bianca.phelan@tudublin.ie

O Object
O Oriented
P Programming

S1 = Python with Bianca

S2 = Java with Bryan

DT228(TU856)/DT282(TU858) - 2

OLLSCOIL TEICNEOLAÍOCHTA BHAILE ÁTHA CLIATH
TUDUBLIN
TECHNOLOGICAL UNIVERSITY DUBLIN

COMPUTER SCIENCE

Python
Flow Control

Objectives

- Learn how to control the flow of your programme
- Discuss the most common control flow elements in Python

Revision

- machine code and assembly are low level

  ➔ represents what the hardware is capable of

- other languages are high level

  ➔ cannot be directly executed, need transformation

  - can be compiled: persistent transformation into machine code
  - or interpreted: need interpreter at runtime!

Statements in Python

- Expressions
- Assignments
- Print

[4]

Python Interactive Mode

```
Python 3.7.4 (default, Aug 13 2019, 15:17:50)
[Clang 4.0.1 (tags/RELEASE_401/final)] :: Anaconda, Inc. on darwin
Type "help", "copyright", "credits" or "license" for more informat
>>> 8*4
32
>>> a = 17+4
>>> a
21
>>> a - 2
19
>>>
```

Expressions in interactive mode are evaluated immediately, result is shown.

Variable assignment

Variable used in expression

Running a Python Programme - From Command Line

contents of file expr.py

```
(base) mimimi:Code andreas$ cat expr.py
8*4
a = 17+4
print(a)
(base) mimimi:Code andreas$ python3 expr.py
21
(base) mimimi:Code andreas$
```

Running a Python Programme - From Command Line Cont'd

expression evaluated, but result not shown!

```
(base) mimimi:Code andreas$ cat expr.py
8*4
a = 17+4
print(a)
(base) mimimi:Code andreas$ python expr.py
21
(base) mimimi:Code andreas$
```

using built-in print function

program output

Interactivity

- In the lab you have seen two of the most basic interactivity elements:
- `input()` for requesting input from a user
- `print()` for printing something to standard output

```
Python Console
>>> input("name:")
name:>? hello
'hello'
>>> myname = input("Please enter your name:")
Please enter your name:>? Bianca
>>> print("The name is "+ myname)
The name is Bianca
```

Special Variables
myname = {str} 'Bianca'

This returns a string variable always!

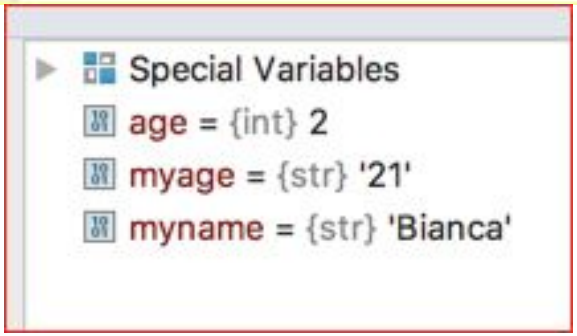
Interactivity cont'd

```
>>> myage = input("the age: ")
the age: >? 12
>>> print("my age is "+myage)
my age is 12
>>> type(myage)
<class 'str'>
```

```
>>> type(my_newage)
<class 'int'>
>>> print("my new age is: "+my_newage)
Traceback (most recent call last):
  File "<input>", line 1, in <module>
TypeError: can only concatenate str (not "int") to str
```

Interactivity and String Formatting

```
>>> print ("My name is %s and I am %s years old."%(myname,myage))
My name is Bianca and I am 21 years old.
>>> print ("My name is %s and I am %d years old."%(myname,myage))
Traceback (most recent call last):
  File "<input>", line 1, in <module>
TypeError: %d format: a number is required, not str
```




```
>>> print ("My name is %s and my dog is %d years old."%(myname,age))
My name is Bianca and my dog is 2 years old.
```

Type of an Input Return

```
>>> myage = input("Enter age:")
Enter age:>? 21
>>> print("The age is "+myage)
The age is 21
>>> myage.isnumeric()
True
>>> myname.isnumeric()
False
>>> myname.isalpha()
True
>>> age = 2
>>> type(myage)
<class 'str'>
>>> type(age)
<class 'int'>
```

