Variables, Assignments, and Functions

Basic Programming in Python

Sebastian Höffner Aline Vilks

Wed, 12 Apr 2017

Python scripting

- Scripts make code reusable
- From now on, write all your homework in scripts
- You can still explore using the interactive interpreter
- To replay your code: `python my_code.py` – or just click the play button in spyder!

Python scripting

- Scripts make code reusable
- From now on, write all your homework in scripts
- You can still explore using the interactive interpreter
- To replay your code: `python my_code.py` – or just click the play button in spyder!

During the first homework, you copied all your commands into files.

You can run them, share them, modify them.

Common mistakes & bonus solutions

- Please name your files as we suggest, if we give you some names. `nicolas.py`, `nicholas.docx`, or `Nicholas.py` are not the same as `nicholas.py`.
- For the bonus questions we needed *escaping*, that means we can write:

```
print("This is \"interesting\".")
```

Output:

```
This is "interesting".
```

Some conventions

- No spaces between function names and parentheses, but after commas: `function_name(arg0, arg1)`
- Spaces around math operators (we learn more about them today): `a * b`
- At least one, better two lines after your `imports`:

```
import turtle



turtle.shape('turtle')
turtle.done()
```

- `imports` should be the first statements in your files.

Variable vs. value

- Variables are placeholders
- Values are the contents

Variable vs. value: Random mug

Figure 1: Mug of Tea (Factorylad, Wikimedia Commons)

Variable vs. value: Mensa mug

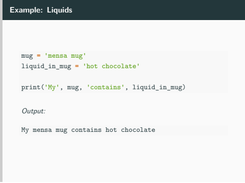
Figure 2: Mug of Hot Chocolate (own picture)

Example: Liquids

```python
mug = 'mensa mug'
liquid_in_mug = 'hot chocolate'

print('My', mug, 'contains', liquid_in_mug)
```

Output:

```
My mensa mug contains hot chocolate
```



Example: Liquids

```
mug = 'mensa mug'
liquid_in_mug = 'hot chocolate'

print('My', mug, 'contains', liquid_in_mug)
```

Output:

```
My mensa mug contains hot chocolate
```

You might know this concept from Logics, Mathematics, or Statistics classes.

A variable is thus just a placeholder for a concept, while the value is its realization.

Example: Names

```python
name_one = 'Aline'
name_two = 'Basti'
greeting = 'Good morning,'

print(greeting, name_one)
print(greeting, name_two)
```

Output:

```
Good morning, Aline
Good morning, Basti
```

Example: Variable to variable assignment

```python
my_fruit = 'Raspberry'
your_fruit = my_fruit

print(my_fruit)
print(your_fruit)
```

Output:

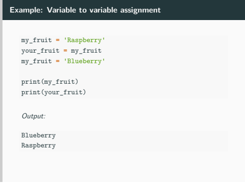
```
Raspberry
Raspberry
```

Example: Variable to variable assignment

```
my_fruit = 'Raspberry'
your_fruit = my_fruit
my_fruit = 'Blueberry'

print(my_fruit)
print(your_fruit)
```

Output:

```
Blueberry
Raspberry
```



Example: Variable to variable assignment

```
my_fruit = 'Raspberry'
your_fruit = my_fruit
my_fruit = 'Blueberry'

print(my_fruit)
print(your_fruit)
```

Output:

```
Blueberry
Raspberry
```

We can copy over one variable to another variable.

However, be careful with assigning variables to variables, we will later learn that sometimes we hold the same fruit, not a copy!

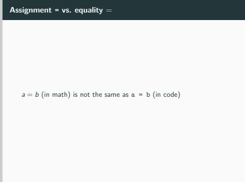
Variables and assignments

A notorious example for a bad idea was the choice of the equal sign to denote assignment.[1]

[1] Wirth (2006): Good Ideas, Through the Looking Glass.

Assignment `=` vs. equality $=$

$a = b$ (in math) is not the same as `a = b` (in code)



Assignment = vs. equality $=$

$a = b$ (in math) is not the same as `a = b` (in code)

- Mathematical equality works different than assignments
- In maths, both sides of the equality sign have to be equal
- In programming, after “assigning b to a”, `a` has the value of `b`
- It does not matter what value `a` or `b` had before (in python)

Variables and math

```
a = 3
b = 5
c = 2
c = a * b
print(c)
```

Variables and math

```python
a = 3
b = 5
c = 2
c = a * b
print(c)
```

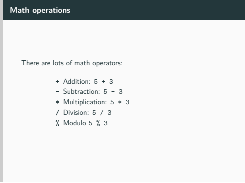
Output:

```
15
```

Math operations

There are lots of math operators:

`+`	Addition: `5 + 3`
`-`	Subtraction: `5 - 3`
`*`	Multiplication: `5 * 3`
`/`	Division: `5 / 3`
`%`	Modulo `5 % 3`



Math operations

There are lots of math operators:

- `+` Addition: `5 + 3`
- `-` Subtraction: `5 - 3`
- `*` Multiplication: `5 * 3`
- `/` Division: `5 / 3`
- `%` Modulo `5 % 3`

They all work just as we are used from mathematics.

