Computational Thinking

Computationeel Denken - T1ACD2E

Exercise Session 2: Variables and Control Structures

This is the assignment for the second exercise session of computational thinking. Firstly, you process the theory and coding videos (see Toledo). Next, you tackle these exercises.

Make sure you downloaded Thonny, see instructions on Toledo (previous session).

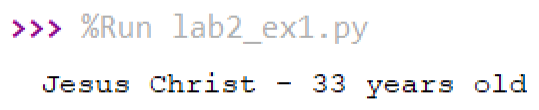
Make a new ".py" file for every exercise. Do so by clicking "New" in the left-hand upper corner or using ctrl+N.

Exercise 1: Variables

1.

Write a script with two variables, `name` and `age`, in which you save your name and age, the first as a string, the second as an integer.

Print this information, so you have e.g. this output:



```
>>> %Run lab2_ex1.py

  Jesus Christ - 33 years old
```

2.

Add an additional line so that `age` is incremented by one. Again, print the result.

Exercise 2: Area of the circle

Define a variable `pi` with pi to a certain amount of decimal numbers. Define a variable `radius`, equal to e.g. 10.

Define a variable `area`, which calculates the area of a circle, using the previous two variables. Print this variable to see the result.

Next, define a variable `circumference` which calculates the circumference of the circle. Print this variable as well.

Exercise 3: Ding!

Define a variable `floor`, which keeps track of which floor a lift is at. Try a negative number as well! Write code so that a bell is sound if the lift is at floor 55. We simulate this by printing `“DING!”` in the shell window. For values other than 55, nothing happens.

Exercise 4: User name

Define a variable `name` in which you put a name as a string. To use this as a user name on a website, it should be at least 4 characters long. Write code so that `name` is overwritten with twice the name concatenated if it is too short.[1] E.g. `“Ina”` becomes `“InaIna”`, `“Al”` becomes `“AlAl”`, but `“Stef”` remains unchanged.

Next, print the user name. Note: always print “`name`”, whether it was doubled or not! What would you have to change in the code if only doubled names are to be printed? Pressing one key suffices!

Tip: you can get the name of a string thus: `len(<fill in string>)`. Try this using e.g. `print(len(<your name>))`.

Tip: strings can be concatenated using `+`. Note that the Python interpreter knows how you want to use `+`: adding numbers vs. concatenating strings, depending on the type of the arguments.

Exercise 5: Area of the circle (bis)

We make a different version of exercise 2, so you can copy code from there.

We will make the first version more foolproof. Now a user can choose the radius, so nothing prevents him/her from choosing a negative radius, which doesn’t make sense. Alter the code: if the variable `radius` is positive, the area and circumference of the circle are calculated and printed, like before. If not, an error message is printed, e.g. `“error – radius negative”`.

Decide for yourself where you want the case where the radius is 0.[2] The difference is one character!

Exercise 6: Area of the circle (ter)

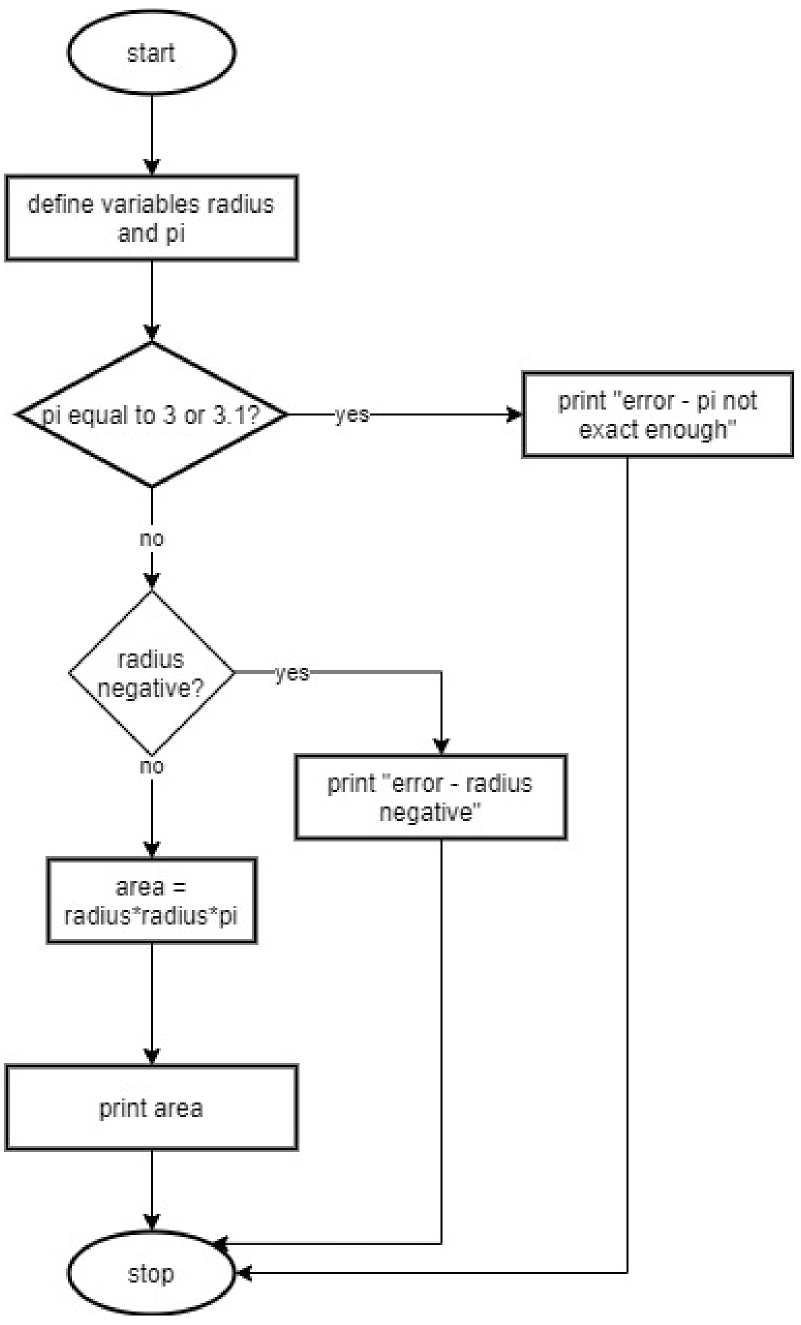
We make a third version. Add a first check to see whether `pi` is exact enough: values of 3 or 3.1 are not exact enough. If the variable `pi` is equal to one of these two, an error message is printed, e.g. `“error – pi not exact enough”`. If pi is exact enough, the previous distinction is made again: positive radius yields the area, negative radius an error message.

