Programming

Flow Control

Vassilis Markos, Mediterranean College

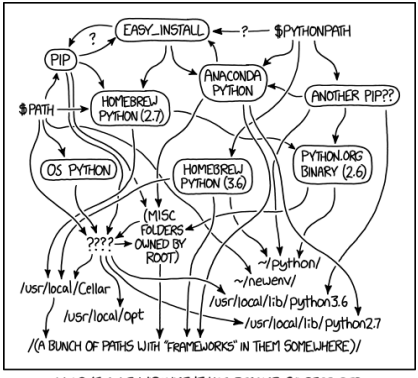
Week 02

Contents

1. Branching
2. Loops (First Try)
3. Fun Time!

Branching

Python Environment



Tidy up your Python environment regularly. Source: `https://xkcd.com/1987/`.

Even Or Odd?

How does the following program work?

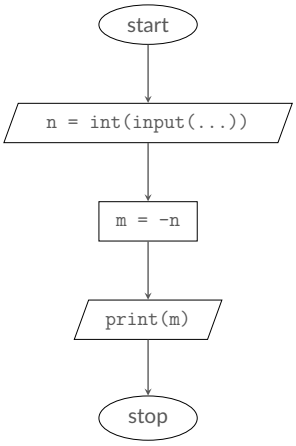
```
# source/flow_control_001.py

n = int(input("Please, enter an integer: "))
if n % 2 == 0:
    print("Even")
else:
    print("Odd")
```

Sequential Coding

- Consider the following program:

```
1 # source/sequential.py
2 n = int(input("n: "))
3 m = -n
4 print(m)
```

- In the above, the code is executed **sequentially**, i.e., one line after the other.

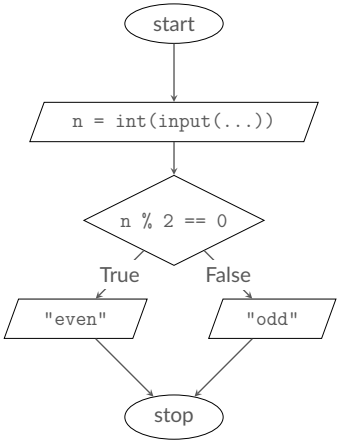


Branching

- Consider the following program:

```
1 # source/flow_control_001.py
2
3 n = int(input("Please, enter an
      integer: "))
4 if n % 2 == 0:
5     print("Even")
6 else:
7     print("Odd")
```

- Here we need to change path upon a condition. This is called **branching**.



Another Example Of Branching

What will the following program print?

```
# source/flow_control_002.py

grade = int(input("Please, enter your grade (0-100): "))
if grade >= 70:
    output = "Pass"
else:
    output = "Fail"
print(output)
```

Python And Indentation

In Python, whitespace matters!

- Whenever we enter a new code block we must indent our code accordingly.
- Usually, we use the `Tab` character to make sure all our code is properly indented.
- In case code is not properly indented, Python will raise an error (recall last lecture's lab).
- Most modern text editors take care of indentation and enter a `Tab` automatically when entering a code block.

Multiple Branches

What will the following print?

```
# source/flow_control_003.py

grade = int(input("Please, enter your grade (0-100): "))
if grade >= 70:
    output = "Excellent"
elif grade < 70 and grade >= 40:
    output = "Pass"
else:
    output = "Fail"
print(output)
```

Complex Branching In Python

Python offers a general purpose `if-elif-...-else` construct. The way this is interpreted is as follows:

1. First, the `if` condition is checked. If it is satisfied, the interpreter exits the block.
2. If the `if` condition fails, the interpreter proceeds with the first `elif` condition. If successful, it exits the block.
3. Step 2 is repeated until all `elif` statements have been checked and failed or one has been successful.
4. If none `if` / `elif` statement has been successful, the `else` case is executed (if there is one).

Condition Simplification

Will this work the same as the previous one?

```
# source/flow_control_003a.py

grade = int(input("Please, enter your grade (0-100): "))
if grade >= 70:
    output = "Excellent"
elif grade >= 40:
    output = "Pass"
else:
    output = "Fail"
print(output)
```

Condition Simplification

Will this work the same as the previous one?

```
# source/flow_control_003a.py

grade = int(input("Please, enter your grade (0-100): "))
if grade >= 70:
    output = "Excellent"
elif grade >= 40:
    output = "Pass"
else:
    output = "Fail"
print(output)
```

Why is that?

