Which of the following is **not** true?

A **type**...

(a) is a set of values & operations on these values

(b) represents something

(c) can be determined by using `type()` in Python

(d) can be changed by using `type()` in Python

(e) determines the meaning of an operation

If there are multiple false answers, pick one!

What does it mean that Python is **dynamically typed**?

(a) Variables can hold values of any type
(b) Variables can hold different types at the same time
(c) Variables can hold different types at different times
(d) A & B
(e) A & C

If this is what happens when I type the following code into python interactive mode:

```
C:\> python
>>> x = 1+2
>>> x = 3*x
>>> x
9
>>> print(x)
9
>>>
```

What gets printed when I run this script?

```
C:\> python script.py
```

```
# script.py
x = 1+2
x = 3*x
x
print(x)
```

The file called script.py

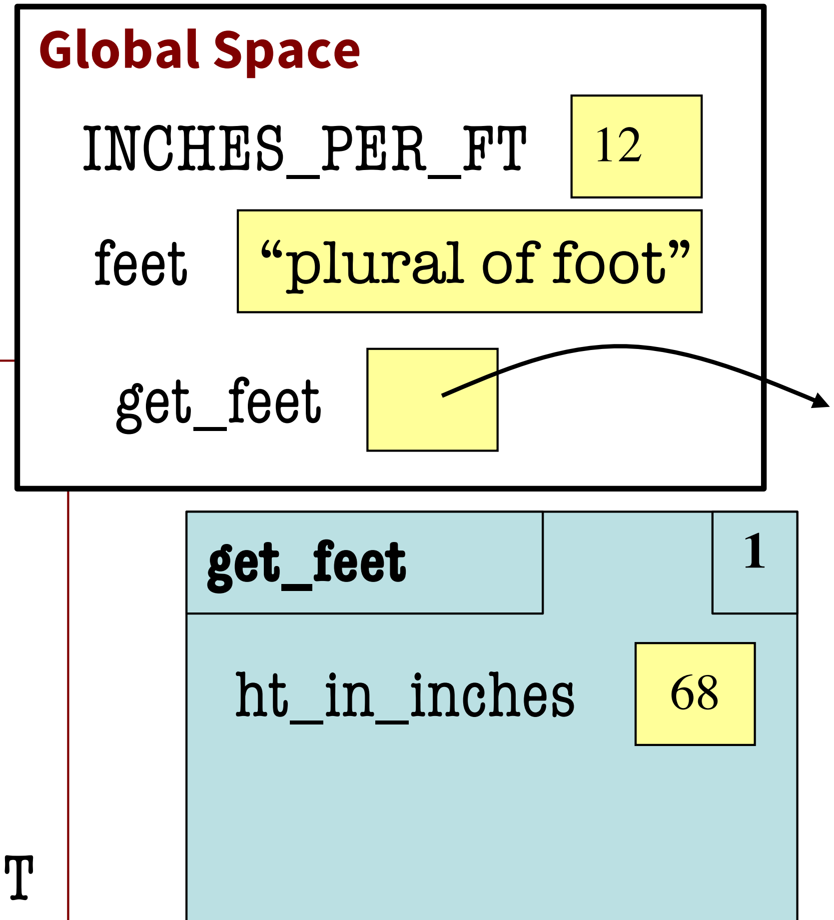
(a)
9
9

(b)
Error

(c)
9

(d)
No clue

How will the diagram change after executing line 1?




```
INCHES_PER_FT = 12

feet = “plural of foot”

...

def get_feet(ht_in_inches):
1       feet = ht_in_inches // INCHES_PER_FT
2       return feet

get_feet(68)
```

(a) line 1 generates an error (b) ??

(c) a new local variable feet is created in the call frame

(d) global variable feet gets a new value

```python
def foo(a,b):
    x = a
    y = b
    return x*y+y
```

The file called fn.py

```
C:\> python
>>> x = 2
>>> import fn
>>> fn.foo(3,4)
16
>>> x
...
```

What does Python give me?

A: 2
B: 3
C: 16
D: None
E: I do not know

Which of the following is true?

