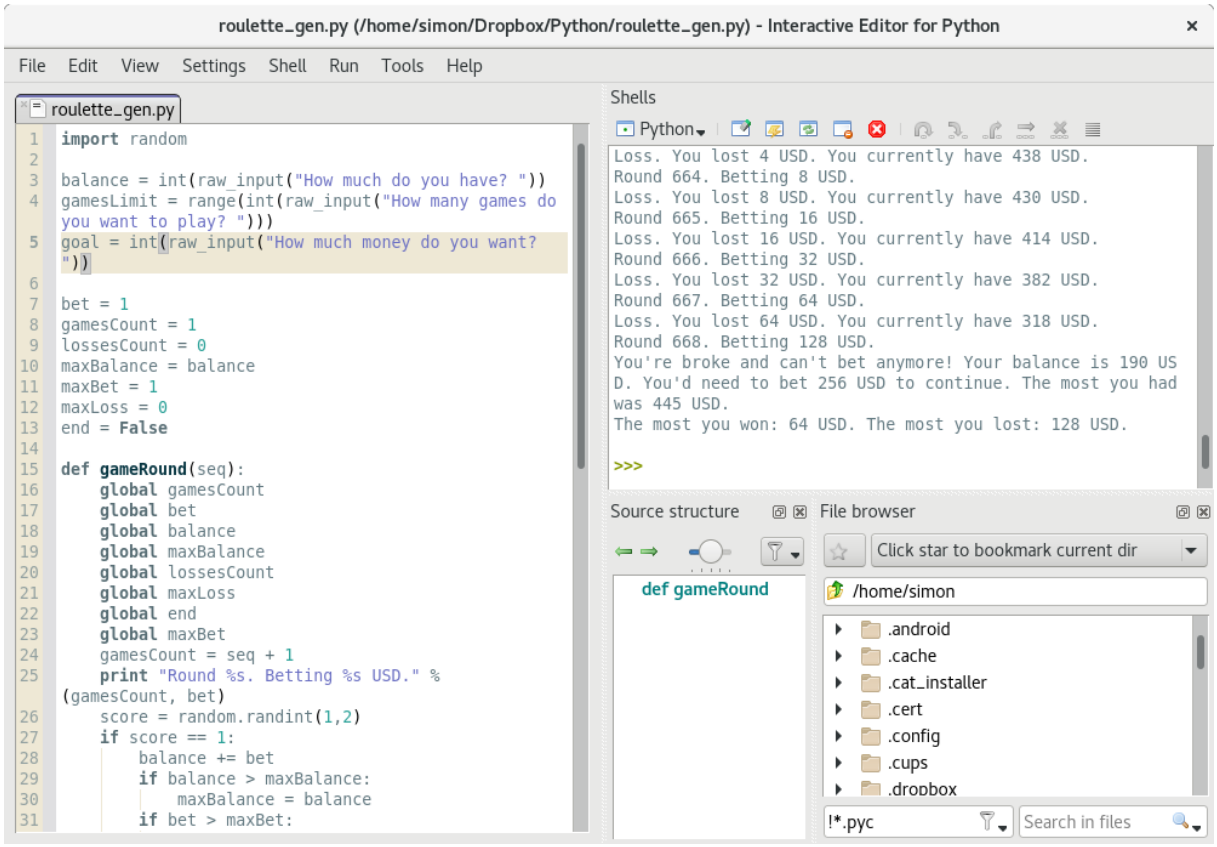
1 PyZo environment (`http://www.pyzo.org/`)



- interactive shell
- block of code (`Ctrl+E` to compile, `Ctrl+S` to save)
- Python 2.7.X (has to be installed, check with `Win+R` → `cmd` → `python -V`)

2 Variables

Variables are reserved memory locations to store values. When you create a variable, you reserve some space in memory.

2.1 Assigning values to variables

We use the equal sign (`=`) to **assign values** to variables:

```
students = 15
latitude = 49.2041869
longitude = 16.5980044
```

Textual values (= strings) are inside quotes:

```
address = 'Kotlářská 2'
```

What is **not allowed** as a value:

```
# mixing data types:
bad1 = 'Kotlářská'2
bad2 = 15students

# will result in an error:
bad3 =
bad4 = 'Kotlářská
bad5 = 10 000

# will be stored as a wrong type!
bad6 = 49,2041869 # float has a decimal point
```

What is **not allowed** in variable names:

```
12apes = 'movie' # beggining with a number
the-answer = 42 # special characters
the answer = 42 # spaces
round = 1 # built-in names and keywords
```

