Technische Universität Braunschweig

Institut für rechnergestützte Modellierung im Bauingenieurwesen

Algorithms & Programming

Lecture 3: Data types, conditionals & loops

Prof. Dr.-Ing. Henning Wessels

Exam

When? March 3, 2023, 11am-12pm

Where? 4303.02.206C - Zi 24.3
Zimmerstraße 24 c - 24 d (4303)

https://vorlesungen.tu-braunschweig.de/qisserver/rds?state=verpublish&status=init&vmfile=no&moduleCall=webInfo&publishConfFile=webInfoRaum&publishSubDir=raum&keep=y&raum.rgid=79152

Question

A. I was able to do the exercises and work out any problems on my own.
B. I have questions regarding the exercises.
C. I haven‘t looked at the exercises.
D. I wasn‘t able to do the execises at all.

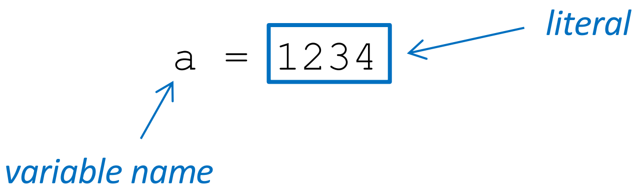
Outline

- Data types
- **Conditionals and loops**

Basic Built-in Data Types & Operators

Variables

- A variable is a name that we use to refer to a data-type value.
- We use variables to keep track of changing values as a computation unfolds.
- Each variable always stores one of the permissible data-type values.



```
a = 1234
```

Basic Built-in Data Types & Operators

Identifiers

- We use identifiers to name individual variables
- An identifier is a sequence of letters, digits, _, and $, the first of which is not a digit.
- You cannot use reserved language keywords – such as `class, def, if` … etc.

🙂 `abc, Ab$, abc123, a_b`

🙁 `Ab*, 1abc, a+b`

Basic Built-in Data Types & Operators

Literals

- A literal is a source-code representation of a data-type value.
- We use strings of digits like `1234` or `99` to define `int` literal values,
- add a decimal point as in `3.14159` or `2.71834` to define `float` literal values,
- keywords `True` or `False` to specify `bool` values and
- a sequence of characters enclosed in quotes, such as `“Hello, World”` to specify a `str` (string)

Basic Built-in Data Types & Operators

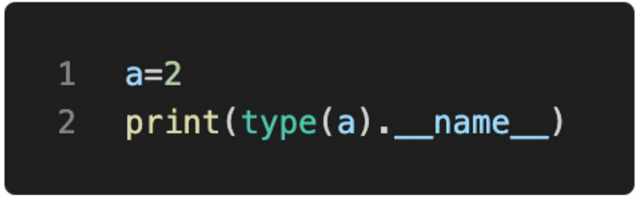
Declaration statements

- A declaration statement associates a variable name with a type at compile time.
- Java requires us to use declarations to specify the names and types of variables.

declaration statement →

```
int a, b;
```

- Python dynamically declares variables at assignment



```
a=2
print(type(a).__name__)
```

→

```
Returns int!
```

Basic Built-in Data Types & Operators

Assignment statements

- An assignment statement associates a data-type value with a variable.
- The left-hand side of an assignment statement must be one or multiple variables.
- The right-hand side can be an arbitrary expression that produces a value of the expected type.
- The meaning of = is decidedly not the same as in mathematical equations (see slide: Comparisons).

```
b = 99
```

← *assignment statement*

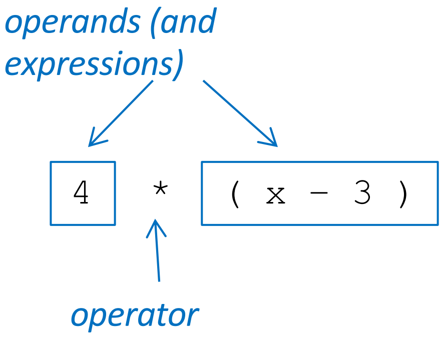
```
a, b = 99, “Hello World‟
```

← *multiple assignments*

Basic Built-in Data Types & Operators

Expressions

- An expression is a literal, a variable, or a sequence of operations on literals and/or variables that produce a value
- Expressions are based on *operators* that specify data-type operations to be performed on one or more *operands*