Condition Simplification

- Each `elif` / `else` statement is executed on condition that all previous ones have failed.
- This means we do not need to specify in subsequent `elif` statements that the above ones have failed, as the interpreter would not have reached that point in our code otherwise.
- Use this as a **good practice** to make your code easier to read / share / maintain!

Condition Simplification

Simplify the conditions of the following program as much as possible:

```
# source/flow_control_004.py

n = int(input("Please, enter an integer: "))
if n < 8 and n < 4:
    output = "Okay"
elif n >= 4 and n < 8:
    output = "Yeah"
elif n < 9 or n < 16:
    output = "So..."
else:
    output = "But..."
print(output)
```

Condition Simplification

Rewrite the following using only < as a comparison operator, i.e., no ==, <=, >= etc:

```
# source/flow_control_005.py
n = int(input("Please, enter an integer: "))
if n == 5:
    output = "Foo"
elif n <= 5:
    output = "Bar"
elif n > 7:
    output = "Foobar"
else:
    output = "Barfoo"
print(output)
```

Bye!

For which values of `n` will the following print `Bye!`?

```
1  # source/flow_control_006.py
2  n = int(input("Please, enter an integer: "))
3  if n % 5 == 0:
4      output = "Fly!"
5  elif n % 2 == 0:
6      output = "Fry!"
7  elif n % 3 == 0:
8      output = "Cry!"
9  else:
10     output = "Bye!"
11 print(output)
```

Fry!

For which values of `n` will the following print `Fry!`?

```
# source/flow_control_007.py
n = int(input("Please, enter an integer: "))
if n % 2 == 0:
    output = "Fly!"
elif n % 6 == 0:
    output = "Fry!"
elif n % 3 == 0:
    output = "Cry!"
else:
    output = "Bye!"
print(output)
```

What Is A Leap Year?

A leap year is a year which:

What Is A Leap Year?

A leap year is a year which:

- Is divisible by 4, but…

What Is A Leap Year?

A leap year is a year which:

- Is divisible by 4, but…
- is not divisible by 100, except for…

What Is A Leap Year?

A leap year is a year which:

- Is divisible by 4, but…
- is not divisible by 100, except for…
- when it is also divisible by 400.

What Is A Leap Year?

A leap year is a year which:

- Is divisible by 4, but…
- is not divisible by 100, except for…
- when it is also divisible by 400.

So, 2020 was a leap year, 2024 too, 2100 will not be, but 2000 was. Write a Python program that asks the user for a year and prints whether it is a leap year or not.

Leap Years

One possible solution is shown below:

```
# source/flow_control_008.py

year = int(input("Please, enter a year: "))
if year % 4 == 0 and year % 100 != 0 or year % 400 == 0:
    print("Leap year!")
else:
    print("Not leap year!")
```

Leap Years

One possible solution is shown below:

```
# source/flow_control_008.py

year = int(input("Please, enter a year: "))
if year % 4 == 0 and year % 100 != 0 or year % 400 == 0:
    print("Leap year!")
else:
    print("Not leap year!")
```

Can you use a shorter condition?

Leap Years

Another way to deal with this:

```
# source/flow_control_009.py

year = int(input("Please, enter a year: "))
if year % 400 == 0:
    print("Leap year!")
elif year % 100 == 0:
    print("Not leap year!")
elif year % 4 == 0:
    print("Leap year!")
else:
    print("Not leap year!")
```

Tips And Tricks

- When dealing with really complex conditions, as in the previous case, we can split them using a more length `if-elif-...-else` statement.
- This is not a general piece of advice: use with caution as it may result to really length code.
- However, in some cases, this makes code easier to read / share / maintain.
- Yet, in other cases we might end up with much redundant code, so...

Grades

Grades (0–100) correspond to A–F grades according to the following rule:

- A: 90-100
- B: 80-89
- C: 70-79
- D: 60-69
- F: 0-59

Write a Python program that asks the user for their grade in 0–100 and returns the corresponding A–F grade.

A Possible Solution

```
1  # source/flow_control_010.py
2
3  grade = int(input("Please, enter your grade: "))
4  if grade >= 90:
5      af_grade = "A"
6  elif grade >= 80:
7      af_grade = "B"
8  elif grade >= 70:
9      af_grade = "C"
10 elif grade >= 60:
11     af_grade = "D"
12 else:
13     af_grade = "F"
14 print(af_grade)
```

Taxes

In Pythonland, taxes are collected according to the following scheme:

- First $10,000\$$ are free of tax.
- The next $5,000\$$ are taxed with a 5% fee.
- The next $15,000\$$ are taxed with a 20% fee.
- Any remaining income is taxed with a 40% fee.