When testing you should…

(a) test a function exclusively with its most likely arguments
(b) write just a few tests with arguments that do not meet the function preconditions
(c) start by testing with inputs that live on the edges of multiple preconditions
(d) test every possible input you can think of
(e) write a bunch of tests before you even code up the function you're writing

What is in global p after calling swap?

```
import shapes
p = shapes.Point3(1,2,3)
q = shapes.Point3(3,4,5)

def swap(p, q):
    t = p
    p = q
    q = t

swap(p, q)
```

A: **id1**
B: **id2**
C: I don’t know

Heap Space

id1	**Point3**
x	1
y	2
z	3

id2	**Point3**
x	3
y	4
z	5

Global Space

p **id1** q **id2**

```
1  # Put max of x, y in z
2  print('before if’)
3  if x > y:
4      print(‘inside if x>y’)
5      z = x
6      print(‘z = ’+str(z))
7  else:
8      print(‘inside else (x<=y)’)
9      z = x
10     print(‘z = ’+str(z))
11 print('after if’)
12 print(“the max of "+str(x)+" and "+str(y)+" is "+str(y))
```

output to screen

```
before if
inside if x>y
z = 3
after if
the max of 3 and -3 is -3
```

Running the code on the left produces the output above.
What line has the bug?

A: 5 B: 9 C: 12 D: 9 & 12
E: this code is bug-free!

Q1: what does the call stack look like at this point in the execution of the code?

```
def f3():
    print("f3")
def f2():
    print("f2")
    f3()
    f3()
    f3()
def f1():
    print("f1")
    f2()
f1()
```

A	B	C	D	E
f1	f1	f1	f1	f1
f2	f2	f2	f2	
f3	f3	f3		
f3	f3			
f3				

Execute the following:

```
>>> x = [1, 2, 3, 4, 5]
>>> z = x
>>> y = x[1:3]
>>> z[y[0]] = x[0]
```

What is x[2]?

A: 1
B: 2
C: 3
D: **ERROR**
E: I don't know

```
1   # error.py
2
3   def function_1(x,y):
4       """ x, y are ints """
5       return function_2(x,y)
6
7   def function_2(x,y):
8       """ x, y are floats """
9       return function_3(x,y)
10
11  def function_3(x,y):
12      """ x, y are nums, y != 0 """
13      return x/y
14
15  function_1(1,0)
```

Crash produces call stack:

```
Traceback (most recent call last):
  File "error.py", line 15, in <module>
    function_1(1,0)
  File "error.py", line 5, in function_1
    return function_2(x,y)
  File "error.py", line 9, in function_2
    return function_3(x,y)
  File "error.py", line 13, in function_3
    return x/y
ZeroDivisionError: division by zero
```

Which line of code is to blame for the program crash?

A: 5   B: 9   C: 13   D: 15   E: multiple

Execute the following:

```
b = [1, 2, 3]
for a in b:
    b.append(a)
print b
```

What gets printed?

A: never prints b
B: [1, 2, 3, 1, 2, 3]
C: [1, 2, 3]
D: I do not know

What is this?

```
def song():
    print("This is the song that never ends.")
    print("Yes, it goes on and on my friend.")
    print("Some people started singing it, not knowing what it was,")
    print("And they'll continue singing it forever just because...")
    song()
```

A: A problem-free recursive function
B: A problematic recursive function
C: A song that will be stuck in my head for the rest of the day.
D: I do not know

What statement is false?

A) Recursion is *provably equivalent* to iteration (for-loops)

B) Recursion is *more powerful* than iteration (for-loops)

C) Some programming problems are easier to solve with recursion

D) Some programming problems are easier to solve with iteration (for-loops)

E) Recursion can be more memory intensive than iteration

Which statement is true?

```python
def num_ancestors(p):
    """Returns: num of known ancestors
    Pre: p is a Person"""

    parent1s = 0
    if p.parent1 != None:
        parent1s = 1+num_ancestors(p.parent1)

    parent2s = 0
    if p.parent2 != None:
        parent2s = 1+num_ancestors(p.parent2)

    return parent1s+parent2s
```

A) This code works fine!

B) This code won't work b/c parent1s and parent2s keep getting set to 0

C) This code won't work b/c there is no base case

D) This code won't work b/c not everyone person p has 2 parents.

E) I don't know.

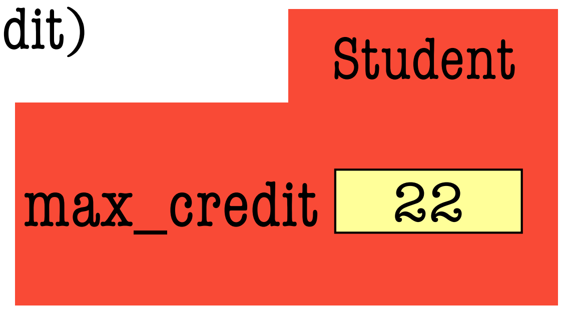
What is the difference between an instance attribute and a class attribute?

