Lecture 3: Conditionals, recursions, fruitful functions

Morten Rieger Hannemose, Vedrana Andersen Dahl

Fall 2023

Today's lecture

1. Course info (ca. 10 min)
2. Conditionals, functions (ca. 20 min)
3. Recursion, input (ca. 20 min)

Course info

1. New deadline for Project 1: 27 September
2. Submission format: token files or Python files
   - `normal_weight.py`
   - `survival_temperature.py`
   - `unit_conversion.py`
   - `hadlock.py`

Conditional execution

Two examples

```
a =
if a > 23:
    print('I know that a is greater than 23')
print('This line runs no matter what')
```

```
a =
if a > 23:
    print('I know that a is greater than 23')
else:
    print('I know that a is not greater than 23')
```

- Branching the flow of execution
- Syntax:
  - Keyword `if`
  - Conditions: something that needs to be evaluated as either `True` or `False`
  - Indented body
  - Optional: alternative execution (`else`), chained execution (`elif`)

Coding example

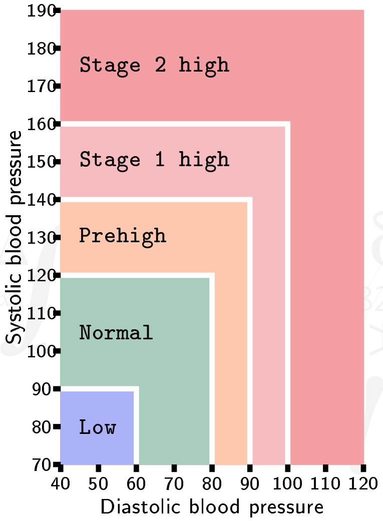
`blood_pressure.py`, exam from June 2021.

Blood pressure

Measurement of blood pressure consists of two values: systolic blood pressure (a higher value) and diastolic blood pressure (a lower value). Based on these two values, blood pressure can be categorized as indicated in the chart on the right.

Problem formulation

Create a function `blood_pressure` which takes a number s with systolic blood pressure and a number d with diastolic blood pressure as input. The function should return a string with the blood pressure category, written exactly as in the chart above. If a measure is between categories, a category for a higher value should be returned.



Conditionals

Common pitfalls

- Forgetting that condition is an expression to be evaluated (seeing `if-statement` as an entity)
- Using the wrong logical operators: mixing `and` and `or`
- Using complicated chained execution that does not cover all options, yet believing that all options are exhausted
- Setting up an extra condition, instead of using `else`
- Using `==True` to check whether a condition is true

Fruitful functions

- Returning a value – we can then use from the main program (or wherever we call the function)
- New keyword `return`
- Coding example: `full_price.py` equipped with handling of negative values

Fruitful functions

Common pitfalls

- Not understanding the difference between returning a value and printing a value. (When testing, we usually just print the returned value, making it difficult to understand the difference.)
- Returning prematurely (dead code)

Recursion

- Recursive function: A function which calls itself
- Important in algorithmics and numerical computation
- Example: Recursively copy the content of a folder and subfolders
- Coding example: `factors_of_2.py`

Input from user

- You can have a conversation with your computer!
- Coding example: `unhappy.py`

Code used for coding examples

Conditionals

```python
def blood_pressure(s, d):
    if s<90 and d<60:
        bp = 'Low'
    elif s<120 and d<80:
        bp = 'Normal'
    elif s<140 and d<90:
        bp = 'Prehigh'
    elif s<160 and d<100:
        bp = 'Stage 1 high'
    else:
        bp = 'Stage 2 high'
    return bp
```

Functions

```python
def full_price(price):
    if price <= 0:
        print('price should be positive')
        return 0
    rate = 0.2
    tip = rate * price
    total = price + tip
    return total

print(full_price(100))
print(full_price(-10))
```

Code used for coding examples

Recursion

```python
def factors_of_2(n):
    if n%2 == 0:
        print('2 * ', end='')
        factors_of_2(n//2)
    else:
        print(n)
```

Input

```python
def unhappy():
    print('I am unhappy.')
    a = input('Write "yes" to make me
     happpy: ')
    if a == 'yes':
        print('I am happy!')
    else:
        unhappy()
```