DIGITAL TALENT SCHOLARSHIP

PROFESSIONAL ACADEMY

Tema Pelatihan

Python for Data Professional Beginner

KOMINFO

#JADIJAGOANDIGITAL

Badan Penelitian dan Pengembangan Sumber Daya Manusia

PROFIL PENGAJAR

Evan Budianto

Latar Belakang Pendidikan:
Bachelor's Degree in Informatics Engineering - ITB

Riwayat Pekerjaan:

Telkomsel

Data Scientist
2020 - present

Orbit by.U

Kontak:
evanbudianto@gmail.com
twitter.com/EvanBudianto

Content

1. Introduction to Python and computer programming
2. Data types, variables, basic input-output operations, basic operators
3. Boolean values, conditional execution, loops, lists and list processing, logical and bitwise operations
4. Functions, tuples, dictionaries, and data processing

What is Python?

Python is a **high-level programming language**, with applications in numerous areas, including web programming, scripting, scientific computing, and artificial intelligence.

Python is processed at runtime by the **interpreter**. There is no need to compile your program before executing it.

What is Python?

There are two different ways of transforming a program from a high-level programming language into machine language:

COMPILATION - the source program is translated once (however, this act must be repeated each time you modify the source code) by getting a file (e.g., an .exe file if the code is intended to be run under MS Windows) containing the machine code; now you can distribute the file worldwide; the program that performs this translation is called a compiler or translator;

INTERPRETATION - you (or any user of the code) can translate the source program each time it has to be run; the program performing this kind of transformation is called an interpreter, as it interprets the code every time it is intended to be executed; it also means that you cannot just distribute the source code as-is, because the end-user also needs the interpreter to execute it.

How to Run Python on Your PC?

ANACONDA
jupyter
SciPy
SPYDER
NumPy
DASK
bokeh
scikit learn
pandas
TensorFlow
Numba
Panel
matplotlib
HoloViews
PYTORCH
CONDA

Anaconda is popular because it brings many of the tools used in **data science and machine learning** with just one install, so it's great for having short and simple setup.

Data Types

Integers (or simply ints) are one of the numerical types supported by Python. They are numbers written without a fractional component, e.g., 256, or -1 (negative integers).

Floating-point numbers (or simply floats) are another one of the numerical types supported by Python. They are numbers that contain (or are able to contain) a fractional component, e.g., 1.27.

Data Types

To encode an apostrophe or a quote inside a string you can either use the escape character, e.g., 'I\'m happy.', or open and close the string using an opposite set of symbols to the ones you wish to encode, e.g., "I'm happy." to encode an apostrophe, and 'He said "Python", not "typhoon"' to encode a (double) quote.

Boolean values are the two constant objects True and False used to represent truth values (in numeric contexts 1 is True, while 0 is False).

Operators – Data Manipulation Tools

Operators are special symbols or keywords which are able to operate on the values and perform (mathematical) operations, e.g., the * operator multiplies two values: x * y.

Arithmetic operators in Python:

- + (addition),
- - (subtraction),
- * (multiplication),
- / (classic division - always returns a float),
- % (modulus - divides left operand by right operand and returns the remainder of the operation, e.g., 5 % 2 = 1),
- ** (exponentiation - left operand raised to the power of right operand, e.g., 2 ** 3 = 2 * 2 * 2 = 8),
- // (floor/integer division - returns a number resulting from division, but rounded down to the nearest whole number, e.g., 3 // 2.0 = 1.0)

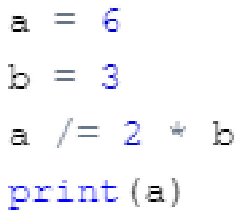
Variables

A **variable** is a named location reserved to store values in the memory. A variable is created or initialized automatically when you assign a value to it for the first time.

Each variable must have a unique name - an **identifier**. A legal identifier name must be a non-empty sequence of characters, must begin with the underscore(_), or a letter, and it cannot be a Python keyword. The first character may be followed by underscores, letters, and digits. Identifiers in Python are case-sensitive.

Variables

What is the output of the following snippet?



```
a = 6
b = 3
a /= 2 * b
print(a)
```

Variables

What is the output of the following snippet?

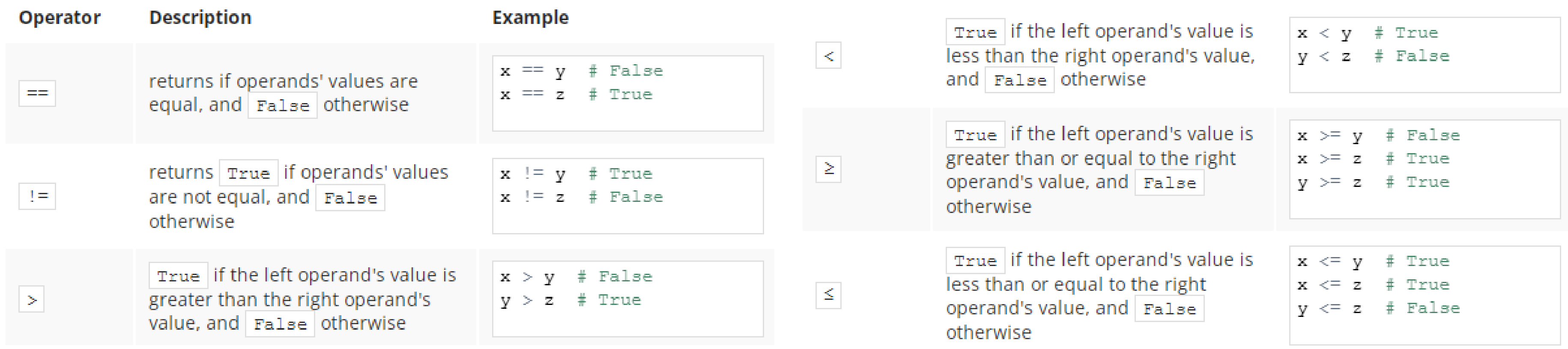
```
a = 6
b = 3
a /= 2 * b
print(a)
```

```
1.0
```

2 * b = 6
a = 6 → 6 / 6 = 1.0

Comparison and Conditional Execution

The **comparison** (or the so-called relational) operators are used to compare values. The table below illustrates how the comparison operators work, assuming that x = 0, y = 1, and z = 0:



Operator	Description	Example

`==`	returns if operands' values are equal, and `False` otherwise	`x == y  # False`
		`x == z  # True`

`!=`	returns `True` if operands' values are not equal, and `False` otherwise	`x != y  # True`
		`x != z  # False`

`>`	`True` if the left operand's value is greater than the right operand's value, and `False` otherwise	`x > y  # False`
		`y > z  # True`

`<`	`True` if the left operand's value is less than the right operand's value, and `False` otherwise	`x < y  # True`
		`y < z  # False`

`≥`	`True` if the left operand's value is greater than or equal to the right operand's value, and `False` otherwise	`x >= y  # False`
		`x >= z  # True`
		`y >= z  # True`

`≤`	`True` if the left operand's value is less than or equal to the right operand's value, and `False` otherwise	`x <= y  # True`
		`x <= z  # True`
		`y <= z  # False`

