Python and Rust

Dirk Lüdtke

prime test

```python
def is_prime(n):
    if n < 2:
        return False
    limit = sqrt(n) + 1
    for i in range(2, limit):
        if n % i == 0:
            return False
    return True
```

```rust
fn is_prime(n: u64) -> bool {
    if n < 2 {
        return false;
    }
    let limit = sqrt(n) + 1;
    for i in 2..limit {
        if n % i == 0 {
            return false;
        }
    }
    true
}
```

prime test

```python
def is_prime(n):
    if n < 2:
        return False
    limit = sqrt(n) + 1
    for i in range(2, limit):
        if n % i == 0:
            return False
    return True
```

```rust
fn is_prime(n: u64) -> bool {
    if n < 2 {
        return false;
    }
    let limit = sqrt(n) + 1;
    for i in 2..limit {
        if n % i == 0 {
            return false;
        }
    }
    true
}
```

integer division: modulo vs. remainder

```python
assert  2 // 3 ==  0 and  2 % 3 == 2
assert  1 // 3 ==  0 and  1 % 3 == 1
assert  0 // 3 ==  0 and  0 % 3 == 0
assert -1 // 3 == -1 and -1 % 3 == 2
assert -2 // 3 == -1 and -2 % 3 == 1
```

```rust
fn main() {
    assert!( 2 / 3 == 0 &&  2 % 3 ==  2);
    assert!( 1 / 3 == 0 &&  1 % 3 ==  1);
    assert!( 0 / 3 == 0 &&  0 % 3 ==  0);
    assert!(-1 / 3 == 0 && -1 % 3 == -1);
    assert!(-2 / 3 == 0 && -2 % 3 == -2);
}
```

integer square root (naïve approach)

```python
def sqrt(n):
    return int(math.sqrt(n))
```

```rust
fn sqrt(n: u64) -> u64 {
    (n as f64).sqrt() as u64
}
```

integer square root

```python
def sqrt(n):
    return int(math.sqrt(n))
```

Since Python 3.8 (2019)

```python
math.isqrt(i)
```

```rust
fn sqrt(n: u64) -> u64 {
    (n as f64).sqrt() as u64
}
```

Rust 1.76. (Feb 2024)
in nightly build (unstable feature)

```rust
u64::isqrt(i)
```

runtime

```python
t = time.time()
assert is_prime(2 ** 61 - 1)
print(f'Mersenne Prime 61 in Python needed  {
   time.time() - t:.3f} s')
```

```
Mersenne Prime 61 in Python needed  77.699 s
```

```rust
let t = std::time::Instant::now();
assert!(is_prime(2_u64.pow(61) - 1));
println!(
  "Mersenne Prime 61 in Rust needed {:.3}
s",
  t.elapsed().as_secs_f32()
);
```

```
Mersenne Prime 61 in Rust needed  2.291 s
```

optimizations

```python
for i in range(2, limit):
    if n % i == 0:
        return False
```

```rust
for i in 2..limit {
    if n % i == 0 {
        return false;
    }
}
```

no optimization: 77.699 s

bool-array: 110.838 s

list of ints: 4.419 s

15-20 Byte per int

no optimization: 2.291 s

bool-array: 1.513 s

vector of ints: 0.114 s

4 Byte per int (u32)

call Rust from Python

```
pip install maturin
maturin init --bindings pyo3
...
maturin build -r
pip install target/wheels/...whl
```

```
from pynrs import is_prime
assert is_prime(2 ** 61 - 1)
```

```
result:   0.303 s
```

```
use pyo3::prelude::*;

#[pyfunction]
fn is_prime(n: u64) -> PyResult<bool> {
    ...
            return Ok(false);
    ...
    Ok(true)
}

#[pymodule]
fn pynrs(_py: Python, m: &PyModule) -> PyResult<()> {
    m.add_function(wrap_pyfunction!(is_prime, m)?)?;
    Ok(())
}
```

conclusions

python is
- flexible and convenient,
- allows rapid changes

rust is
- fast and small,
- compiler ensures correctness

conclusions

python is
- flexible and convenient,
- allows rapid changes

rust is
- fast and small,
- compiler ensures correctness

the good things can be combined

cf. https://github.com/daimonji-devel/pynrs