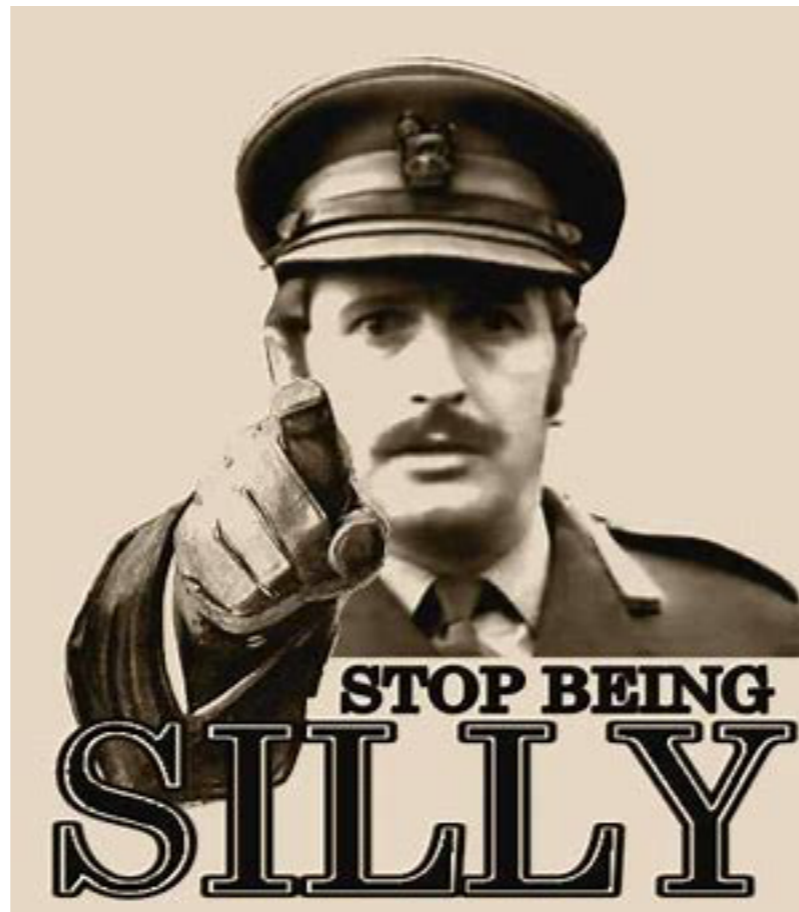


# Python

## String Formatting





# Formatting Strings

- We really need to learn how to format strings
  - Python has made several attempts before settling on an efficient syntax.
    - You can find information on the previous solutions on the net.
  - Use the `format` function
    - Distinguish between the **blueprint**
    - and the **string to be formatted**
    - Result is the formatted string.



# Formatting Strings

- Blueprint string
  - Uses { } to denote places for variables
  - Simple example

• `"{} {}".format('one', 'two')`

Blueprint

Calling  
format

String to be  
formatted

• Result `'one two'`



# Formatting Strings

- Inside the brackets, we can put indices to select variables
  - 0 means first variable, 1 second, ...
  - Can reuse variables

```
>>> "{0}, {0}, {1}, just {0}".format("great", "extraordinary")
'great, great, extraordinary, just great'
```



# Formatting Strings

- Additional formatting inside the bracket after a colon
- Can assign the number of characters to print out

```
>>> "{0:10}, {1:10}, {0:10}".format("funny", "nuts")  
'funny      , nuts      , funny      '
```

- Default alignment is to the left



# Formatting Strings

- Use ^ to center
- Use < to left-align
- Use > to right-align

```
>>> "{0:10}|{1:^10}|{0:>10}".format("sheep", "wolf")  
'sheep      |      wolf      |      sheep'
```



# Formatting Strings

- Numbers are handled without specifying format instructions.

```
>>> "{} divided by {} is {} modulo {}".format(143, 29, 143//29, 143%29)
'143 divided by 29 is 4 modulo 27'
```

- Or we can insist on special types
  - Use s for string
  - Use d for decimal
  - Use f for floating point
  - Use e for floating point in exponential notation



# Formatting Strings

- By specifying “f” we ask for floating point format
- By specifying “e” we ask for scientific format

```
>>> "{0:f}, {0:e}".format(3.141)
'3.141000, 3.141000e+00'
```





# Formatting Strings

- Padding
  - If the variable needs more space to print out, it will be provided automatically

```
>>> "{:10s}".format("Pneumonoultramicroscopicsilicovolcanoconiosis")
'Pneumonoultramicroscopicsilicovolcanoconiosis'
```

- This is actually the longest officially recognized word in English



# Formatting Strings

- Padding:
  - On the reverse, we can give the number of significant digits after a period

```
>>> "{:8.2f}".format(3.141592653589793238462643383279502884197169399375105  
82097494459230781640628620899862803482534211706798214808651328230664709384  
4609550582231725359408128481)  
'      3.14'
```

- We only want to keep two decimal digits after the period
- But use a total of 8 spaces for the number.