Comparison and Conditional Execution

When you want to execute some code only if a certain condition is met, you can use a **conditional statement**:

```python
x = 10

if x == 10: # condition
    print("x is equal to 10") # Executed if the condition is True.
```

Comparison and Conditional Execution

When you want to execute some code only if a certain condition is met, you can use a **conditional statement**:

```
x = 10

if x == 10: # True
    print("x == 10")

if x > 15: # False
    print("x > 15")
elif x > 10: # False
    print("x > 10")
elif x > 5: # True
    print("x > 5")
else:
    print("else will not be executed")
```

Comparison and Conditional Execution

What is the output of the following snippet?

```
x, y, z = 5, 10, 8

print(x > z)
print((y - 5) == x)
```

Comparison and Conditional Execution

What is the output of the following snippet?

```
x, y, z = 5, 10, 8

print(x > z)
print((y - 5) == x)
```

```
False
True
```

Comparison and Conditional Execution

What is the output of the following snippet?

```
x = 10

if x == 10:
    print(x == 10)

if x > 5:
    print(x > 5)

if x < 10:
    print(x < 10)
else:
    print("else")
```

Comparison and Conditional Execution

What is the output of the following snippet?

```
x = 10

if x == 10:
    print(x == 10)

if x > 5:
    print(x > 5)

if x < 10:
    print(x < 10)
else:
    print("else")
```

```
True
True
else
```

Comparison and Conditional Execution

What is the output of the following snippet?

```
x = 1
y = 1.0
z = "1"

if x == y:
    print("one")
if y == int(z):
    print("two")
elif x == y:
    print("three")
else:
    print("four")
```

Comparison and Conditional Execution

What is the output of the following snippet?

```
x = 1
y = 1.0
z = "1"

if x == y:
    print("one")
if y == int(z):
    print("two")
elif x == y:
    print("three")
else:
    print("four")
```

```
one
two
```

Loops

There are two types of loops in Python: **while** and **for**:

The **while** loop executes a statement or a set of statements as long as a specified boolean condition is true, e.g.:

```
counter = 5
while counter > 2:
    print(counter)
    counter -= 1
```

The **for** loop executes a set of statements many times; it's used to iterate over a sequence (e.g., a list, a dictionary, a tuple, or a set) or other objects that are iterable (e.g., strings).

```
for i in range(3):
    print(i, end=" ") # Outputs: 0 1 2

for i in range(1, 10):
    if i % 2 == 0:
        print(i)
```

Loops

You can use the **break** and **continue** statements to change the flow of a loop:

You use **break** to exit a loop, e.g.:

```
text = "OpenEDG Python Institute"

for letter in text:
    if letter == "P":
        break
    print(letter, end="")
```

You use **continue** to skip the current iteration, and continue with the next iteration, e.g.:

```
text = "pyxpyxpyx"

for letter in text:
    if letter == "x":
        continue
    print(letter, end="")
```

Logical Operations

Python supports the following logical operators:

and → if both operands are true, the condition is true, e.g., (True and True) is True,
or → if any of the operands are true, the condition is true, e.g., (True or False) is True,
not → returns false if the result is true, and returns true if the result is false, e.g., not True is False.

Logical Operations

What is the output of the following snippet?

```
x = 1
y = 0

z = ((x == y) and (x == y)) or not(x == y)
print(not(z))
```

Lists

The **list is a type of data** in Python used to **store multiple objects**. It is an **ordered and mutable collection** of comma-separated items between square brackets, e.g.:

```
my_list = [1, None, True, "I am a string", 256, 0]
```

Lists

Lists can be **indexed and updated**, e.g.:

```
my_list = [1, None, True, 'I am a string', 256, 0]
print(my_list[3]) # outputs: I am a string
print(my_list[-1]) # outputs: 0

my_list[1] = '?'
print(my_list) # outputs: [1, '?', True, 'I am a string', 256, 0]

my_list.insert(0, "first")
my_list.append("last")
print(my_list)
# outputs: ['first', 1, '?', True, 'I am a string', 256, 0, 'last']
```

Lists

List elements and lists can be **deleted**, e.g.:

```
my_list = [1, 2, 3, 4]
del my_list[2]
print(my_list) # outputs: [1, 2, 4]
del my_list # deletes the whole list
```

Lists can be **iterated** through using the for loop, e.g.:

```
my_list = ["white", "purple", "blue", "yellow", "green"]
for color in my_list:
print(color)
```

Lists

The **len()** function may be used to check the list's length, e.g.:

```
my_list = ["white", "purple", "blue", "yellow", "green"]
print(len(my_list)) # outputs 5
del my_list[2]
print(len(my_list)) # outputs 4
```

Function

A function is a block of code that performs a specific task when the function is called (invoked). You can use functions to make your code reusable, better organized, and more readable. Functions can have parameters and return values.

You can define your own function using the def keyword:

```
def hello(name):    # defining a function
    print("Hello,", name)    # body of the function


name = input("Enter your name: ")

hello(name)    # calling the function
```

Dictionaries

Each dictionary is a set of key: value pairs. You can create it by using the following syntax:

```
my_dictionary = {
    key1: value1,
    key2: value2,
    key3: value3,
    }
```

Dictionaries

If you want to access a dictionary item, you can do so by making a reference to its key inside a pair of square brackets (ex. 1) or by using the get() method (ex. 2):

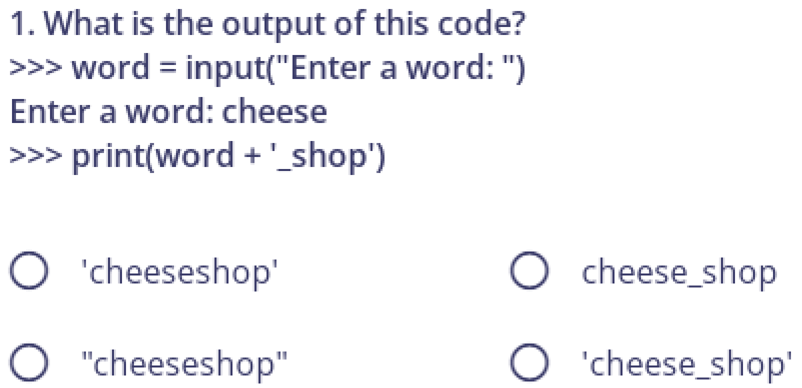
```
pol_eng_dictionary = {
"kwiat": "flower",
"woda": "water",
"gleba": "soil"
}

item_1 = pol_eng_dictionary["gleba"] # ex. 1
print(item_1) # outputs: soil

item_2 = pol_eng_dictionary.get("woda")
print(item_2) # outputs: water
```

Dictionaries

If you want to change the value associated with a specific key, you can do so by referring to the item's key name in the following way:

```
pol_eng_dictionary = {
"zamek": "castle",
"woda": "water",
"gleba": "soil"
}

pol_eng_dictionary["zamek"] = "lock"
item = pol_eng_dictionary["zamek"]
print(item) # outputs: lock
```