Try out:

- `5 + 3 * 2`
- `(5 + 3) * 2`
- `5 - 2 + 3`
- `5 - (2 + 3)`
- etc.

Recap: modulo

You have 13 apples and want to share between you and your three friends when you head out for a hike. But since the apples become brown if you slice them, you only take whole apples with you. How many apples do you leave at home?

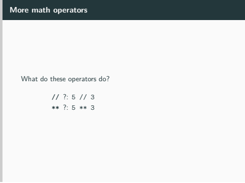
$$13 : 4 = 3R1 \tag{1}$$
$$\underline{12} \tag{2}$$
$$1 \tag{3}$$

More math operators

What do these operators do?

`//` ?: `5 // 3`

`**` ?: `5 ** 3`



More math operators

What do these operators do?

```
// ?: 5 // 3
** ?: 5 ** 3
```

// is the integer division (floors the value / cuts off the remainder)

** is exponentiation

Python handles very large numbers

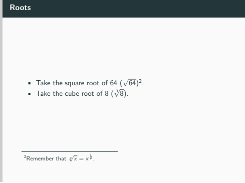
Try:

```
>>> 2394 ** 23
5246327039149698177875720100207165927515907418409239739316
>>> 5.331 ** 413
1.485887341202073e+300
```

Roots

- Take the square root of 64 ($\sqrt{64}$)[2].
- Take the cube root of 8 ($\sqrt[3]{8}$).

[2] Remember that $\sqrt[p]{x} = x^{\frac{1}{p}}$.



Roots

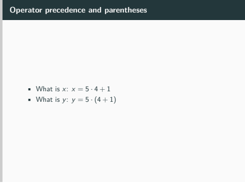
- Take the square root of 64 ($\sqrt{64}$)[2].
- Take the cube root of 8 ($\sqrt[3]{8}$).

[2]Remember that $\sqrt[p]{x} = x^{\frac{1}{p}}$.

```
x = 64
sqrt_x = x ** (1 / 2)
print(sqrt_x)
y = 8
cbrt_y = y ** (1 / 3)
print(cbrt_y)
```

Operator precedence and parentheses

- What is x: $x = 5 \cdot 4 + 1$
- What is y: $y = 5 \cdot (4 + 1)$



Operator precedence and parentheses

- What is x: $x = 5 \cdot 4 + 1$
- What is y: $y = 5 \cdot (4 + 1)$

Multiplication has a higher precedence than addition.

Parentheses overwrite precedence.

Operator Precedences

Strength[3]	Operators	Explanations
strongest	`(...)`	Parentheses[4]
stronger	`**`	Exponentiation[5]
strong	`+x`, `-x`	Positive/Negative numbers
weak	`*`, `/`, `//`, `%`	Multiplication, (Integer) Division, Modulo
weaker	`+`, `-`	Addition, Subtraction
weakest	`=`	Assignment (not equality!)

[3]Equally strong operators are executed left to right, unless overwritten with parentheses.

[4]Parentheses (and other brackets) are resolved from inner to outer.

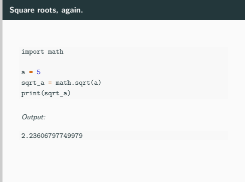
[5]Exception: `**` is weaker than `-x` on its *right hand side* (i.e. in the exponent).

Square roots, again.

```
import math


a = 5
sqrt_a = math.sqrt(a)
print(sqrt_a)
```

Output:

```
2.23606797749979
```



Square roots, again.

```
import math

a = 5
sqrt_a = math.sqrt(a)
print(sqrt_a)
```

Output:

```
2.23606797749979
```

- There are lots of useful math functions already implemented in Python.
- Search the web for `python 3 math`. Do it now!
- All functions listed there are available by calling `math.function(...)` after `import math`.
- `math` is a **module**

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

Let’s solve a “real world” problem!

In the beat ‘em up game Castle Crashers, four heroic knights are on an epic journey to save four princesses who were kidnapped by a dark wizard. During the knights’ journey they fight many evil-doers.