Special Variables
age = {int} 2
myage = {str} '21'
myname = {str} 'Bianca'

Only works since Python3, not in earlier versions

```
first_name = 'Bianca'
last_name = 'Phelan'

message = f'Hello, {first_name} {last_name}'
print(message)
```

This version is very similar to other languages, such as Java Script and C#.

```
"/Volumes/GoogleDrive/My
Hello, Bianca Phelan
```

Raw strings with r flag

- Tired of escaping characters?
- Use a raw string!

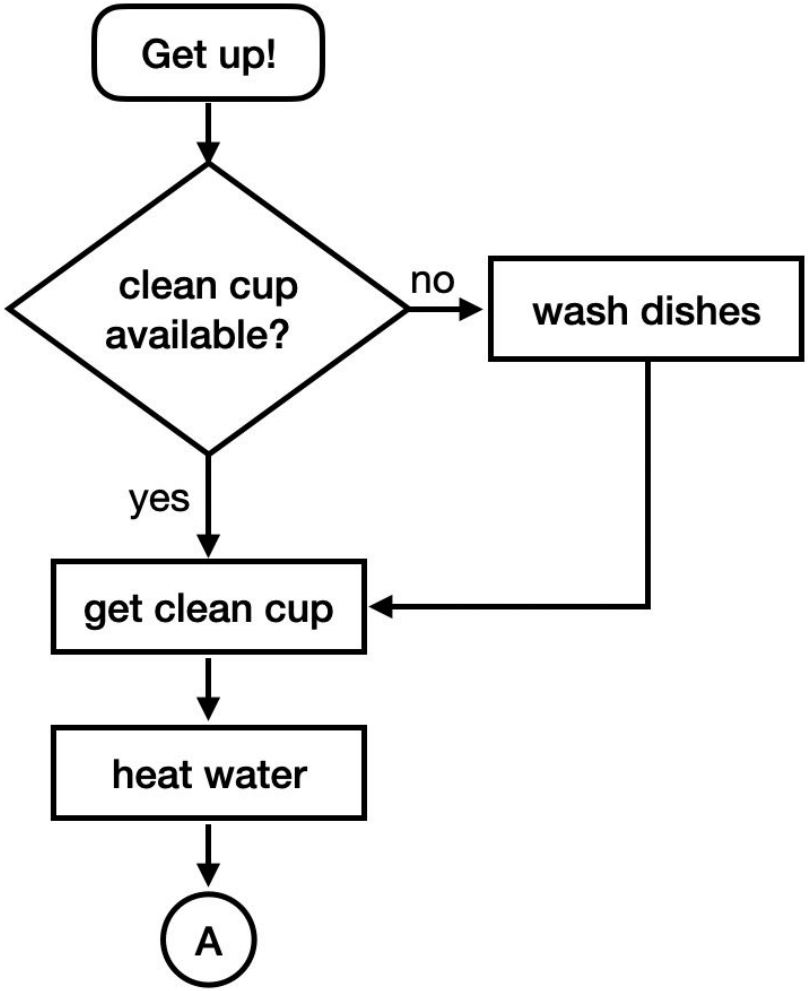
```
>>> print(r'Hello\t World')
Hello\t World
```

