while loops, random numbers, tuples

while loops

```
while test:
    statements
```

```
>>> n = 91
>>> factor = 2      # find first factor of n

>>> while n % factor != 0:
...     factor += 1
...

>>> factor
7
```

exercise

```
>>> smallest_factor(91)
7
```

- write a function `smallest_factor` that takes any integer and returns its smallest factor

solution

```
def smallest_factor(n):
    if n < 2:
        return n

    factor = 2
    while n % factor != 0:
        factor += 1

    return factor
```

bool

```
>>> b = 5 < 10
>>> b
True

>>> if b:
...     print "The bool value is true"
...
The bool value is true

>>> b = not b
>>> b
False
```

- like java's `boolean` type

logical operators

Operator	Meaning	Example	Result
`==`	equals	`1 + 1 == 2`	`True`
`!=`	does not equal	`3.2 != 2.5`	`True`
`<`	less than	`10 < 5`	`False`
`>`	greater than	`10 > 5`	`True`
`<=`	less than or equal to	`126 <= 100`	`False`
`>=`	greater than or equal to	`5.0 >= 5.0`	`True`

Operator	Example	Result
`and`	`(2 == 3) and (-1 < 5)`	`False`
`or`	`(2 == 3) or  (-1 < 5)`	`True`
`not`	`not (2 == 3)`	`True`

exercise

```
>>> is_prime(11)
True
>>> is_prime(12)
False
```

- write a function `is_prime` that takes any integer and returns `True` if it is prime, or `False` otherwise

solution

```
def is_prime(n):
    return smallest_factor(n) == n
```

random numbers

- `from random import *`
- `randint(min, max)` returns a random int in the range [min, max] inclusive
- `choice(sequence)` returns a randomly chosen value from a sequence (string, range, list, tuple...)

tuples

```
>>> y = -5
>>> p = (3, y, 42)
>>> p
(3, -5, 42)

>>> a, b, c = p
>>> a
3
>>> b
-5
>>> c
42
```

- can be used to store multiple values in a single variable

divmod

```
>>> divmod(20, 7)
(2, 6)
```

- `divmod(a, b)` returns a tuple whose first value is `(a / b)`, and whose second value is `(a % b)`

exercise

```
>>> roll_dice()
(3, 1)
>>> roll_dice()
(6, 3)
```

- write a function `roll_dice` that rolls two dice and returns their values as a tuple

solution

```python
from random import *

def roll_dice():
    roll1 = randint(1, 6)
    roll2 = randint(1, 6)
    return (roll1, roll2)
```

exercise

```
>>> craps()
rolled 4 + 4 = 8
rolled 3 + 1 = 4
rolled 2 + 2 = 4
rolled 6 + 5 = 11
```

- write a function `craps` that calls `roll_dice` repeatedly, until it returns a pair of dice whose sum are 7 or 11

solution

```
def craps():
    total = 0 # prime the loop
    while total != 7 and total != 11:
        (roll1, roll2) = roll_dice()
        total = roll1 + roll2
        print "rolled", roll1, "+", roll2, "=", total
```

exercise

```
>>> loaded_dice()
(6, 1)
>>> loaded_dice()
(5, 2)
>>> loaded_dice()
(3, 4)
```

- write a function loaded_dice that always returns a roll of 7

solution

```
def loaded_dice():
    roll = randint(1, 6)
    return (roll, 7 - roll)
```

bonus content!

higher-order functions

```
>>> filter(is_prime, range(100))
[0, 1, 2, 3, 5, 7, 11, 13, 17, 19,
23, 29, 31, 37, 41, 43, 47, 53, 59,
61, 67, 71, 73, 79, 83, 89, 97]
```

- `filter(func, sequence)` returns all values in sequence for which `func(value)` returns `True`