Python and Rust

Dirk Lüdtke

prime test

```
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</pre>
```





```
fn is prime(n: u64) -> bool {
  if n < 2 {
  let limit = sqrt(n) + 1;
  for i in 2..limit {
          return false;
```

prime test

```
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</pre>
```





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

integer division: modulo vs. remainder



```
R
```

```
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
```

```
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)



```
B
```

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

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

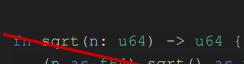
integer square root



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

Since Python 3.8 (2019)

math.isqrt(i)
```



```
B
```

```
In sqrt(n: u64) -> u64 {
    (n as fo4).sqrt() as u64
}

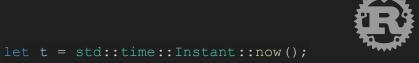
Rust 1.76. (Feb 2024)
in nightly build (unstable feature)
u64::isqrt(i)
```

runtime



```
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



```
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

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



```
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
```



#[pymodule]

Ok(())

fn pynrs(py: Python, m: &PyModule) -> PyResult<()> {

m.add function(wrap pyfunction!(is prime, m)?)?;

conclusions

```
python is
   flexible and convenient,
   allows rapid changes
```

```
rust is
    fast and small,
    compiler ensures correctness
```

conclusions

```
python is rust is

flexible and convenient, fast and small,

allows rapid changes correctness
```

the good things can be combined

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