Basic Built-in Data Types & Operators

Strings

- A String (`str`) is a sequence of characters within double quotes
- for tab, newline, backslash, single quote and double quote to be used the special *escape sequences* `‘\t’, ‘\n’, ‘\\’, ‘\’’`, *and* `‘\”’`
- The characters are encoded as 8-bit integers using an encoding scheme known as Unicode https://de.wikipedia.org/wiki/Unicode

values	sequence of characters
typical literals	“Hello, “ “1 “ “ * “
operation	concatenate
operator	+

expression	*value*
`“Hi, “ + “Bob”`	`“Hi, Bob”`

Basic Built-in Data Types & Operators

Integers and Floating-point numbers

- An `int` is an integer (natural number)
- The `float` type is for representing *floating-point* numbers

values	any integer				
typical literals	1234 99 -99 0 1000000				
values	real numbers				
typical literals	3.14159 6.022e23 -3.0 2.0 1.414211356237309 1e5				
operation	add	subtract	multiply	divide	remainder
operator	+	-	*	/	%

Basic Built-in Data Types & Operators

Booleans

- A `bool` type has just two values: true and false.

values	true or false		
literals	True or False		
operation	and	or	not
operator	and	or	not

a	not a
True	False
False	True

a	b	a and b	a or b
False	False	False	False
False	True	False	True
True	False	False	True
True	True	True	True

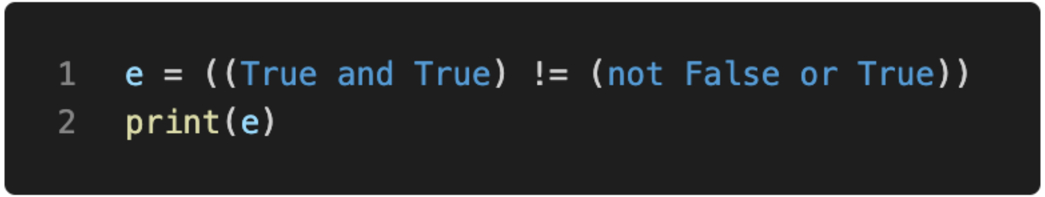
Basic Built-in Data Types & Operators

Comparisons

- The comparison operators are defined for each primitive numeric type and produce a `bool` result

operator	meaning	true	false
==	equal	2 == 2	2 == 3
!=	not equal	3 != 2	2 != 2
<	less than	2 < 13	2 < 2
<=	less than or equal	2 <= 2	3 <= 2
>	greater than	13 > 2	2 > 13
>=	greater than or equal	3 >= 2	2 >= 3

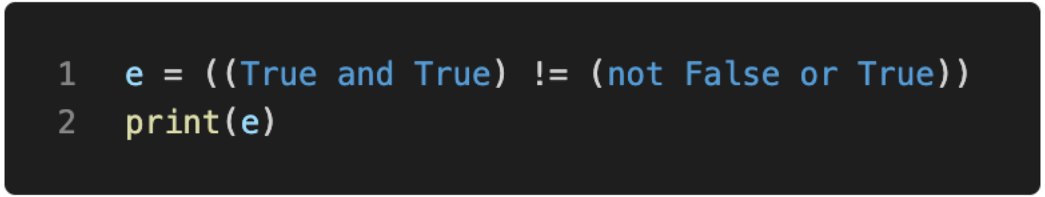
What is the output of this python code?



```python
e = ((True and True) != (not False or True))
print(e)
```

A. True
B. False
C. No idea

What is the output of this python code?



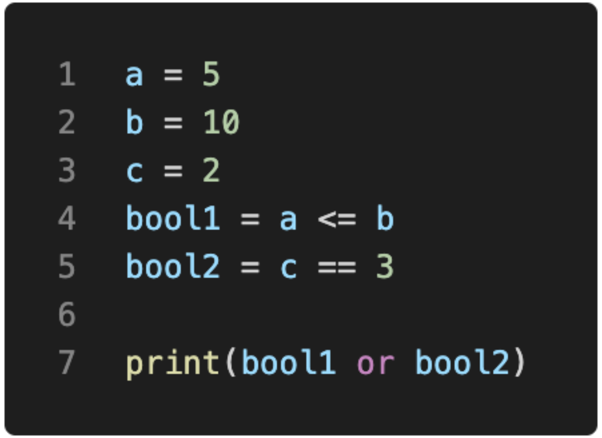
```python
e = ((True and True) != (not False or True))
print(e)
```