Quiz

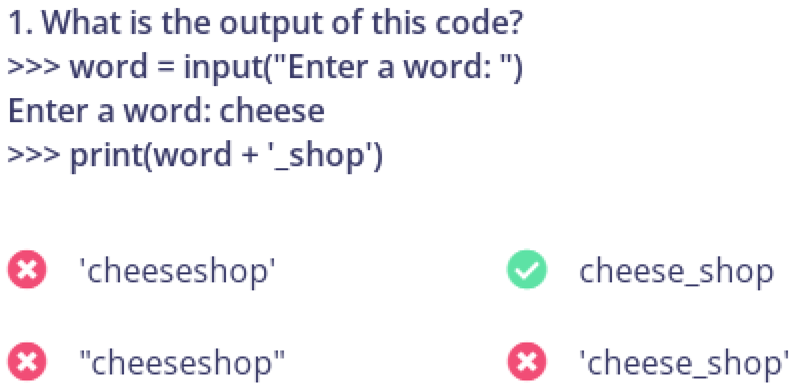


1. What is the output of this code?

```
>>> word = input("Enter a word: ")
Enter a word: cheese
>>> print(word + '_shop')
```

- ○ 'cheeseshop'
- ○ cheese_shop
- ○ "cheeseshop"
- ○ 'cheese_shop'

Quiz

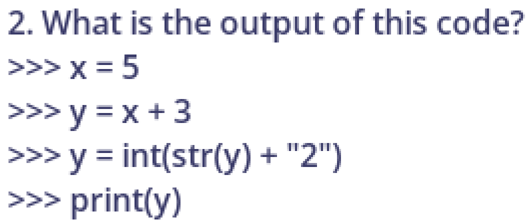


1. What is the output of this code?

```
>>> word = input("Enter a word: ")
Enter a word: cheese
>>> print(word + '_shop')
```

- ✗ 'cheeseshop'
- ✓ cheese_shop
- ✗ "cheeseshop"
- ✗ 'cheese_shop'

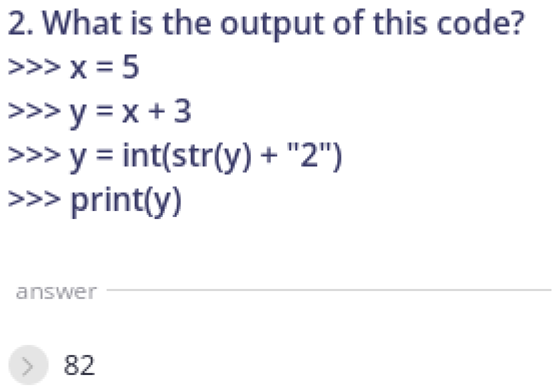
Quiz



2. What is the output of this code?

```
>>> x = 5
>>> y = x + 3
>>> y = int(str(y) + "2")
>>> print(y)
```

Quiz



2. What is the output of this code?

```
>>> x = 5
>>> y = x + 3
>>> y = int(str(y) + "2")
>>> print(y)
```

answer

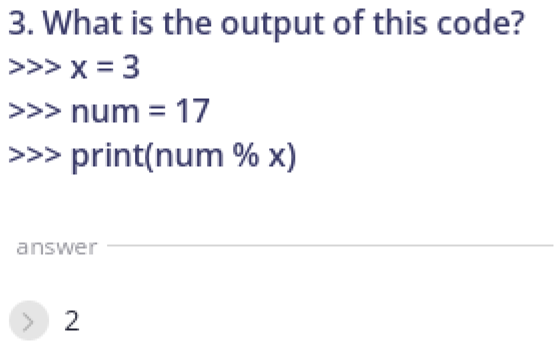
82

Quiz

3. What is the output of this code?

```
>>> x = 3
>>> num = 17
>>> print(num % x)
```

Quiz



3. What is the output of this code?

```
>>> x = 3
>>> num = 17
>>> print(num % x)
```

answer

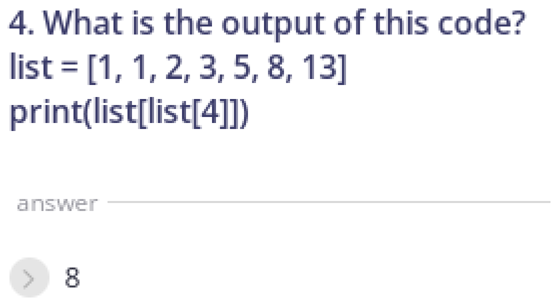
```
2
```

Quiz

4. What is the output of this code?

```
list = [1, 1, 2, 3, 5, 8, 13]
print(list[list[4]])
```

Quiz



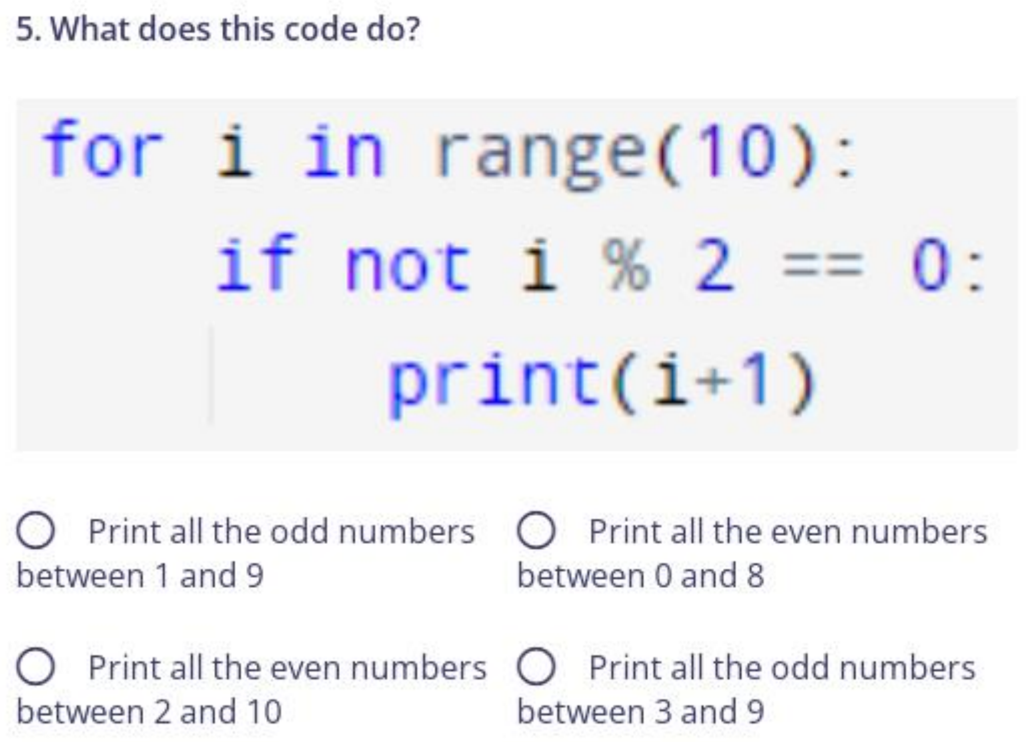
4. What is the output of this code?

```
list = [1, 1, 2, 3, 5, 8, 13]
print(list[list[4]])
```