Write a Python program that asks the user for their income and computes their total tax fee.

A Possible Solution

```
# source/flow_control_011.py

income = float(input("Your income: "))
if income <= 10000:
    tax = 0
elif income <= 15000:
    tax = (income - 10000) * 0.05
elif income <= 30000:
    tax = 5000 * 0.05 + (income - 15000) * 0.20
else:
    tax = 5000 * 0.05 + 15000 * 0.20 + (income - 30000) * 0.40
print("Tax due:", tax)
```

Some Remarks

- Observe how in each case, we do not check whether the previous ones have failed, as we know that this is true (as discussed before).
- Also, since each portion of one's income is taxed differently, we have in each case to make use of the adequate formula for amounts that fall into one of the lower categories:
  - For instance, for an income of $14,000\$$, the first $10,000\$$ are free of tax, while the remaining $4,000\$$ are taxed with a 5% fee.
  - For instance, for an income of $24,000\$$, the first $10,000\$$ are free of tax, while the remaining $14,000\$$ are taxed as follows: the first $5,000\$$ with a 5% fee and the remaining $9,000\$$ with a 20% fee.

Loops (First Try)

Checking User Input

Assume that you want to write a program that decides whether a student has passed their exams. This could look like something we have seen above:

```
1 # source/flow_control_002.py
2
3 grade = int(input("Please, enter your grade (0-100): "))
4 if grade >= 70:
5     output = "Pass"
6 else:
7     output = "Fail"
8 print(output)
```

Checking User Input

This is really nice, but what if a user enters a number outside the range $[0, 100]$?

Checking User Input

This is really nice, but what if a user enters a number outside the range $[0, 100]$?

```
1  # source/flow_control_012.py
2
3  grade = int(input("Please, enter your grade (0-100): "))
4  if grade > 100 or grade < 0:
5      grade = int(input("Please, enter your grade (0-100): "))
6  if grade >= 70:
7      output = "Pass"
8  else:
9      output = "Fail"
10 print(output)
```

Users: The Source of All Bad

Run `../source/flow_control_012.py` and provide the following inputs:

- First, enter 115.
- Then, once prompted to enter your grade again, enter -56.
- What happened now?
- How can we prevent this, as well?

How About This?

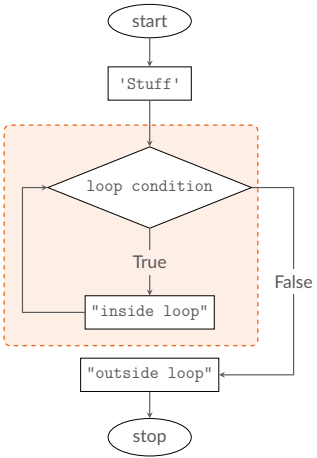
```
# source/flow_control_013.py

grade = int(input("Please, enter your grade (0-100): "))
if grade > 100 or grade < 0:
    grade = int(input("Please, enter your grade (0-100): "))
if grade > 100 or grade < 0:
    grade = int(input("Please, enter your grade (0-100): "))
if grade >= 70:
    output = "Pass"
else:
    output = "Fail"
print(output)
```

Well...

- While the chances that a user makes twice the same mistake are quite low, they are not null.
- This means that we should also take care of the case where a user enters wrong input three times.
- Even worse: four or even five times! Let alone six!
- How can we handle this?

Repeated Branching

- What we actually need is something like what is shown in the next flow chart, where we keep visiting our condition until it becomes `False`.
- In programming, this is called a **loop**.
- Loops are really necessary in almost any realistically useful program, so we have to master them!



A Simple Python Loop

So, the above program can be efficiently written as follows:

```
1  # source/flow_control_014.py
2
3  grade = int(input("Please, enter your grade (0-100): "))
4  while grade > 100 or grade < 0:
5      grade = int(input("Please, enter your grade (0-100): "))
6
7  if grade >= 70:
8      output = "Pass"
9  else:
10     output = "Fail"
11 print(output)
```

Python `while` Loop Syntax

In Python, a `while` loop adheres to the following syntax:

```
while <some logical condition>:
    <Stuff to be repeated while the condition holds>
```

- Each `while` loop starts with the **`while`** keyword followed by a logical condition, i.e., a Python expression evaluating to `True` / `False`.
- Then, we have to use a colon (`:`) to denote that the loop declaration line has ended.
- Below we write any code we want to repeat **properly indented.**

Important Notes

- It is important to somehow **change** variables appearing in **a loop's logical condition!** Why?