A. True

B. False

C. No idea

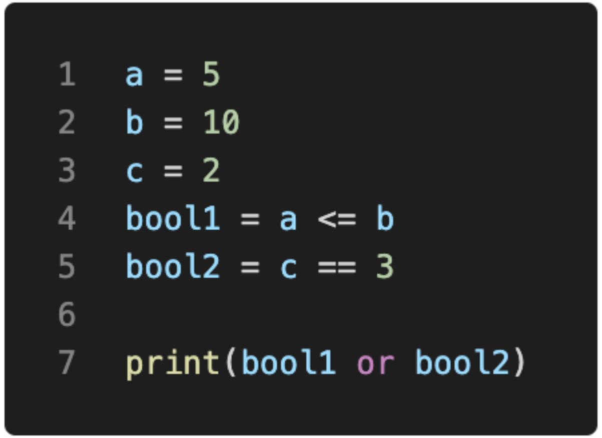
What is the output of this python code?



```python
a = 5
b = 10
c = 2
bool1 = a <= b
bool2 = c == 3

print(bool1 or bool2)
```

A. True
B. False
C. No Idea

What is the output of this python code?



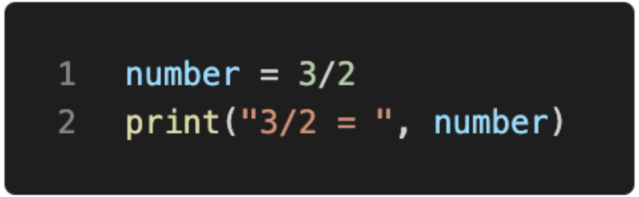
```python
a = 5
b = 10
c = 2
bool1 = a <= b
bool2 = c == 3

print(bool1 or bool2)
```

A. True

B. False

C. No Idea

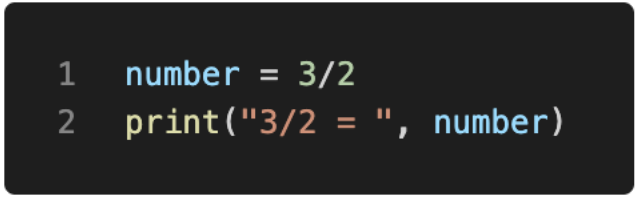
What is the output of this python code?



```python
number = 3/2
print("3/2 = ", number)
```

A. 3/2 = 1.5
B. 3/2 = 0.0
C. 3/2 = 1.0
D. No idea

What is the output of this python code?



```python
number = 3/2
print("3/2 = ", number)
```

A. 3/2 = 1.5
B. 3/2 = 0.0
C. 3/2 = 1.0
D. No idea

Basic Built-in Data Types & Operators

Type conversion

- Explicit type conversion: `int(), float(), str()`
- Automatic promotion for numbers

expression	expression type	expression value
“1234” + str(99)	str	“123499”
int(“123”)	int	123
int(2.71828)	int	2
11 * 0.3	float	3.3
int(11) * 0.3	float	3.3
11 * int(0.3)	int	0
int(11 * 0.3)	int	3

Basic Built-in Data Types & Operators

Type	Description
bool	True or False.
int	Integer numbers of arbitrary size.
float	Double precision floating point numbers
str	Immutable string of characters
tuple	Immutable sequence of arbitrary objects
set	Mutable collection of unique objects
list	Mutable collection of arbitrary objects
dict	Mutable collection of key-value pairs