answer

```
8
```

Quiz



5. What does this code do?

```
for i in range(10):
    if not i % 2 == 0:
        print(i+1)
```

○ Print all the odd numbers between 1 and 9

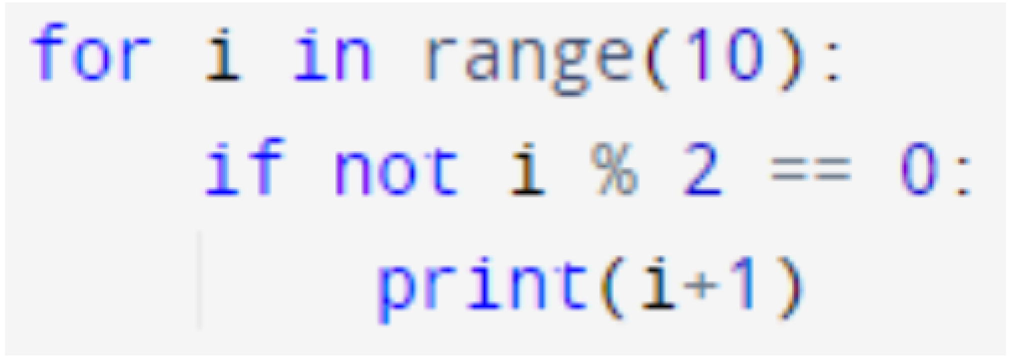
○ Print all the even numbers between 0 and 8

○ Print all the even numbers between 2 and 10

○ Print all the odd numbers between 3 and 9

Quiz

5. What does this code do?



```
for i in range(10):
    if not i % 2 == 0:
        print(i+1)
```

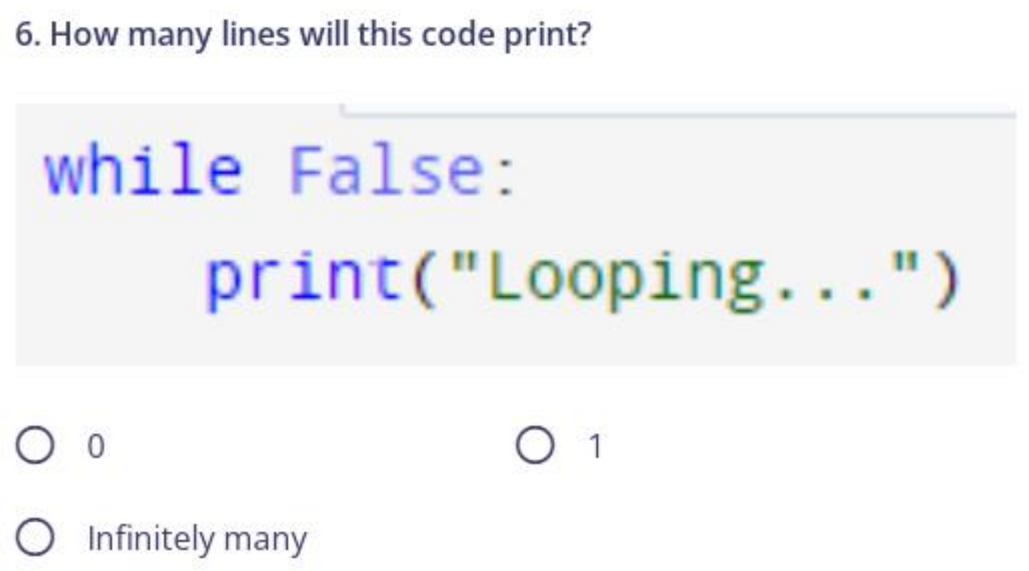
✗ Print all the odd numbers between 1 and 9

✗ Print all the even numbers between 0 and 8

✓ Print all the even numbers between 2 and 10

✗ Print all the odd numbers between 3 and 9

Quiz



6. How many lines will this code print?

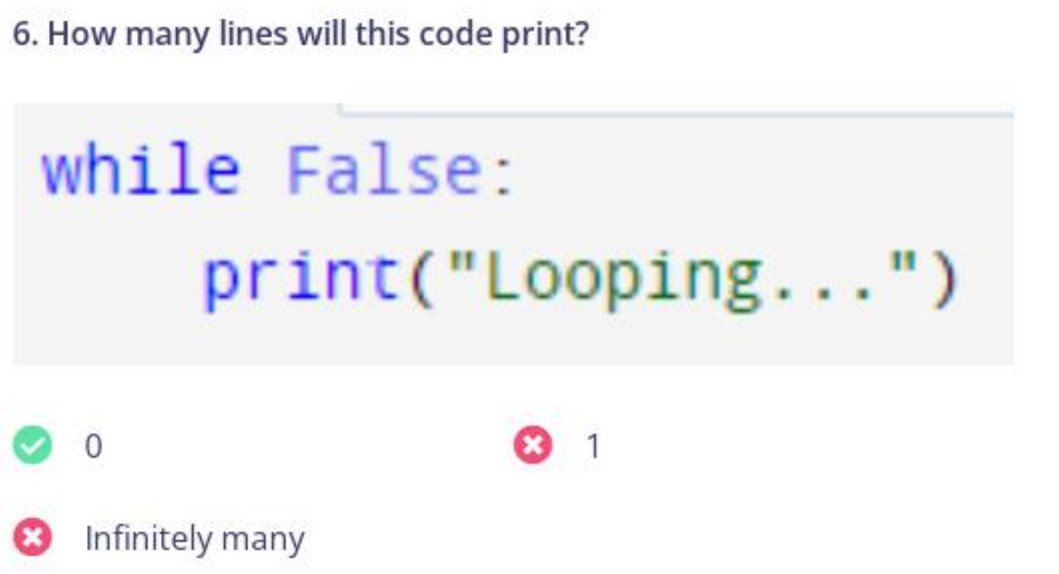
```
while False:
    print("Looping...")
```

