

A. The widget is internal to your application and is the base class of an inheritance hierarchy, and we do not want derived class methods to access value.

B. The widget is internal to your application, and we do not want any class methods of the Widget to access value.

C. The widget is internal to your application and is the base class of a diamond-shaped inheritance hierarchy, and we do not want derived class methods to access value.

D. Widget is defined as part of a public API, and the clients of this API are expected to subclass Widget in their own inheritance hierarchies. We do not want methods of these subclasses to access value.

E. Widget is defined as part of a public API and we do not want any class methods of Widget to access value.

TURING

Python
Question 27 of 30

When a particular numeric data field is not available in a data set for some reason, how should a Python data scientist go about deciding whether to denote that missing value as a NaN, or of type None?

Quick Note
If the question is same as the last one or showing blank content, please click "Reload Content"

Reload content Report a problem

Quick Note
If the question is same as the last one or showing blank content, please click "Reload Content"

Select one answer 00:16:08 Next que

A. It is often more efficient to store a missing numeric value as a NaN, since it maps to floating point representation in most Python numeric libraries, as opposed to None, which is represented as an object.

B. It is always preferable to denote missing numeric values as a NaN, since this representation is the most generic, and covers all cases.

C. It is always preferable to denote missing numeric values as having a None type, since this representation is the most generic, and covers all cases.

D. One can choose to represent the missing numeric value as either NaN or None since both are efficiently stored generically as an object.

E. One can choose to represent a missing numeric value as either NaN or None since both are efficiently stored as a floating point number.

TURING

Python
Question 28 of 30

Consider the following Python program that tries to manipulate a tuple:

```
# Pseudocode
document = (20005001, 'Brahma Gupta', (101, 132, 345),
['singing', 'quizzing'])
document[-1].append('poetry')
```

What would be the behavior of this program, and why?

Quick Note
If the question is same as the last one or showing blank content, please click "Reload Content"

Reload content | Report a problem

Select one answer 00:15:25 Next qu

A The program would run successfully, since when we try to update the tuple, a copy of the tuple would be generated, containing the modification. The modification does not reflect in the original tuple.

B The program would run successfully since only a reference to the list is stored, which is immutable, but the list pointed to by it can be safely modified.

C The program would return an error since a tuple cannot have an immutable data element (a list, in this example) as an item.

D The program would return an error since by definition, a tuple is immutable

E The program would run successfully. However, the tuple is inherently immutable, so would simply ignore the attempt to modify one of its elements.

TURING

Python
Question 29 of 30

What is wrong with the argument list for the Python function defined below? Select all statements that apply.

```
def fine_tune_model(hyper_param=8, iterations,
**grid_search_params, cut_off, *engg_features):

  # Function body goes here
```

(Select all that apply.)

Reload content | Report a problem

Select all that apply 00:14:18 Next quest

A The variable length argument list *engg_features needs to be listed at the second position from the front of the argument list

B The default value positional argument hyper_param needs to be listed third position from the front of the argument list

C The variable length keyword argument list **grid_search_params needs to be listed at the back end of the argument list

D The non-default positional arguments iterations and cut_off need to be listed together at the front of the argument list

E The variable length keyword argument list **grid_search_params needs to be listed at the front of the argument list

TURING

Python
Question 30 of 30

You are working on a Python 3 program. Select the statement that is not true with respect to subprocesses in Python

Select one answer 00:13:34 Finis

A
Special value that can be used as the stdin, stdout or stderr argument to process creation functions. It indicates that the special file os.devnull will be used for the corresponding subprocess stream.

B
If asyncio.subprocess.PIPE is passed to stdin, stdout, or stderr arguments, the corresponding Process.stdin, Process.stdout or Process.stderr will point to StreamReader instances

C
Like Popen, Process instances do have an equivalent to the poll() method

D
Special value that can be used as the stderr argument and indicates that standard error should be redirected into standard output