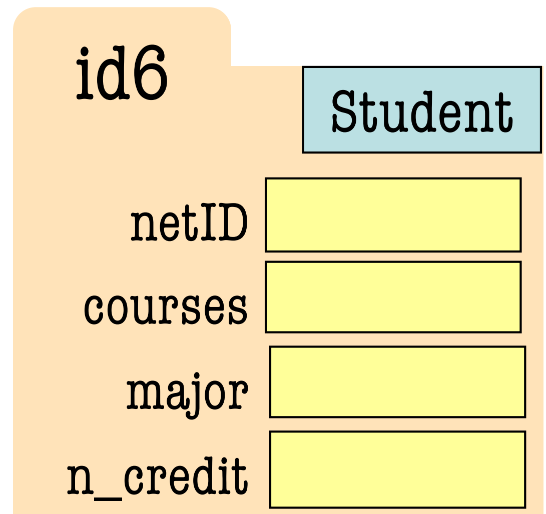
A) An instance attribute lives in Global Space.
B) Instance attributes can be modified, but class attributes cannot.
C) Class attributes cannot be accessed by class instances, but instance attributes can be.
D) There can be one copy of a class attribute but possibly many copies of instance attributes.
E) I don't know.

What gets Printed?

```
import cs1110

s1 = cs1110.Student(“jl200", [], "Art")
print(s1.max_credit)
s2 = cs1110.Student(“jl202", [], "History")
print(s2.max_credit)
s2.max_credit = 23
print(s1.max_credit)
print(s2.max_credit)
print(cs1110.Student.max_credit)
```

A:	B:	C:	D:
22	22	22	22
22	22	22	22
23	23	22	22
23	23	23	23
23	22	22	23





isinstance and Subclasses

```
class A():
   # definition here

class B(A):
  # definition here

class C(A):
  # definition here

class D(C):
  # definition here

class E(D):
  # definition here

class F(B):
  # definition here
```

Execute the following:

```
>>> b = B()
>>> e = E()
>>> x = isinstance(b, F)
>>> y = isinstance(e, D)
```

A: x is True, Y is True
B: x is False, Y is True
C: x is True, Y is False
D: x is False, Y is False
E: I don't know

Executing the following:

```
bigger_than_x = x + 1
```

Where can python look for the variable x?

- the current call frame
- the call frame of an earlier (still executing) function that called the current function
- the global space
- (if this line of code is inside a class method) an instance attribute
- (if this line of code is inside a class method) a class attribute

How many correct answers are there?

A: 1 B: 2 C: 3 D: 4 E: 5

On the hangman question of Prelim 1, why did we not ask you to replace **multiple** underscores with a guessed character in the `hidden` word?

A: You need a **for loop** to do that and it was too soon in the semester to ask that.
B: You need a **while loop** to do that and it was too in the semester to ask that.
C: You need either a **for loop** or a **while loop** to do that and it was too soon in the semester to ask that.
D: You *still* don't have the tools to do that.
E: I don’t know.

Consider a Person class with attributes children (a list of children) and n_male and n_female with the class invariant: n_male + n_female == len(children)

Think about how one would implement the class method add_child(self, is_male). What is true of this invariant?

A: It must be true after every line of add_child executes.

B: It must be true before and after add_child executes.

C: If the invariant is ever not true, Python will throw an error.

D: A & C

E: B & C

What range of s has been processed?

2. Write the command and equivalent postcondition
3. Write the basic part of the while-loop

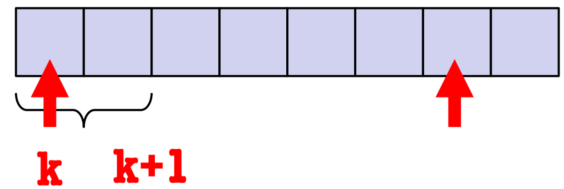
A: 0..k
B: 1..k
C: 0..k–1
D: 1..k–1
E: I don't know

```
# set n_pair to number of adjacent equal pairs in s

while k < len(s):

    k = k + 1
# POST: n_pair = # adjacent equal pairs in s[0..len(s)-1]
```

k: next integer to process.
What range of s has been processed?

What is the loop condition?