○ 0

○ 1

○ Infinitely many

Quiz



6. How many lines will this code print?

```
while False:
    print("Looping...")
```

✓ 0

✗ 1

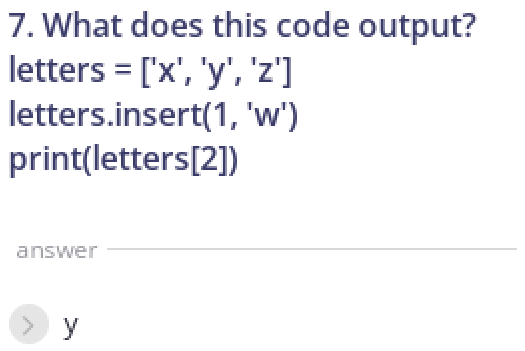
✗ Infinitely many

Quiz

7. What does this code output?

```
letters = ['x', 'y', 'z']
letters.insert(1, 'w')
print(letters[2])
```

Quiz



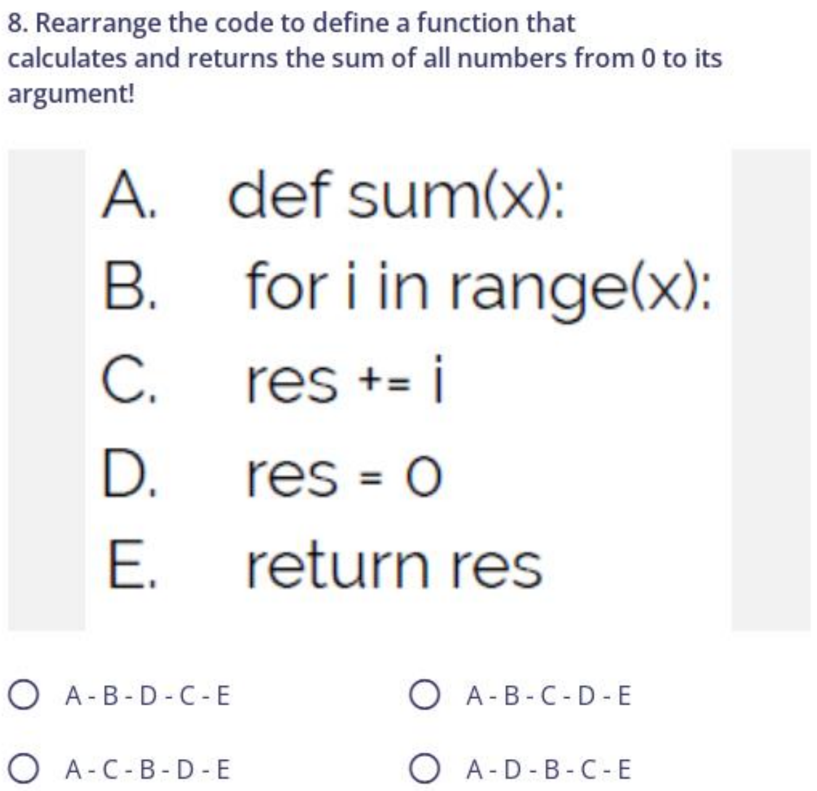
7. What does this code output?

```
letters = ['x', 'y', 'z']
letters.insert(1, 'w')
print(letters[2])
```

answer

y

Quiz

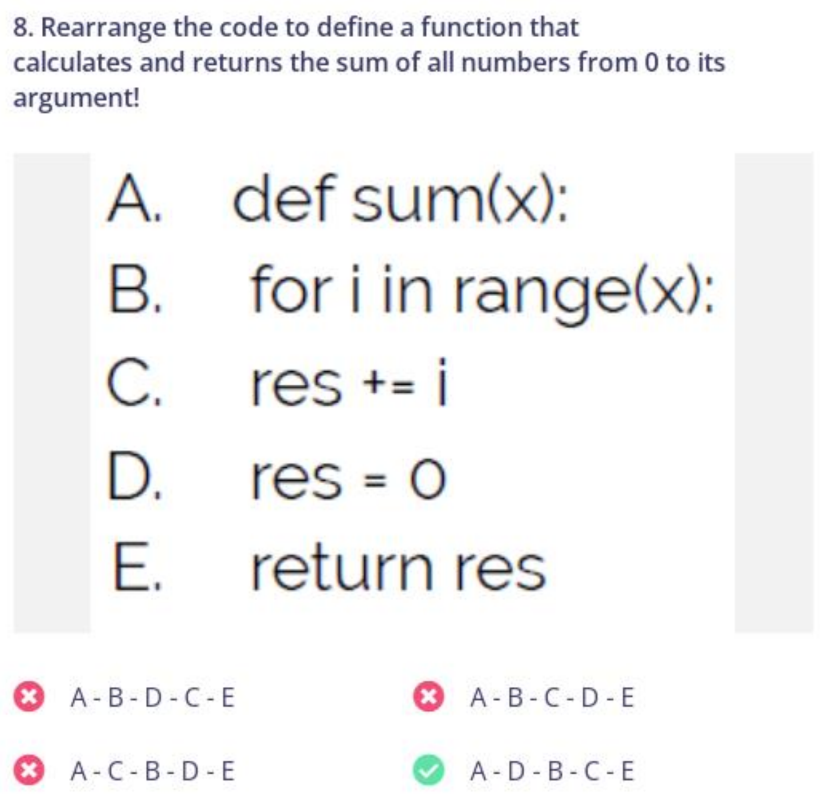


8. Rearrange the code to define a function that calculates and returns the sum of all numbers from 0 to its argument!

```
A.   def sum(x):
B.    for i in range(x):
C.    res += i
D.    res = 0
E.    return res
```

- ○ A - B - D - C - E
- ○ A - B - C - D - E
- ○ A - C - B - D - E
- ○ A - D - B - C - E

Quiz



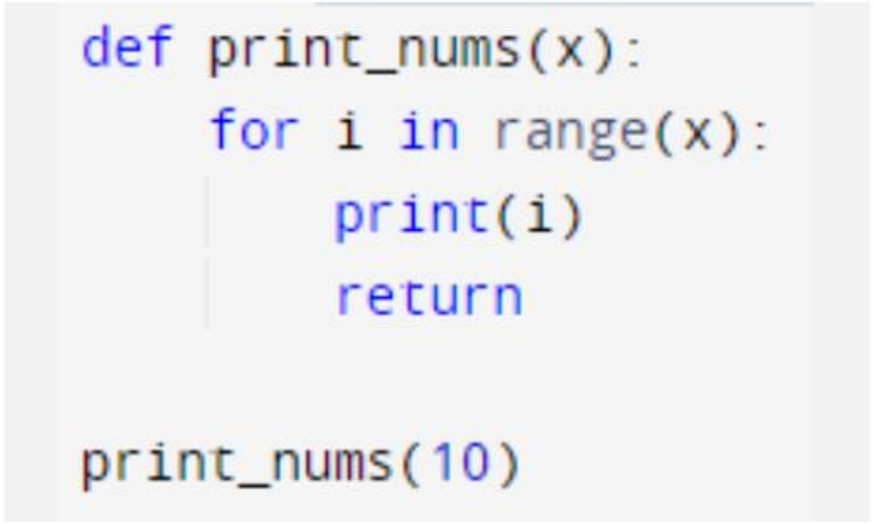
8. Rearrange the code to define a function that calculates and returns the sum of all numbers from 0 to its argument!

```
A.  def sum(x):
B.    for i in range(x):
C.    res += i
D.    res = 0
E.    return res
```