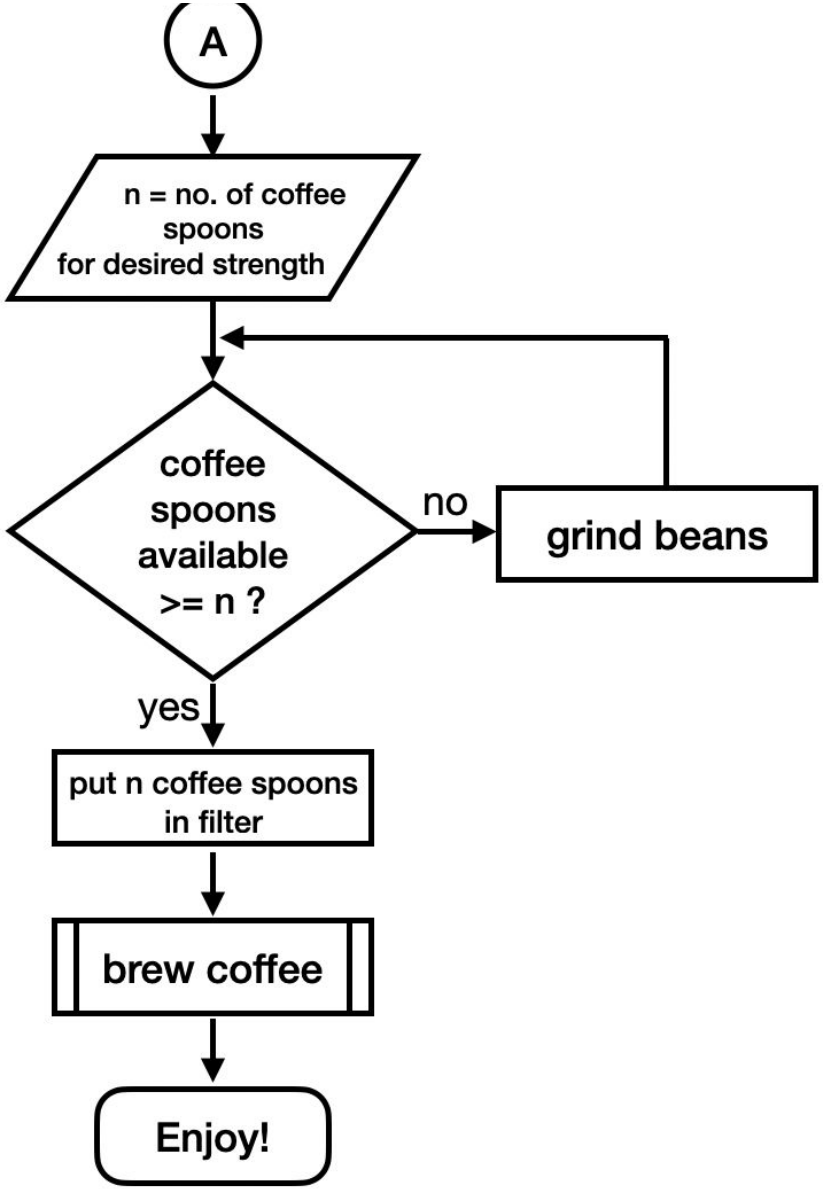
A raw string treats a backslash as a literal character!

Backslashes followed by quotes are tricky! **The backslash is escaped but also remains in the result!!**

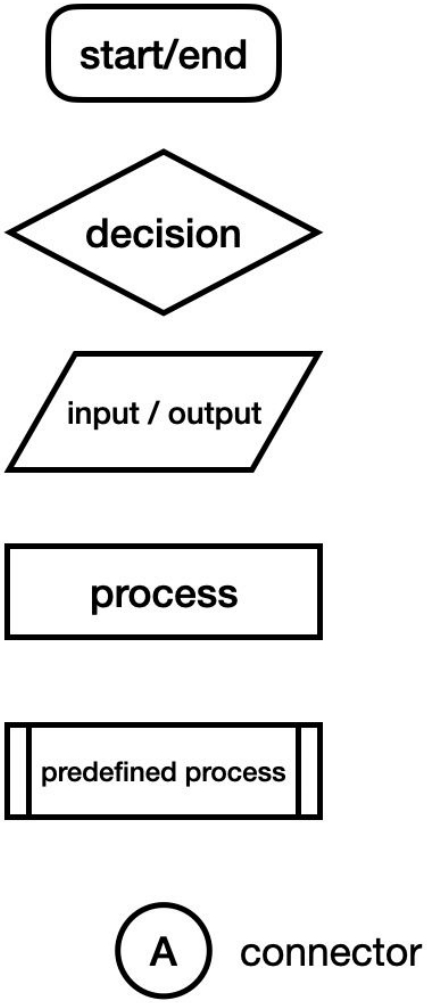
Choices and Decisions

Making Coffee in the Morning - How Is It Done? (Programmer's Edition)