What **is allowed** in variable names:

```
point1 = 'Brno' # number not in the beginning
the_answer = 42 # underscore the only allowed special character
theAnswer = 42 # camelCase naming convention
```

3 Basic commands

3.1 Print

Outputs result to command line

- debugging (e.g. why is there an invalid value on *this* variable?)
- informative purposes

Simple text:

```
print 'Hi there!'
print 'Kotlářská 267/2, Brno, 602 00'
print '''
Hi, I am a multiline comment,
and don't mind being all over the place!
'''
```

Printing variables:

```
latitude = 49.2041869
longitude = 16.5980044
address = 'Kotlářská 2'

print address
print 'I am at: ' + address
print 'I am at %s, N%s E%s' %(address, latitude, longitude)
```

3.2 Input

Get data from the user / let him change the program behaviour:

```
city = raw_input('Where are you? ')
print city + ' is the best!'
```

3.3 Comments

Documentation, informative purposes, annotations.

```
# What are the coordinates most to the north and south?
print max(latitudeList) # northernmost, e.g. 89.5
print min(latitudeList) # southernmost, e.g. -60.3
```

Recommendation

Comments **shouldn't** be necessary to **understand the code!**
Use self-explanatory variable names, try to write *clean* code.

- bad names: `var1, var2, …; a, aaa, b, ab, …`
- good names: `cities, latDD, lonDMS`

4 Built-in types (`https://docs.python.org/2.7/library/stdtypes.html`)

4.1 Truth testing

```
False
True
```

We use *boolean operators* to compare values:

```
x or y          # True or False → True
x and y         # True and False → False
not x           # not True → False
```

Other data types can be *evaluated as boolean values*:

```
1 or False      # True
"Brno" and 0    # False
not 0           # True
"" or 0         # False
```

4.2 Numerical types

Four in total (`int, long, float, complex`), we will mostly use only `int` and `float`.

```
type(42)        # <type 'int'>
type(49.2)      # <type 'float'>
```

4.3 Operations

Basic arithmetic operations. Some of them *also work with other data types*!

```
3 + 5           # obvious
"Hello " + user # but also this!
5 - 3
"bad" - "b"     # this throws error! what did you expect …
10 * 3
"-" * 5         # "-----" this also works! (can be very useful)
3 / 2           # = 1 ! careful, Python 2 needs to be told if the result is
    ↪ float or int!
3.0 / 2         # either like this
3 / float(2)    # or like this (float / int, int / float, float / float all
    ↪ result in a float)
5 % 4           # = 1; modulo (remainder = zbytek)
3 ** 2          # power
```

When we modify variables:

```
a = 5
a = a + 3       # a = 8
a += 1          # a = 9; easier, right?
```

```
a ++            # a = 10; this is even better
a -= 10         # also *= /=
```

also `round()`, `math.floor()`, …

4.4 Comparisons

```
"A" < "B"       # less than (alphabet order)
lat <= 90       # less than or equal to
5 > 3           # greater than
city == "Brno"  # equal to
city != "Praha" # not equal to
```

4.5 Strings

```
"This is a string"
'And this one is too'
"It's because of this - apostrophes."
```

Usings substrings with `[]` – usually we count 1,2,3, …; here we count 0,1,2,3, …!

```
message = "hello everyone"
message[0]            # 'h' → character at index 0
message[-1]           # 'e' → character at the last index
message[0:5]          # 'hello' → from index 0 to (not including!) index 5
message[6:]           # from index 6 to the end
```

String operations:

```
"a" in message        # False
"hello" in message    # True
"hello " + user       # merging strings, we know this already
"-" * 5               # string repetition, we know this too
```

A lot of string methods (`str.find()`, `str.count()`, `str.isdigit()`, `len(str)`, `str.replace()`):

```
message.find("l")     # 2
message.find("a")     # -1 (e.g. there is no "a")
message.count("l")    # 2
len(message)          # 14
message.replace("hello", "bye") # "bye everyone"
```

4.6 Data type conversion

```
# int()
int(5.2)        # 5
int("5.2")      # ERROR!
int("5")        # 5

# float()
float(5)        # 5.0
float("5.2")    # 5.2

# str()
str(42)         # '42'
str(False)      # 'False'

# bool()
a = 5
bool(a)         # True
bool('False')   # True !!
```

We do this often to avoid data type conflicts:

```
students = 5
print "Number of students: " + students       # ERROR
print "Number of students: " + str(students) # correct
```