Basic Built-in Data Types & Operators

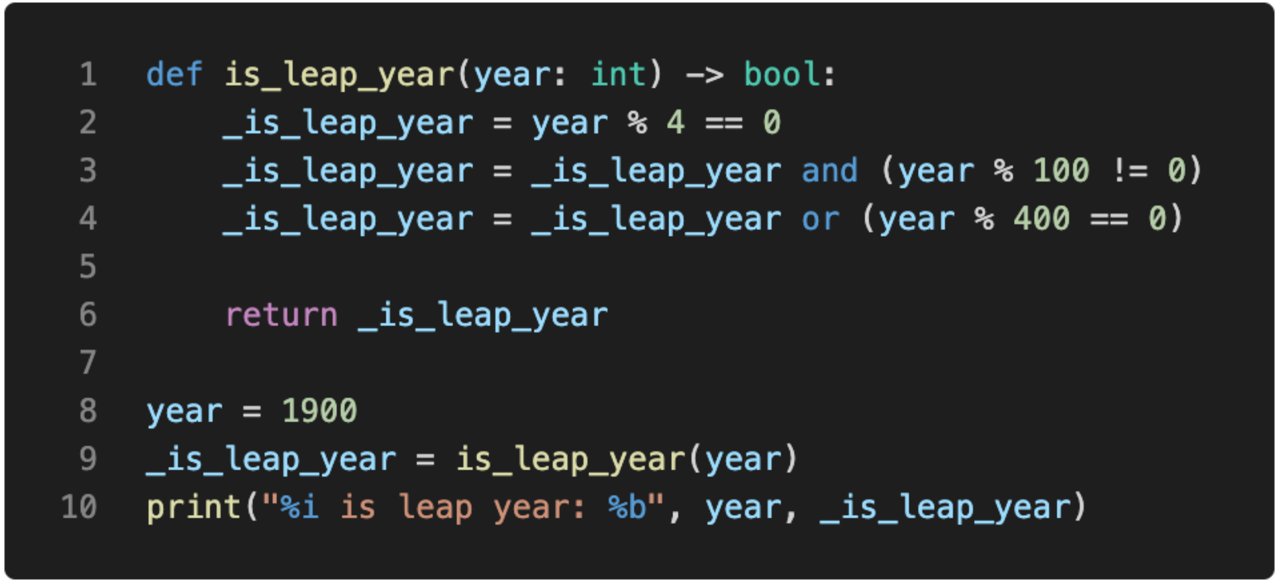
Write a function that …

- returns true if N corresponds to a **leap year**, and false otherwise.
- assumes N >= 1582, corresponding to a year in the Gregorian calendar.

In the Gregorian calendar, each leap year has 366 days instead of 365, by extending February to 29 days rather than the common 28.

These extra days occur in each year that is an integer multiple of 4 (except for years evenly divisible by 100, but not by 400).

Basic Built-in Data Types & Operators



```python
def is_leap_year(year: int) -> bool:
    _is_leap_year = year % 4 == 0
    _is_leap_year = _is_leap_year and (year % 100 != 0)
    _is_leap_year = _is_leap_year or (year % 400 == 0)

    return _is_leap_year

year = 1900
_is_leap_year = is_leap_year(year)
print("%i is leap year: %b", year, _is_leap_year)
```

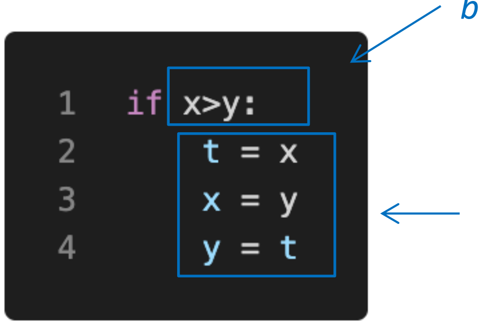
Outline

- Data types
- **Conditionals and loops**

Conditionals and Loops

If statements

```
if <boolean expression>:
    <statements>

else if <boolean expression>:
    <statements T>
else:
    <statements F>
```



```
if x>y:
    t = x
    x = y
    y = t
```