Flow Chart Building Blocks



Flow Charts

Pro

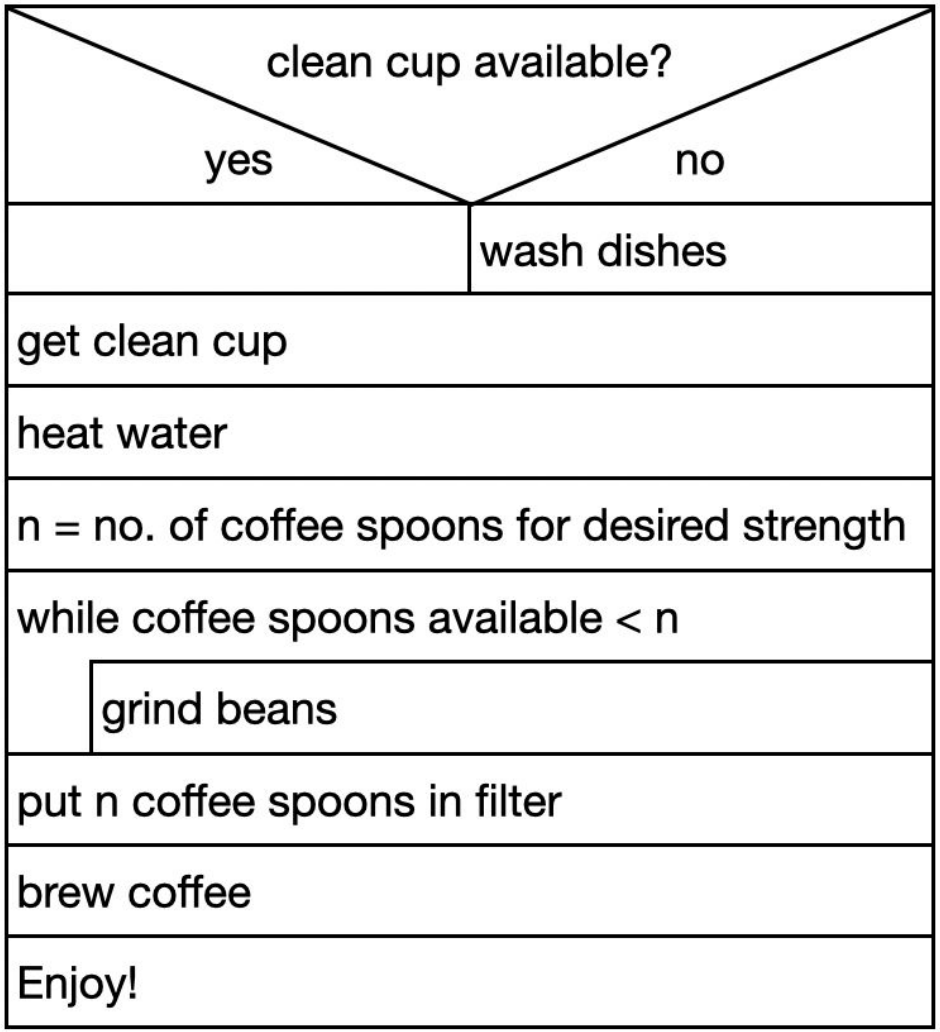
- Standardised internationally ISO 5807, DIN 66001
- Easy to understand

Con

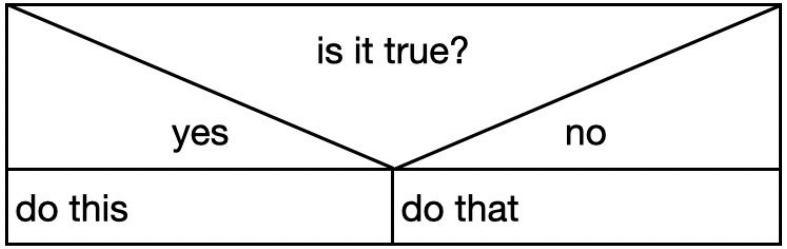
- Diagram becomes cluttered easily
- No difference between a branch and a loop

An alternative are
Nassi-Shneidermann-Structograms

Making Coffee In The Morning - How Is It Done?