[1] We will assume a name always has at least two letters. This way, the doubled name is guaranteed to be at least 4 characters long.

[2] One could argue that a radius of 0 doesn’t make sense and should lead to an error message, or that this represents a point, i.e. a circle with area 0.

Note that our approach for checking the accuracy of pi is very naive: all values besides 3 and 3.1 are considered “accurate enough” – this includes completely unrelated (and even negative) numbers! Later in the semester you can try to devise a better algorithm for estimating the accuracy of pi, but for now our naive approach is sufficient.

Tip: use `elif`. Base your solution on the following flow chart.


start
define variables radius and pi
pi equal to 3 or 3.1?
yes
print "error - pi not exact enough"
no
radius negative?
yes
print "error - radius negative"
no
area = radius*radius*pi
print area
stop

Exercise 7: Even or odd

1.

Define a variable `number` in which you put a natural number (= positive integer). Write code to determine whether or not this number is even. If so, "even" is printed, if not, nothing happens. Tip: use the `%`-operator, `a % b` means: remainder after division of a by b.

2.

Add code so that "odd" is printed if the number is odd.

3.

In 1. you probably wrote something like this:

```
      if <a condition>:
            <do something>
```

Add code to the previous so that you save this condition in a variable. Print this variable.

4.

Let's make a final version. Now you don't get the number, but a boolean indicating whether the number was even, e.g. as a first line you have: `number_was_even = False`. Write code that again prints "even" or "odd", depending on the value of `number_was_even`.

Pro tip: do this *without* using `==` in your if-statement!

Exercise 8: Debugging

Download two files from Toledo: ***'lab2_ex8_debug1_en.py'*** and ***'lab2_ex8_debug2_en.py'***.

In the first file the code doesn't do what you expect: a 10-year-old yields an error as well. Correct the code.

The second file contains an error too: 6 is even *and* divisible by 3, so we want that both lines are printed in that case. Try again with 3, 4, 5, 12… Make sure all correct (and *only* the correct!) lines are printed.

Don't make it too complicated: both bugs can be solved by only altering a couple of characters!

Exercise 9: Lift

1.

We make a slightly more complex control structure, in multiple levels.

We want to program a lift for a building with 60 floors. Define a variable `floor` which saves the floor on which a lift is positioned.

Depending on the value of this variable, the following needs to happen:

- On floors < 0 an error message is printed: “`error - too low`”;
- On floors > 60 an error message is printed “`error - too high`”;
- On floors 0 and 60 “`DING!`” is printed;
- On floors that are multiples of 10 (but not on 0 and 60) the floor number is printed;
- On all other floors nothing happens.

Tip: drawing a flow chart helps for this assignment.

Tip: “multiple of 10” can be checked using the `%`-operator.

2.

Repeat the previous exercise for a building with 1000 floors. This is simple: All you need to do is copy-paste the code and adjust the maximum floor from 60 to 1000 in two places.

But what if there were conditions based on the maximum floor in ten places in your code? Or in a hundred different different places? In that case you would have to manually change all of these instances from 60 to 1000. A better approach is to make your code more generic, by saving the maximum floor in a variable (called `max_floor`, for example). You can then replace all instances of 60 in your code by this variable. Now, if your highest floor changes, you only need to change your code in one spot – the location where you defined the variable `max_floor` – no matter how many times this variable is used in your code.