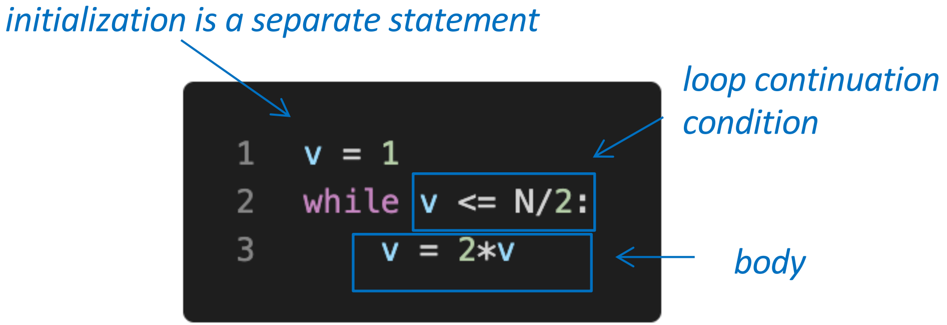
boolean expression

Sequence of statements

Conditionals and Loops

While loops

```
while <boolean expression>:
    <statements>
```



initialization is a separate statement

loop continuation condition

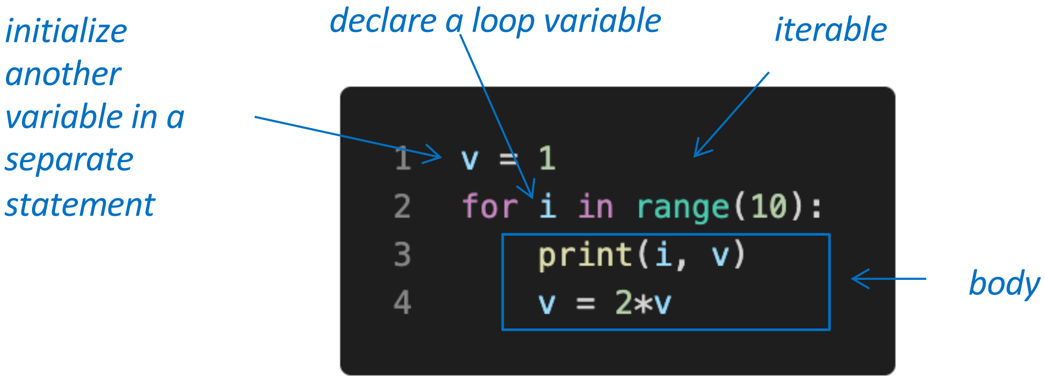
body

```
v = 1
while v <= N/2:
    v = 2*v
```

Conditionals and Loops

For loops

```
for element in <iterable>:
    <statements>
```



```
v = 1
for i in range(10):
    print(i, v)
    v = 2*v
```

initialize another variable in a separate statement

declare a loop variable

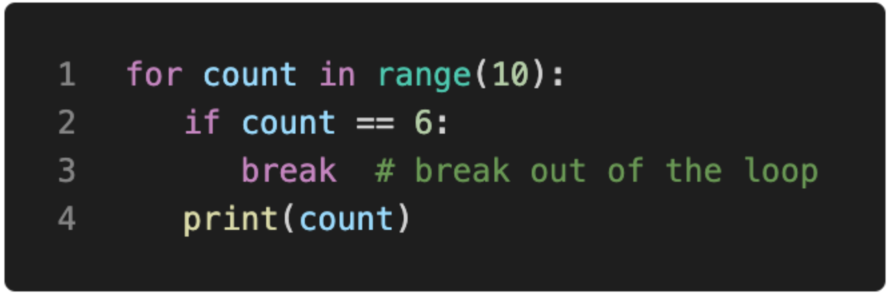
iterable

body

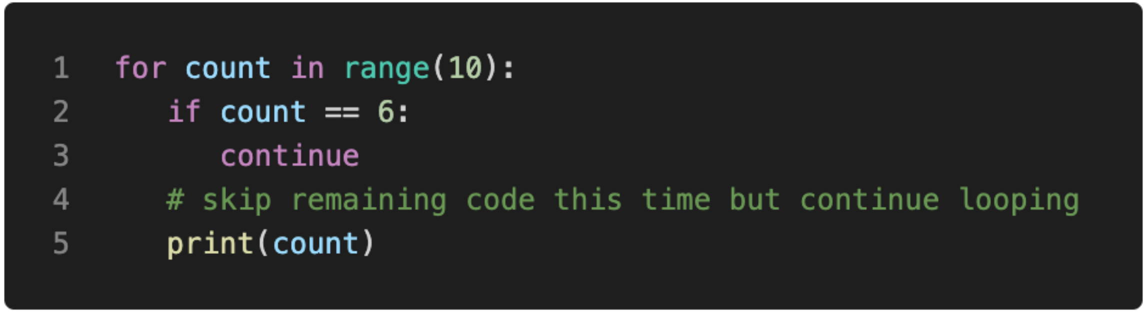
Conditionals and Loops

Break and continue statement

- A `break` statement exits the loop without executing the rest of the code in the loop.
- A `continue` statement skips the rest of the code in the loop and moves on to the next iteration.