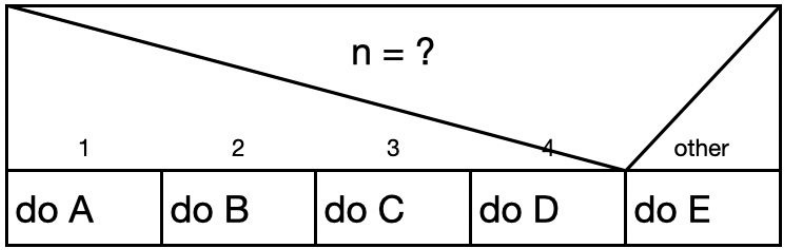


Nassi-Shneidermann Building Blocks

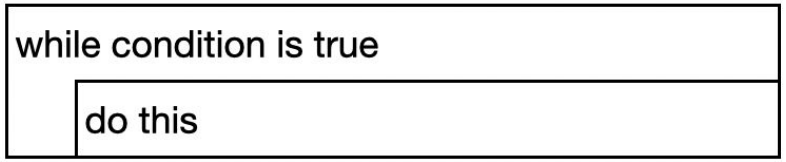


decision / branch

branches and loops can be nested!

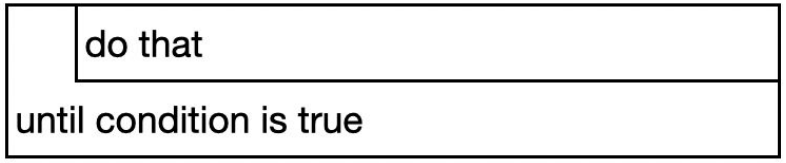


branch with multiple options



"test first"-loop

← possibly never executed, if condition already met



← always executed at least once

"test last"-loop

Nassi-Shneidermann

Pro

- Standardised DIN 66261 (Germany)
- Less prone to clutter
- Can differentiate between branches and loops
- Forces structure

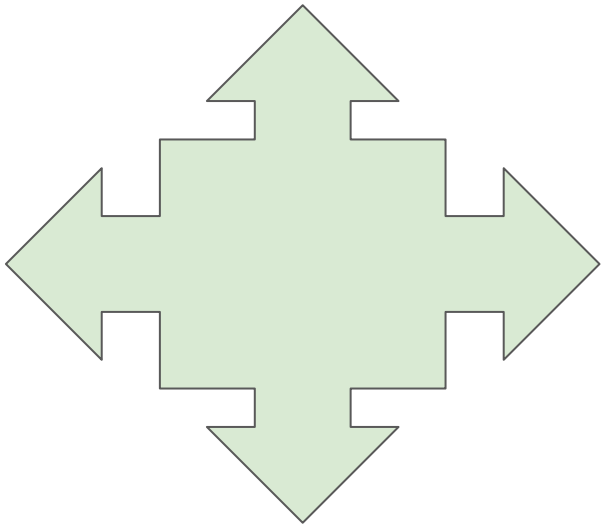
Con

- Less intuitive to read

Making your Python Programs Sophisticated

Use choices and decisions

- If… else
- For loop
- While loop
- Try, except



Conditional Control Flow

- Control flows rely on you evaluating a condition
- Most common condition is the comparison
  - Example: `x == y`, means you are asking if `x` and `y` have the same value, if they are, then the condition is met and the statement evaluates as `TRUE`, otherwise it evaluates as `FALSE`

Control Flow - Branches

expression that evaluates as True or False

```
if clean_cup_available == "y":
    print("That's great!")
else:
    print("Go wash dishes!")
```

executed if True

executed if False

Indentation is part of the Python syntax!