✗ A - B - D - C - E	✗ A - B - C - D - E
✗ A - C - B - D - E	✓ A - D - B - C - E

Quiz

9. What is the highest number output by this code?

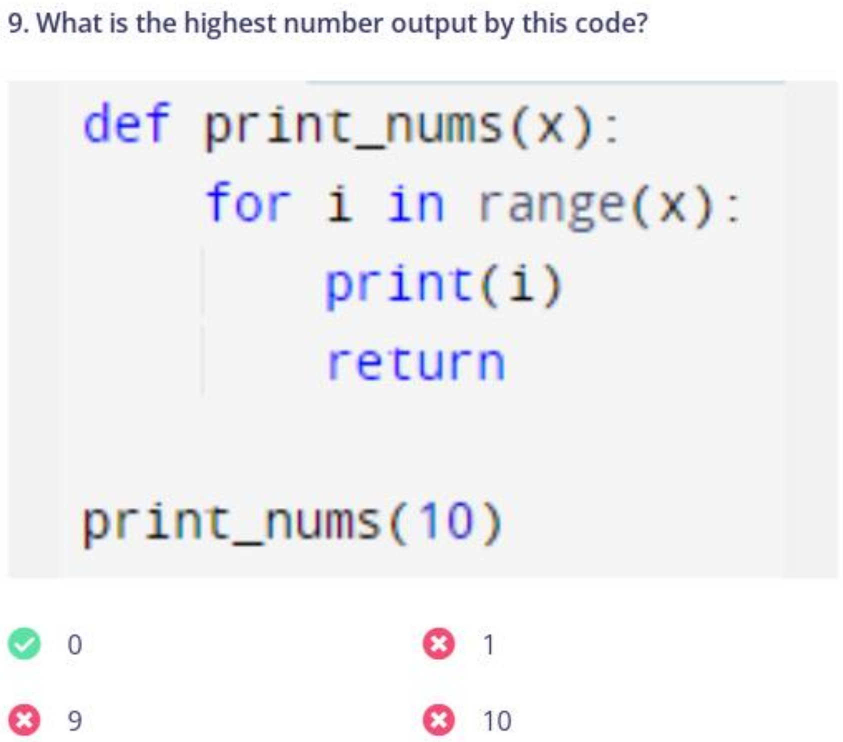


```
def print_nums(x):
    for i in range(x):
        print(i)
        return

print_nums(10)
```

○ 0   ○ 1

○ 9   ○ 10

Quiz



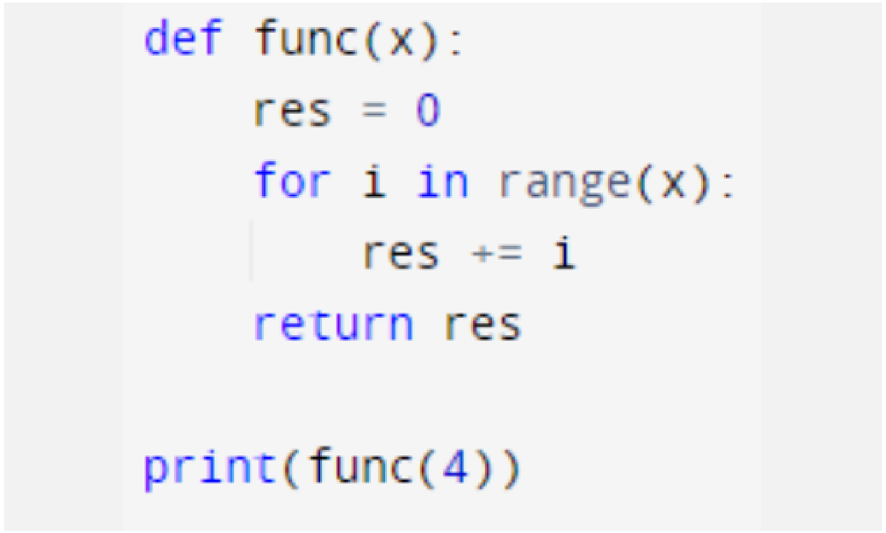
9. What is the highest number output by this code?

```
def print_nums(x):
    for i in range(x):
        print(i)
        return

print_nums(10)
```

✓ 0	✗ 1
✗ 9	✗ 10

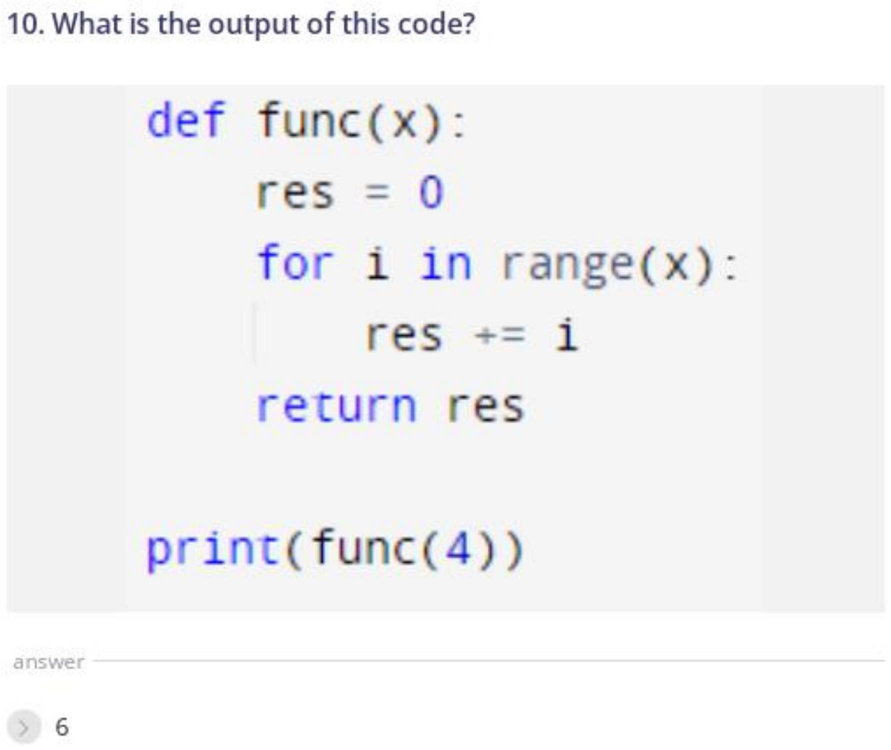
Quiz

10. What is the output of this code?



```python
def func(x):
    res = 0
    for i in range(x):
        res += i
    return res

print(func(4))
```

Quiz



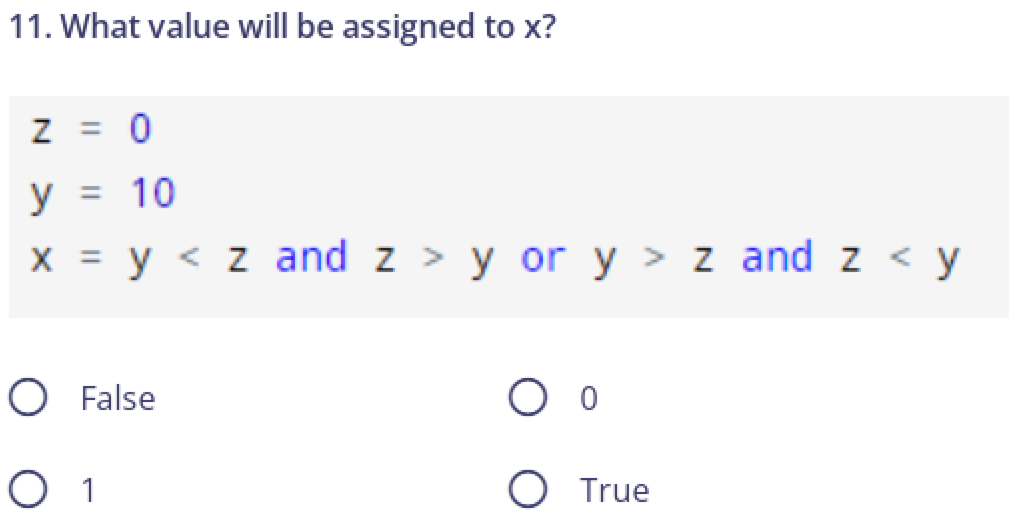
10. What is the output of this code?

```python
def func(x):
    res = 0
    for i in range(x):
        res += i
    return res

print(func(4))
```