Keep this principle in mind from now on. If you need to use the same information multiple times, it is almost always preferable to store this information in a variable, rather than copy-pasting it multiple times.

Exercise 10: Nested structures

1.

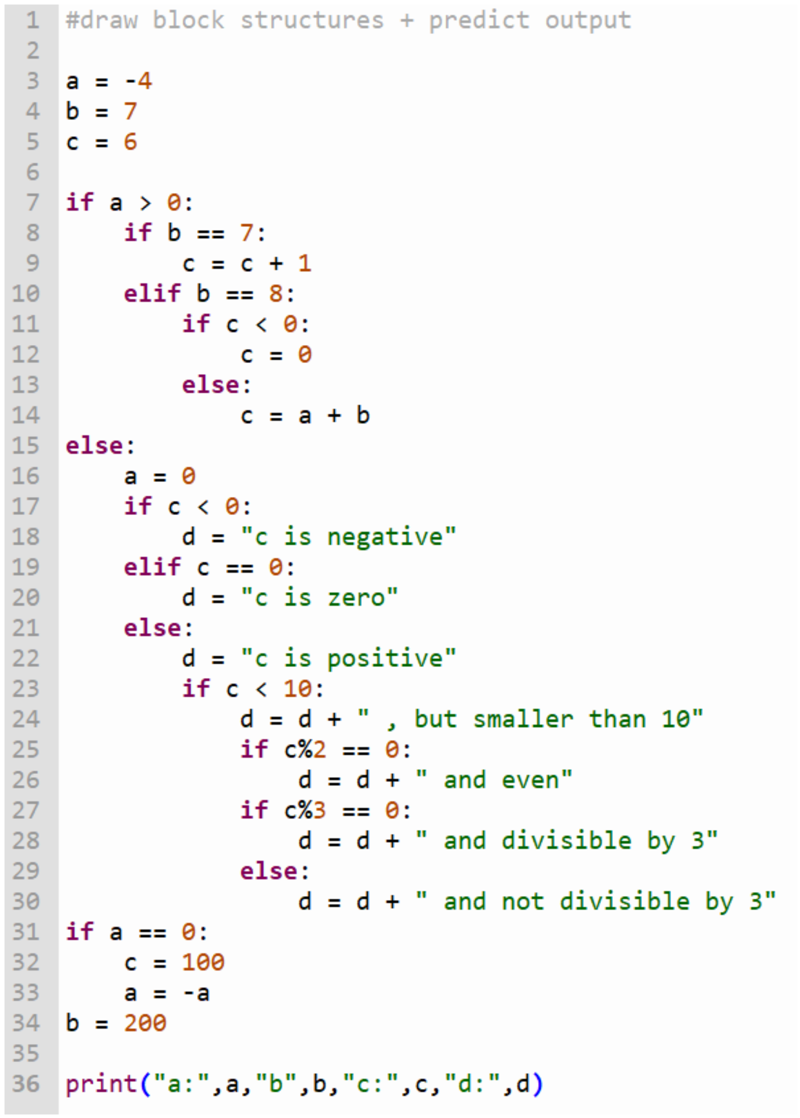
Below you see a piece of code which calculates the price of admission in a theme park, based on the visitor’s age, the day of the week and whether the park is presently open. Delineate the block structures in this code with frames, as explained in the lecture videos.

```python
park_open = True
day = "Thursday"
age = 22

if park_open:
    if age <= 6:
        print( "Free admission" )
    elif age >= 60:
        print( "Ticket price = €20" )
    else:
        if day == "Saturday" or day == "Sunday":
            print( "Ticket price = €45" )
        else:
            print( "Ticket price = €35" )
else:
    print( "Park is closed, come back some other time" )
```

2.

Time for a more complex example. The code below doesn't do anything useful, the important thing is to understand the control structure. Once again delineate the block structures with frames.



```python
#draw block structures + predict output

a = -4
b = 7
c = 6

if a > 0:
    if b == 7:
        c = c + 1
    elif b == 8:
        if c < 0:
            c = 0
        else:
            c = a + b
else:
    a = 0
    if c < 0:
        d = "c is negative"
    elif c == 0:
        d = "c is zero"
    else:
        d = "c is positive"
        if c < 10:
            d = d + " , but smaller than 10"
            if c%2 == 0:
                d = d + " and even"
            if c%3 == 0:
                d = d + " and divisible by 3"
            else:
                d = d + " and not divisible by 3"
if a == 0:
    c = 100
    a = -a
b = 200

print("a:",a,"b",b,"c:",c,"d:",d)
```

3.

Predict the output of the last line for given input for a, b and c. You can check your answer by downloading the file ***'lab2_ex10_start_en.py'*** from Toledo and running the code. Repeat for input:

- a = -4, b = 7, c = -10.
- a = -4, b = 7, c = 5.
- a = -4, b = 7, c = 6.
- a = 10, b = 7, c = 6. Remove d from the print, because this gives an error (why?).
- a = 10, b = 8, c = 5. Same, remove d.