Comparisons

Sign	Meaning	Example, evaluating TRUE
==	Equals	4 == 4
!=	Not Equals	5 != 10
<	Smaller than	4 < 10
>	Greater than	10 > 4
<=	Smaller than or equals to	5 <= 5; 5 <= 10
>=	Greater than or equals to	10 >= 10; 10 >= 1

Logical Operators

Operator	Example, evaluates TRUE
AND	5 == 5 AND 2 > 1
OR	5 == 5 OR 3 < 1
NOT	NOT 2 > 6

- OR: if any statement on either side evaluates TRUE, then the result is TRUE
- AND: both conditions must be TRUE in order for the result to evaluate to TRUE

If Statement

- Probably the most common control flow statement
- Allows the programme to check for a certain condition and perform the appropriate action if the condition is either met or not met
- In Python:

```
if condition 1 is met:
    do A
elif condition 2 is met:
    do B
else:
    do C
```

- No brackets needed, many other languages like Java and C need brackets with a statement like this
- Python instead uses indentation. Everything that is indented will be treated as one block. Everything in a block will be executed if a condition is TRUE.

If Example

```
letters = input("Enter a or b")

if letters == 'a':
    print("An a has been inputted.")
    print("How are you?")
elif letters == 'b':
    print("A b has been inputted.")
    print("Today is awsome weather.")
else:
    print("Wrong input")
    print("An apple a day keeps the doctor away.")
```

Inline If

- Convenient for simple tasks

```
myinput = input("Enter either A or B:")
print("This is a good day" if myinput == 'A' else "Tomorrow is a good day")
```

- Does not support elif
- Needs an else
- Can add `..., end = "always print")` to print something regardless

Control Flow - Loops

```
while coffee_available < desired_strength:
    print("Grind coffee!")
    coffee_available += 1
```

executed repeatedly as long as the condition is met

Syntax similar to *if*

For Loops

- Executes a block of code repeatedly until the condition for execution is no longer valid
- Example: looping through an iterable

```
pets = ["Horse", "Dog", "Cat", "Hamster"]
for mypets in pets:
    print(mypets)
```

An “iterable” is everything that can be looped over, for example strings, tuples, lists.

Iterate with index enumerate

```
pets = ["Horse", "Dog", "Cat", "Hamster"]
for index, mypets in enumerate(pets):
    print(index, mypets)
```

```
0 Horse
1 Dog
2 Cat
3 Hamster
```

Default in enumerate() is 0, but can be set to a different start index position.

Loop through a String

```
message = "Hello World"
for i in message:
    print(i)
```

```
H
e
l
l
o

W
o
r
l
d
```

Loop through a sequence of numbers using range()

- `range(start, step, end)`
  - `start` optional integer, default is 0
  - `end` mandatory integer, defines where to stop
  - `step` optional integer, default is 1

```
for i in range(4):
    print(i)
```

```
0
1
2
3
```

While loop

It's terribly easy to create endless loops using while. **Remember to force it to stop at some stage**!

- A certain action is performed as long as a certain condition is met
- Structure:

```
While conditionA is true:
    Do A
```

Realistic use cases of while loops include an automatic increase of the condition. For example, when reading files. If it needs to be forced, use a for loop instead.

```
counter = 4
while counter >0:
    print("Counter: ", counter)
    counter -=1
```

```
Counter:  4
Counter:  3
Counter:  2
Counter:  1
```

Break

- Exit the entire loop if a certain condition is met
- Can be tricky if the program gets very complicated
- Example to end a loop prematurely:

```
i = 0
for b in range(12):
    i += 3
    print("i: ", i, " and b: ", b)
    if i == 6:
        break
```

```
i:  3  and b:  0
i:  6  and b:  1
```

Continue

- Like break can be used to manipulate loops
- Skips a certain iteration

```
i = 0
for b in range(12):
    i += 3
    print("i: ", i, " and b: ", b)
    if i == 6:
        continue
    print("I'll skip i=6")
```

```
i:  3  and b:  0
I'll skip i=6
i:  6  and b:  1
i:  9  and b:  2
I'll skip i=6
i:  12  and b:  3
I'll skip i=6
i:  15  and b:  4
I'll skip i=6
i:  18  and b:  5
I'll skip i=6
i:  21  and b:  6
I'll skip i=6
i:  24  and b:  7
I'll skip i=6
i:  27  and b:  8
I'll skip i=6
i:  30  and b:  9
I'll skip i=6
i:  33  and b:  10
I'll skip i=6
i:  36  and b:  11
I'll skip i=6
```