answer

6

Quiz



11. What value will be assigned to x?

```
z = 0
y = 10
x = y < z and z > y or y > z and z < y
```

- False
- 0
- 1
- True

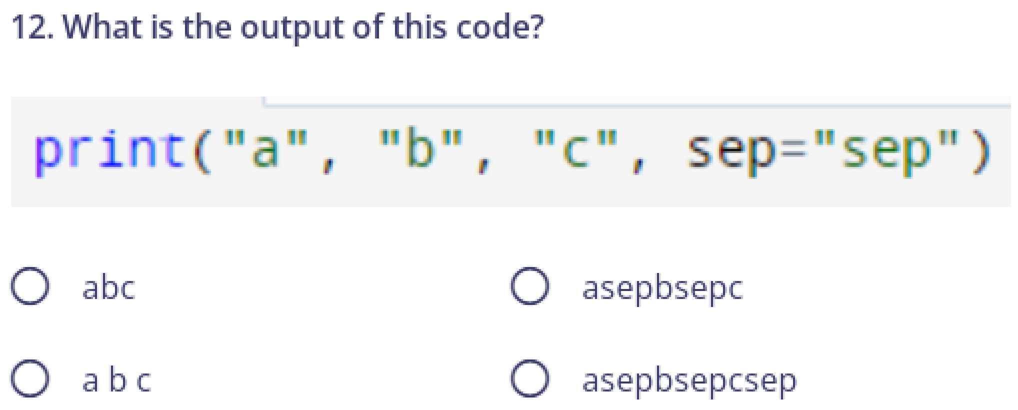
Quiz

11. What value will be assigned to x?

```
z = 0
y = 10
x = y < z and z > y or y > z and z < y
```

- ✗ False
- ✗ 0
- ✗ 1
- ✓ True

Quiz

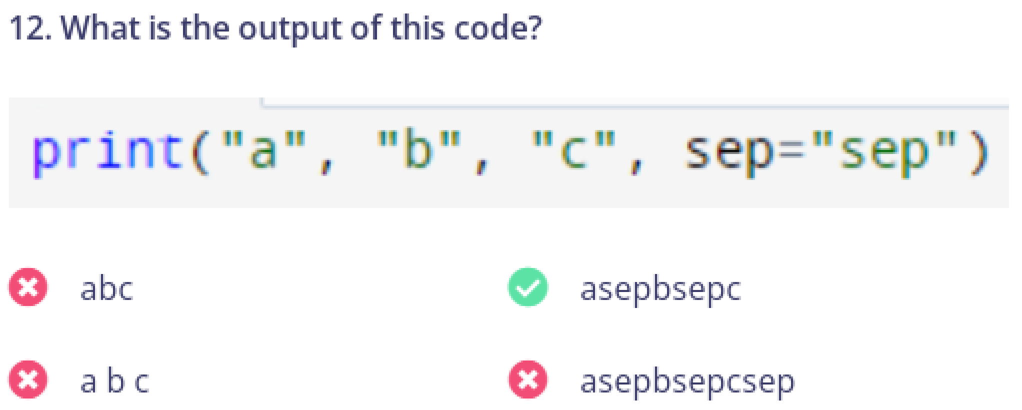


12. What is the output of this code?

```
print("a", "b", "c", sep="sep")
```

- ○ abc
- ○ asepbsepc
- ○ a b c
- ○ asepbsepcsep

Quiz



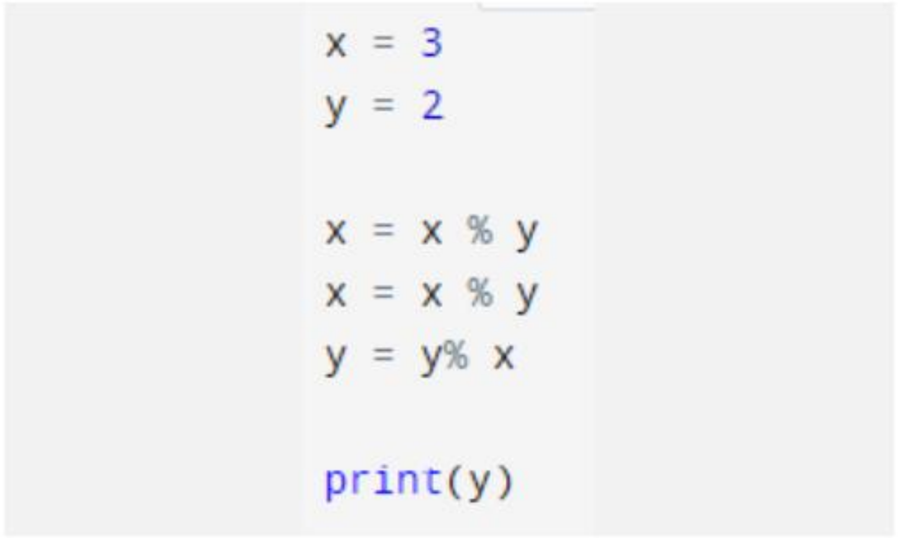
12. What is the output of this code?

```python
print("a", "b", "c", sep="sep")
```

✗ abc	✓ asepbsepc
✗ a b c	✗ asepbsepcsep

Quiz

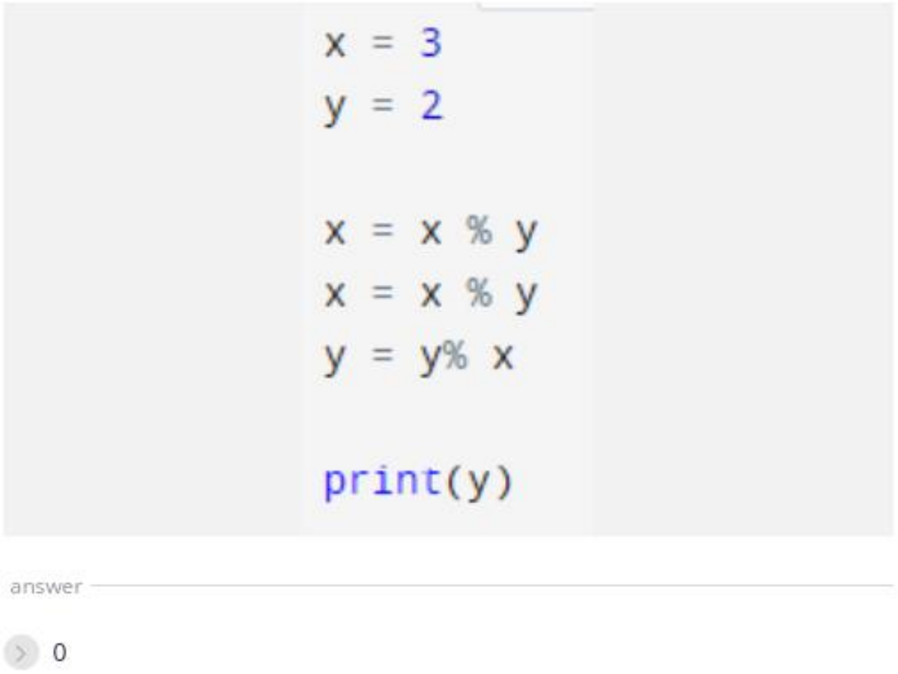
13. What is the output of this code?



```
x = 3
y = 2

x = x % y
x = x % y
y = y% x

print(y)
```