Important Notes

- It is important to somehow **change** variables appearing in **a loop's logical condition!** Why?
- Otherwise, the condition will never change from `True` to `False`, so we will end up with an **infinite loop.**

Important Notes

- It is important to somehow **change** variables appearing in **a loop's logical condition!** Why?
- Otherwise, the condition will never change from `True` to `False`, so we will end up with an **infinite loop.**
- Similarly, it is important to make sure the loop condition evaluates to `True` at some time, otherwise we will never enter the loop.

Important Notes

- It is important to somehow **change** variables appearing in **a loop's logical condition!** Why?
- Otherwise, the condition will never change from `True` to `False`, so we will end up with an **infinite loop.**
- Similarly, it is important to make sure the loop condition evaluates to `True` at some time, otherwise we will never enter the loop.
- As a consequence of the above, `while` loops might never be executed (bug or feature, depending on our purpose) on some occasions.

Booleanity In Python

What will the following print?

```
1 # source/flow_control_015.py
2
3 while 4:
4     print("Hi! How are you?")
5 print("Well...")
```

Booleanity In Python

What will the following print?

```
1 # source/flow_control_015.py
2
3 while 4:
4     print("Hi! How are you?")
5 print("Well...")
```

The above results to an infinite loop, since Python evaluates `4` to `True` (as any non–zero number). So this will keep printing `Hi! How are you?` all the time.
Press Ctrl + C to stop this.

Number Fun (?)

- What we get if we divide 1 by 2?

Number Fun (?)

- What we get if we divide 1 by 2?
  - 1/2, of course…

Number Fun (?)

- What we get if we divide 1 by 2?
  - 1/2, of course…
- What do we get if we divide 1 by 2 twice?

Number Fun (?)

- What we get if we divide 1 by 2?
  - 1/2, of course…
- What do we get if we divide 1 by 2 twice?
  - 1/4 (1/2 the first time and then 1/4).

Number Fun (?)

- What we get if we divide 1 by 2?
  - 1/2, of course…
- What do we get if we divide 1 by 2 twice?
  - 1/4 (1/2 the first time and then 1/4).
- Can we ever get 0 by dividing 1 by 2 finitely many times?

Number Fun (?)

- What we get if we divide 1 by 2?
  - 1/2, of course…
- What do we get if we divide 1 by 2 twice?
  - 1/4 (1/2 the first time and then 1/4).
- Can we ever get 0 by dividing 1 by 2 finitely many times?
  - Of course, no! The result will be of the form $1/2^n$ but never 0!

Number Fun (?)

Do you expect the following program to terminate?

```
# source/flow_control_016.py

n = 1
i = 0
while n > 0:
    n = n / 2
    i = i + 1
print(i)
```

Number Fun (?)

Do you expect the following program to terminate?

```
# source/flow_control_016.py

n = 1
i = 0
while n > 0:
    n = n / 2
    i = i + 1
print(i)
```

Then, why it does terminate?

Computer Arithmetic

- Computers cannot make calculations like humans, i.e., with “infinite” precision.
- On the contrary, due to their finite memory, they can only remember a **finite count of numbers** and with **limited precision** (so, rationals, and not even all of them).
- This means that very small numbers are just 0 in a computer’s memory.
- In our case, 2^{-1074} is one of the smallest non–zero numbers for the computer this was initially run on.
- So, again, beware of **numerical errors!**

Summing Numbers

Write a Python program that:

- asks the user to enter positive integers, and;
- computes and prints their sum.

Number entry should stop once the user enters a non–positive integer (i.e., negative or zero).

A Possible Solution

```
# source/flow_control_017.py

s = 0 # We initialise the target sum at 0.
n = int(input("Enter an integer: "))
while n > 0:
    s = s + n
    n = int(input("Enter an integer: "))
print("The sum is:", s)
```

Averaging Numbers

Write a Python program that:

- asks the user to enter positive integers, and;
- computes and prints their average.

Number entry should stop once the user enters a non–positive integer (i.e., negative or zero).

A Possible Solution

```
# source/flow_control_018.py

s = 0 # We initialise the target sum at 0.
c = 0 # The count of user-provided numbers
n = int(input("Enter an integer: "))
while n > 0:
    s = s + n
    c = c + 1
    n = int(input("Enter an integer: "))
avg = s / c
print("The average is:", avg)
```

What Is Going Wrong Here?