Iterators and enumerate()

- Iterators are objects that contain a **countable** number of values
- Python provides a built-in function called `enumerate()` to iterate over countable objects and gives you their index position
- Objects that support iteration: dictionaries, lists, strings and many more

enumerate() is useful when you need the actual index location of the iterable.

Non-sequenced objects, such as sets rely on enumerate().

Example of enumerate()

```
fruits = ("apple", "banana", "pear")
for index, fruit in enumerate(fruits):
    print("index is %d and value is %s " % (index, fruit))
```

```
index is 0 and value is apple
index is 1 and value is banana
index is 2 and value is pear

Process finished with exit code 0
```

Example without enumerate() means that you have to keep track **manually** of your iterating counter variable, here called “i”.

```
fruits = ("apple", "banana", "pear")
i = 0

for fruit in fruits:
    print("index is %d and value is %s " % (i, fruit))
    i += 1
```

Try, Except

- How to manage errors in your program
- Structure:

```
Try do something:
Except: do something else in case of an error
```

```
try:
    result = 4 / 0
    print(result)
except:
    print('An error occurred.')
```

```
try:
    result = 4 / 0
    print(result)
except ZeroDivisionError:
    print('cannot divide by zero.')
except Exception as e:
    print("Unknown error.")
```

Build in errors

- ValueError
- ZeroDivisionError
- IOError
- ImportError
- IndexError
- KeyError
- NameError
- TypeError

Check Python documentation for more!

https://docs.python.org/3/library/exceptions.html

exception **Exception** ¶

All built-in, non-system-exiting exceptions are derived from this class. All user-defined exceptions should also be derived from this class.

exception **ArithmeticError**

The base class for those built-in exceptions that are raised for various arithmetic errors: `OverflowError`, `ZeroDivisionError`, `FloatingPointError`.

exception **BufferError**

Raised when a buffer related operation cannot be performed.

exception **LookupError**

The base class for the exceptions that are raised when a key or index used on a mapping or sequence is invalid: `IndexError`, `KeyError`. This can be raised directly by `codecs.lookup()`.

Syntax, Runtime,

```
if message == first_name
    print("oh no")
```

```
"/Volumes/GoogleDrive/My Drive/Vor
  File "/Volumes/GoogleDrive/My Dr
    if message == first_name
                              ^
SyntaxError: invalid syntax

Process finished with exit code 1
```

Syntax errors are the easy ones. Here a colon is missing.

```
a = 45
b = 0

print(a/b)
```

```
"/Volumes/GoogleDrive/My Drive/Vorlesu
Traceback (most recent call last):
  File "/Volumes/GoogleDrive/My Drive/
    print(a/b)
ZeroDivisionError: division by zero

Process finished with exit code 1
```

Runtime errors are the next best ones. Here a division by 0. Start fixing by going to the line pointed out, then move up.

```
a = 45
b = 4

if a < b:
    print(a, ' is greater than ', b)
```

Logical errors are the hardest to track down.

Logical Errors

- Unit testing is very useful in this context
  - We will talk about testing in teaching week 9
- Look at the `traceback`
- Re-read your code and look at the documentation
- Search online
- Ask for help
- Take a break

Summary

- ★ **Control flow using if**
- ★ **For loop and range()**
- ★ **While loop**
- ★ **try/except**
- ★ **Error codes in Python**

References

1. Learn Python in one day, Jamie Chan, 2014.
2. Lecture Notes, A. Hess, Hochschule Furtwangen, 2020.
3. Expressions in Python, https://runestone.academy/runestone/books/published/thinkcspy/SimplePythonData/StatementsandExpressions.html#:~:text=An%20expression%20is%20a%20combination,expression%20and%20displays%20the%20result., accessed Oct 2020