Quiz

13. What is the output of this code?



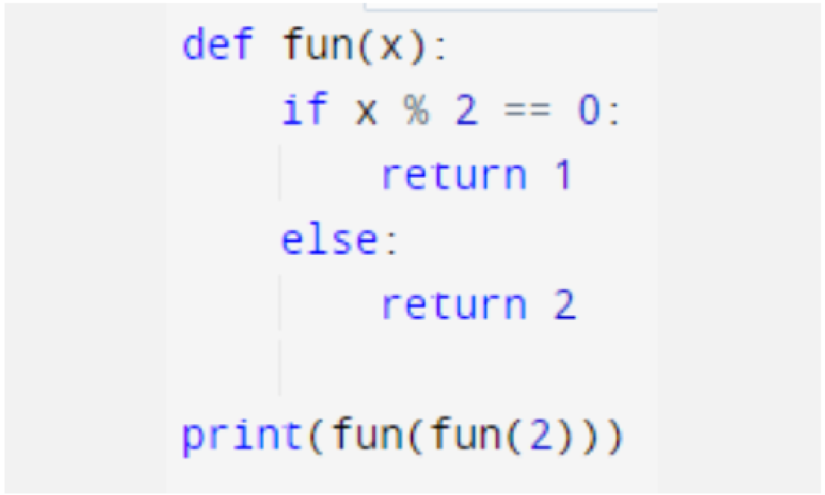
```
x = 3
y = 2

x = x % y
x = x % y
y = y% x

print(y)
```

answer

```
0
```

Quiz

14. What is the output of this code?



```
def fun(x):
    if x % 2 == 0:
        return 1
    else:
        return 2


print(fun(fun(2)))
```

○ 1

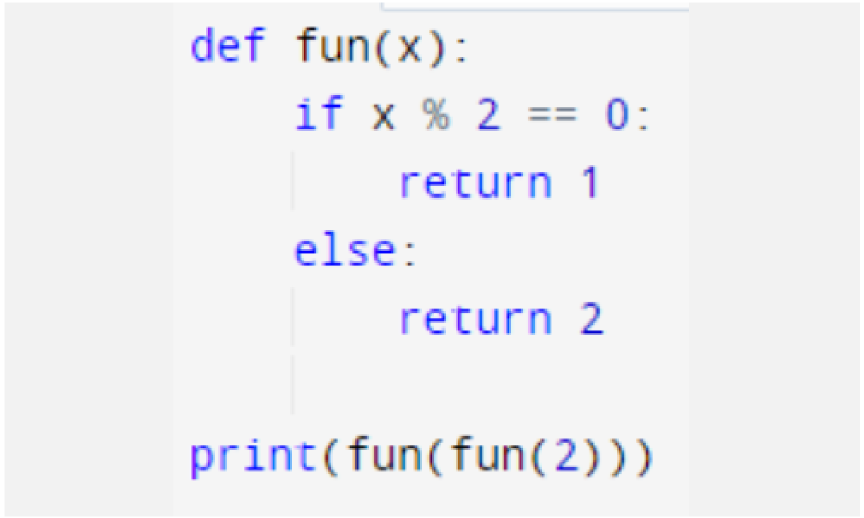
○ 2

○ The code will cause a runtime error

○ 2None

Quiz

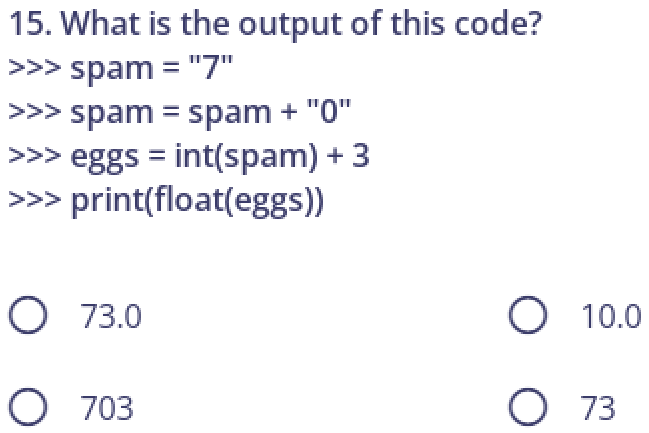
14. What is the output of this code?



```python
def fun(x):
    if x % 2 == 0:
        return 1
    else:
        return 2

print(fun(fun(2)))
```

✗ 1	✓ 2
✗ The code will cause a runtime error	✗ 2None

Quiz



15. What is the output of this code?

```
>>> spam = "7"
>>> spam = spam + "0"
>>> eggs = int(spam) + 3
>>> print(float(eggs))
```

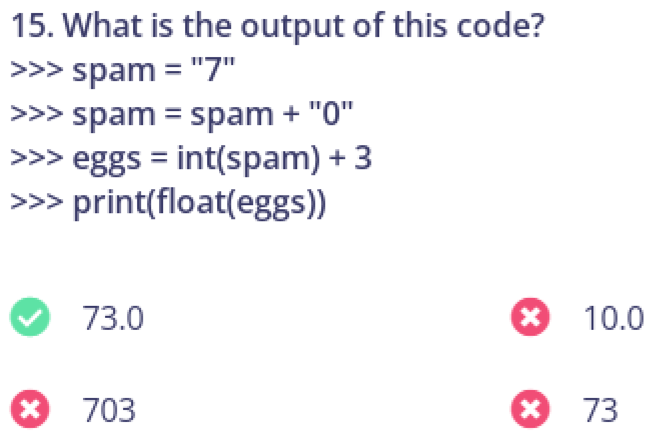
○ 73.0

○ 10.0

○ 703

○ 73

Quiz



15. What is the output of this code?

```
>>> spam = "7"
>>> spam = spam + "0"
>>> eggs = int(spam) + 3
>>> print(float(eggs))
```

✓ 73.0	✗ 10.0
✗ 703	✗ 73

HANDS-ON

#JADIJAGOANDIGITAL

TERIMA KASIH

digitalent.kominfo

DTS_kominfo

digitalent.kominfo

digital talent scholarship