# Formatting Strings

- Escaping curly brackets:
  - If we want to write strings with format containing the curly brackets “{“ and “}”, we just have to write “{{“ and “}}”

```
>>> "{{ { }, {} }}".format(3, 4)
' { 3, 4 }'
```

- A single bracket is a placeholder, a double curly bracket is a single one in the resulting string.



# Application: Pretty Printing

- Develop a mortgage payment plan
  - Accountants have formulae for that, but it is fun to do it directly
  - Assume you take out a loan of  $L$  dollars
    - The loan is financed at a rate of  $r\%$  annually
    - Interest is paid monthly, i.e. at a rate of  $r/12\%$
  - Each month you make a repayment
    - Part of the repayment is to pay the interest
    - The remainder pays down the debt



# Mortgage Payments

- Use a while-loop
  - Condition is that there is still an outstanding debt
  - Adjust outstanding debt
  - Count the number of payments
- Need to initialize values



# Mortgage Payments

- We need values for:
  - Monthly Rate (interest in percent)/1200
  - Principal
  - Repayment
- Get those from the user
  - A true application would contain code that checks whether these numbers make sense.





# Mortgage Payments

- Initialization

```
princ = float(input("What is the prinipal "))
rate = float(input("What is the interest rate (in percents)? ")) / 1200
print("Your minimum rate is ", rate*princ)
paym = float(input("What is the monthly payment? "))
month = 0
```



# Mortgage Payments

- We continue until we paid down the principal to zero

```
while princ > 0:
```





# Mortgage Payments

- Update the situation in the while loop
- Last payment does not need to be full, so we calculate it

```
intpaid = princ*rate
princ = princ + princ*rate - paym
if princ < 0:
    lastpayment = paym + princ
    princ = 0
month += 1
```

\*\*\*\*\*  
\*\* The Ultimate Mortgage Calculator \*\*  
\*\*\*\*\*

What is the prinipal 40000

What is the interest rate (in percents)? 4

Your minimum rate is 133.33

What is the monthly payment? 1950

This is what your mortgage scheme looks like

Month Interest Principal

-----

1	133.33	38183.33
2	127.28	36360.61
3	121.20	34531.81
4	115.11	32696.92
5	108.99	30855.91
6	102.85	29008.76
7	96.70	27155.46
8	90.52	25295.98
9	84.32	23430.30
10	78.10	21558.40
11	71.86	19680.26
12	65.60	17795.86
13	59.32	15905.18
14	53.02	14008.20
15	46.69	12104.89
16	40.35	10195.24
17	33.98	8279.22
18	27.60	6356.82
19	21.19	4428.01
20	14.76	2492.77
21	8.31	551.08
22	1.84	0.00

You paid of the loan in 22 months, and your last payment was 552.92

# Pretty-Printing Tables

- Format Strings revisited:
  - Format string — blueprint
  - Uses { } to denote spots where variables get inserted

# Pretty-Printing Tables

- Syntax
  - `{a:^10.3f}`
    - `a` — the number of the variable
      - Can be left out
    - `:` — what follows is the formatting instruction
    - `10` — number of spaces for the variable
    - `.` — what follows is the precision
    - `3` — precision
    - `f` — print in floating point format

# Pretty-Printing Tables

- If the variable is larger than the space given:
  - Full value is printed out
  - Alignment by default is
    - left (<) for strings
    - right (>) for numbers

# Pretty-Printing Tables

- Task:
  - A program that gives a table for the log and the exponential function between 1 and 10
  - Hint:  $x=1+i/10$

x		exp(x)		log(x)
1.00		2.71828		0.00000
1.10		3.00417		0.09531
1.20		3.32012		0.18232
1.30		3.66930		0.26236
1.40		4.05520		0.33647
1.50		4.48169		0.40547
1.60		4.95303		0.47000
1.70		5.47395		0.53063