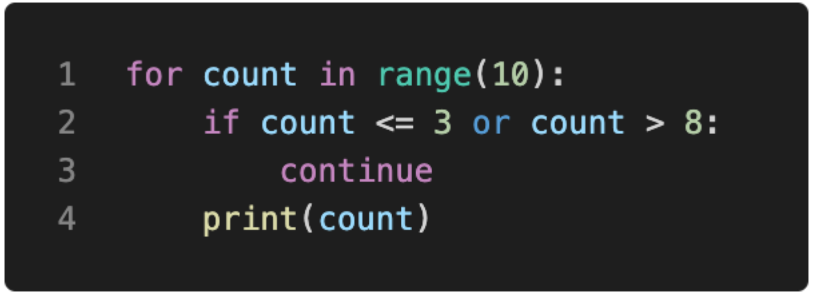


```python
for count in range(10):
    if count == 6:
        break  # break out of the loop
    print(count)
```



```python
for count in range(10):
    if count == 6:
        continue
    # skip remaining code this time but continue looping
    print(count)
```

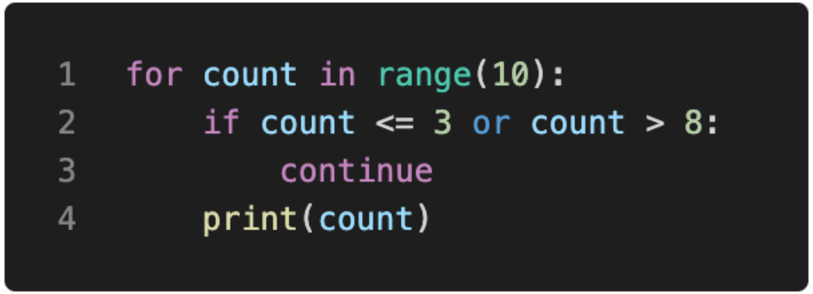
What is the output of this code?



```python
for count in range(10):
    if count <= 3 or count > 8:
        continue
    print(count)
```

A. 4, 5, 6, 7
B. 4, 5, 6, 7, 8
C. 3, 4, 5, 6, 7
D. No Idea

What is the output of this code?



```python
for count in range(10):
    if count <= 3 or count > 8:
        continue
    print(count)
```

A. 4, 5, 6, 7

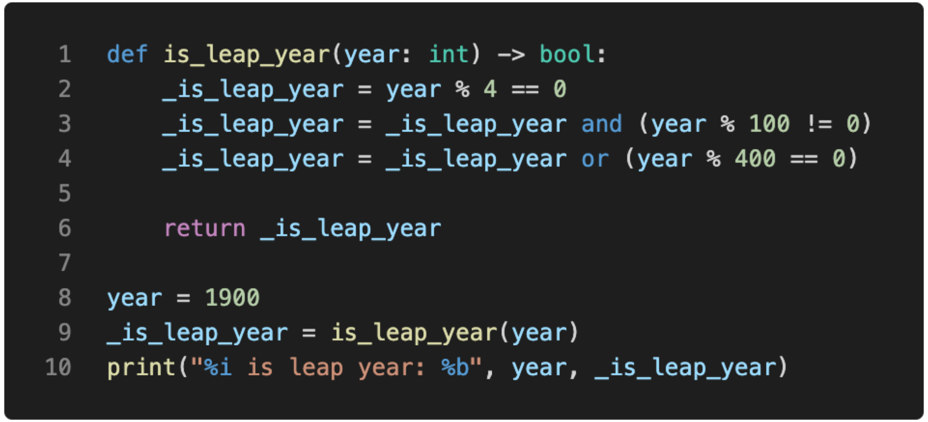
B. 4, 5, 6, 7, 8

C. 3, 4, 5, 6, 7

D. No Idea

Homework

Use if statements and break conditions in order to speed up the function is_leap_year below



```python
def is_leap_year(year: int) -> bool:
    _is_leap_year = year % 4 == 0
    _is_leap_year = _is_leap_year and (year % 100 != 0)
    _is_leap_year = _is_leap_year or (year % 400 == 0)

    return _is_leap_year

year = 1900
_is_leap_year = is_leap_year(year)
print("%i is leap year: %b", year, _is_leap_year)
```

Conditionals and Loops

Prime Factorization

- A *prime* is an integer greater than one whose only positive divisors are one and itself.
- The **prime factorization** of an integer is the multiset of primes whose product is the integer.
- For example, 3757208 = 2*2*2*7*13*13*397

This computation would not be feasible without the help of a computer. Imagine you wanted to find the factors of a number like 287994837222311

Conditionals and Loops

Prime factorization by trial division

Given an integer *n* (*n* refers to "the integer to be factored"), the trial division consists of systematically testing whether *n* is divisible by any smaller number.