Alice tried to write a program that computes the sum of all even numbers provided by a user (number insertion terminated by inserting `0`). Does it work as intended?

```
1 # source/flow_control_019.py
2
3 n = int(input("Enter an integer: "))
4 s = 0
5 while n != 0:
6     n = int(input("Enter an integer: "))
7     if n % 2 == 0:
8         s = s + n
9 print("Sum of even numbers: ", s)
```

Order Matters!

If you did not spot the error yet, try the following sequences of inputs:

- `1`, `2`, `3`, `4`, `0`, which should print `6`.
- `2`, `3`, `4`, `5`, `0`, which should print `6`.

In the second case, the first user input, `2`, is ignored since when entering the `while` loop we immediately ask for a new number.
How can we fix this?

Order Matters!

Below you can see a fixed version of Alice's program:

```
# source/flow_control_020.py

n = int(input("Enter an integer: "))
s = 0
while n != 0:
    if n % 2 == 0:
        s = s + n
    n = int(input("Enter an integer: "))
print("Sum of even numbers: ", s)
```

The idea is to ask for new user input last so as to make sure that the first (outside the loop) input is not ignored!

Fun Time!

In–class Exercise #001

A pyflix subscription (fictional python tutorials streaming service) has the following pricing scheme:

- The first 5 tutorials are free.
- The next 10 tutorials cost 5$ each.
- The next 20 tutorials cost 4$ each.
- Any further tutorials cost 2.5$ each.

Write a Python program that asks the user the number of tutorials they want to attend and computes the corresponding total cost.

In-class Exercise #002

Rewrite the following without any `elif`, i.e., only using `if-else`:

```
# source/exercise_002.py

n = int(input("Please, enter an integer: "))
if n > 10:
    print("What?")
elif n > 15:
    print("Why?")
elif n > 20:
    print("How?")
else:
    print("Well...")
```

In–class Exercise #003

A quadratic equation, $ax^2 + bx + c = 0$, $a \neq 0$, is solved using the following formula:

$$x = \frac{-b \pm \sqrt{\Delta}}{2a}, \quad \Delta = b^2 - 4ac.$$

if $\Delta > 0$. If $\Delta = 0$ then the equation has a single solution, $x = -\frac{b}{2a}$, while if $\Delta < 0$ the equation has no (real) roots.
Write a Python program that asks the user to provide the three coefficients of a quadratic equation, a, b, c, and prints its solution(s), if any, or an appropriate message if no solutions exist.

In–class Exercise #004

Assuming that you are allowed to use only division by (and modulo of) 2 and 3, write a python program that:

- Asks the user for an integer number, n.
- Prints on screen an appropriate message for all the following cases:
  - Whether n is a multiple of 2.
  - Whether n is a multiple of 3.
  - Whether n is a multiple of 6.
  - Whether n is a multiple of 24.
- Explain your rationale by providing appropriate comments in your code.

In–class Exercise #005

A number n divides a number m if `m % n == 0`. Write a Python program that asks the user for a positive integer n and prints on screen all of its divisors.
Your program **should check that the user indeed provides a positive integer**, i.e., it should ask the user to provide another number in case they provided a non–positive integer (ignore cases where user input might not be integer).

In–class Exercise #006

- We say that n is a proper divisor of m if n is a divisor of m and $n \neq m$. A positive integer is said to be **perfect** if it is equal to the sum of its proper divisors.
- Write a Python program that asks the user for a positive integer and checks if its is perfect or not, printing a relevant message on screen.
- Your program should check that the user indeed provides a positive integer as in the previous exercise.

In–class Exercise #007

- Two numbers are said to be **amicable** if each one is equal the sum of the proper divisors of the other.
- Write a Python program that asks the user for two positive integers and checks if they are amicable or not, printing a relevant message on screen.
- Your program should check that the user indeed provides a positive integer as in the previous exercise.

Homework 1

Before loops and ifs and all that stuff, most programming languages offered a single statement that allowed programmers to control the flow of a program: `GOTO`. Write a 300–400 words essay regarding:

- the use of `GOTO` in computer programming;
- why it was abandoned;
- which languages still offer it and when is its use suggested.

Submit your work at: `v.markos@mc-class.gr`.

Homework 2

- In this week's materials, under the `homework` directory, you can find some Python programming Tasks. Complete as many of them as you can (preferably all).
- This is important, since tasks such as those provided with this lecture's materials will most probably be part of your course assessment portfolio. So, take care to solve as many of those tasks as possible!
- Share your work at: `v.markos@mc-class.gr`

Any Questions?

Do not forget to fill in the questionnaire shown right!

https://forms.gle/dKSrmE1VRVWqxBGZA