Figure 3: Castle Crashers, Screenshot (The Behemoth 2012)

Fighting for princesses

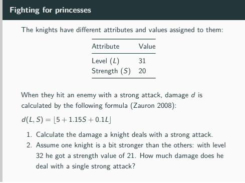
The knights have different attributes and values assigned to them:

Attribute	Value
Level (L)	31
Strength (S)	20

When they hit an enemy with a strong attack, damage d is calculated by the following formula (Zauron 2008):

$$d(L,S) = \lfloor 5 + 1.15S + 0.1L \rfloor$$

1. Calculate the damage a knight deals with a strong attack.
2. Assume one knight is a bit stronger than the others: with level 32 he got a strength value of 21. How much damage does he deal with a single strong attack?



Fighting for princesses

The knights have different attributes and values assigned to them:

Attribute	Value
Level (L)	31
Strength (S)	20

When they hit an enemy with a strong attack, damage d is calculated by the following formula (Zauron 2008):

$d(L, S) = \lfloor 5 + 1.15S + 0.1L \rfloor$

1. Calculate the damage a knight deals with a strong attack.
2. Assume one knight is a bit stronger than the others: with level 32 he got a strength value of 21. How much damage does he deal with a single strong attack?

```python
import math

level = 31
strength = 20

damage = math.floor(5 + 1.15 * strength + 0.1 * level)

print(damage)
# Or: (5 + 1.15 * strength + 0.1 * level) // 1
```

1. 31
2. 32

Fighting for princesses solution

File: `castlecrashers.py`

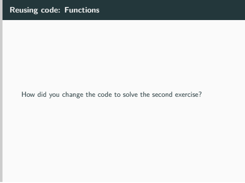
```
import math


level = 31
strength = 20


damage = math.floor(5 + 1.15 * strength + 0.1 * level)


print(damage)
# Or: (5 + 1.15 * strength + 0.1 * level) // 1
```

Output:

```
31
```

Reusing code: Functions

How did you change the code to solve the second exercise?


Reusing code: Functions

How did you change the code to solve the second exercise?

Split your code into small parts which solve one task.

- This follows a pattern called DNRY (Do Not Repeat Yourself)
- Fewer mistakes/easy to fix: only need to change them in one place
- We will learn later: It's easier to test
- Makes code reusable

Reusing code: Functions

```python
def strong_attack_damage(level, strength):
    return (5 + 1.15 * strength + 0.1 * level) // 1

level = 31
strength = 20
damage = strong_attack_damage(level, strength)
print(damage)
```

Output:

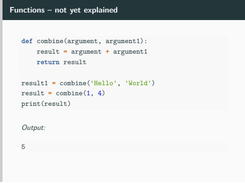
```
31.0
```

Functions – not yet explained

```
def combine(argument, argument1):
    result = argument + argument1
    return result

result1 = combine('Hello', 'World')
result = combine(1, 4)
print(result)
```

Output:

```
5
```



Functions – not yet explained

```
def combine(argument, argument1):
    result = argument + argument1
    return result

result1 = combine('Hello', 'World')
result = combine(1, 4)
print(result)
```

Output:

```
5
```

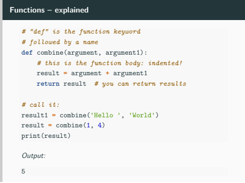
- `result1` is `HelloWorld`

Functions – explained

```python
# "def" is the function keyword
# followed by a name
def combine(argument, argument1):
    # this is the function body: indented!
    result = argument + argument1
    return result  # you can return results

# call it:
result1 = combine('Hello ', 'World')
result = combine(1, 4)
print(result)
```

Output:

```
5
```



Functions – explained

```
# "def" is the function keyword
# followed by a name
def combine(argument, argument1):
    # this is the function body: indented!
    result = argument + argument1
    return result  # you can return results

# call it:
result1 = combine('Hello ', 'World')
result = combine(1, 4)
print(result)
```

Output:

```
5
```

- Watch out for indentation
- Take care of enough whitespace around a function (at least one line above and below)
- You can have arbitrarily many arguments
- We will discuss functions in much more details soon, but for now this should be sufficient

Your second homework

- Calculate the area of different St. Nicholas' houses. Use the `random` module to generate useful random variables.
- Extend the repertoire of the four Castle Crasher knights and let two of them fight.

The last slide

dad you know so much stuff
that's because my dad taught me.
did he google everything too?
lizclimo.tumblr.com

Figure 4: happy father's day (Climo 2014)

References

Climo, Liz. 2014. "Happy Father's Day." *Hi, I'm Liz*, June.

The Behemoth. 2012. "Screenshot 7." *Castle Crashers*.

Wirth, Niklaus. 2006. "Good Ideas, Through the Looking Glass." *IEEE Computer* 39 (1): 28–39.

Zauron. 2008. "Character Guide." *GameFAQs: Castle Crashers*, September.