Conditionals and Loops

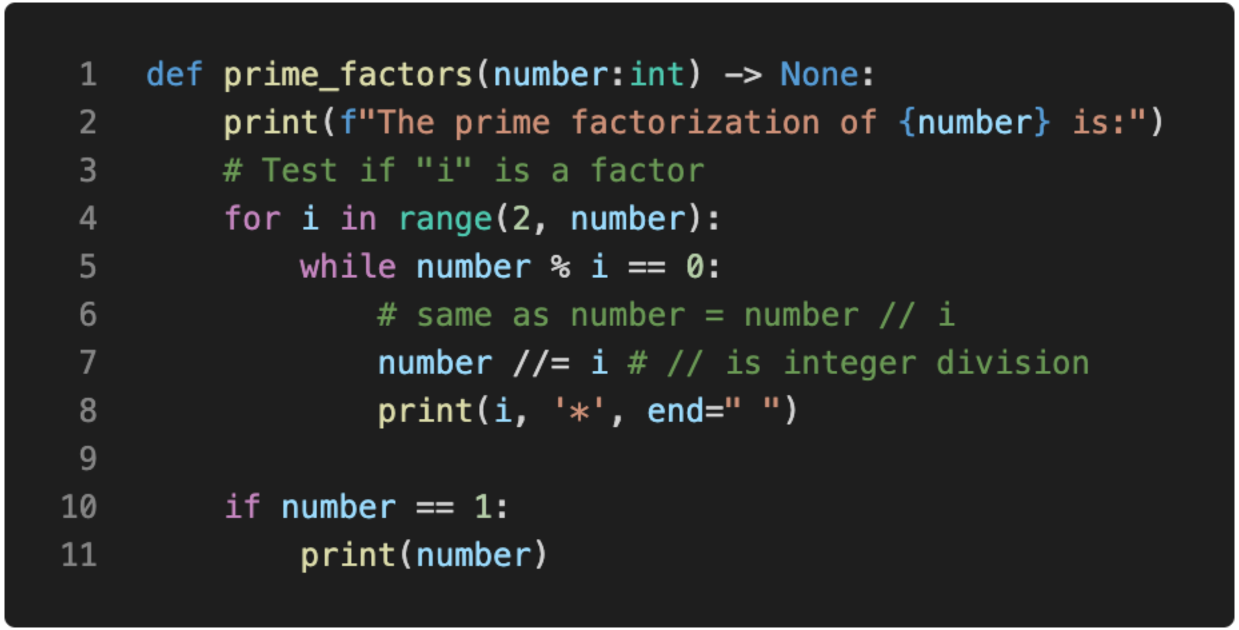
Prime factorization by trial division

Given an integer *n* (*n* refers to "the integer to be factored"), the trial division consists of systematically testing whether *n* is divisible by any smaller number.

Clearly, it is only worthwhile to test candidate factors less than *n*, and in order from two upwards because an arbitrary n is more likely to be divisible by two than by three, and so on. With this ordering, there is no point in testing for divisibility by four if the number has already been determined not divisible by two, and so on for three and any multiple of three, etc. Therefore, the effort can be reduced by selecting only prime numbers as candidate factors.

https://en.wikipedia.org/wiki/Trial_division

Conditionals and Loops



```python
def prime_factors(number:int) -> None:
    print(f"The prime factorization of {number} is:")
    # Test if "i" is a factor
    for i in range(2, number):
        while number % i == 0:
            # same as number = number // i
            number //= i # // is integer division
            print(i, '*', end=" ")

    if number == 1:
        print(number)
```

Handwritten illustration

Conditionals and Loops

Example: N = 140

i = 2
- 140 % 2 == 0
- number = 140/2
- 70 % 2 == 0
- number = 70/2
- 35 % 2 = 1

i = 3
- 35 % 3 = 2

i = 4
- 35 % 4 = 3

i = 5
- 35 % 5 = 0
- number = 35/5

i = 6
- 7 % 6 = 5

i = 7
- 7 % 7 = 0
- number = 7/7

Result

140 = 2 * 2 * 5 * 7

Questions

A. Yes
B. No
C. Of course!
D. I did not understand the question…