Compare s[k] to the character after it (s[k+1])




```
# set n_pair to # adjacent equal pairs in s
precondition: s is a string
n_pair = 0
k = 0
# INV: n_pair = # adjacent equal pairs in s[0..k]
while XXXXXXXXX:
    if (s[k] == s[k+1]):
        n_pair += 1
    k = k + 1
postcondition: n_pair = # adjacent equal pairs in s[0..len(s)-1]
```

A: k-1 < len(s)

B: k < len(s) - 1

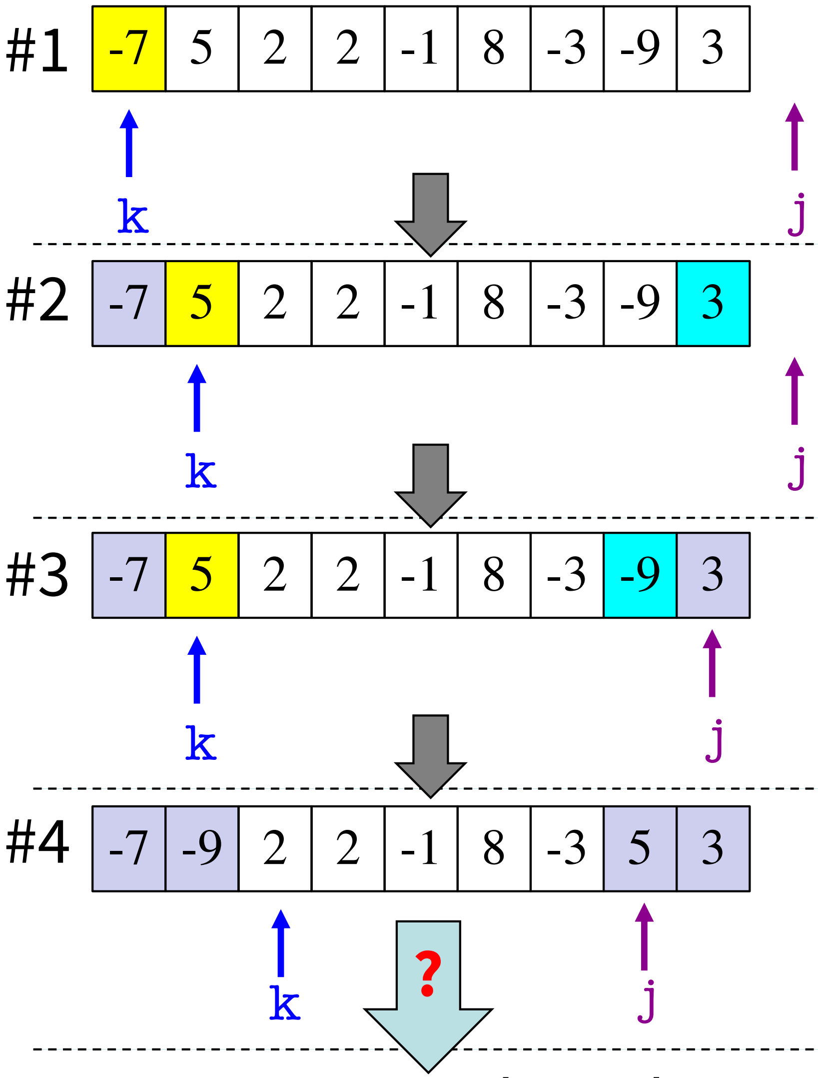
C: k < len(s)

D: k < len(s) + 1

E: I don't know

x Shaded elements have been processed

High Level Approach



Case A: inspect s[k]: stays where it is
→ just increment k

Case B: inspect s[k]: needs to be moved
inspect s[j-1]: stays where it is
→ just decrement j

Case C: inspect s[k]: needs to be moved
inspect s[j-1]: needs to be moved
→ swap the elements, increment k, decrement k

Iterations #1-#3 have been categorized.
What case is iteration #4?

A: Case A
B: Case B
C: Case C
D: a new Case
E: I don't know

What is your favorite Sorting Algorithm?

A) Selection Sort
B) Insertion Sort
C) Merge Sort
D) Bubble Sort
E) Quick Sort

https://www.youtube.com/watch?v=ZZuD6iUe3Pc

Answers

1) B & D
2) C
3) C
4) C
5) A
6) E
7) A
8) D
9) D
10) A
11) B
12) A
13) B
14) B
15) A
16) D
17) C
18) B
19) B
20) C
21) B
22) C
23) B
24) C
25) ?