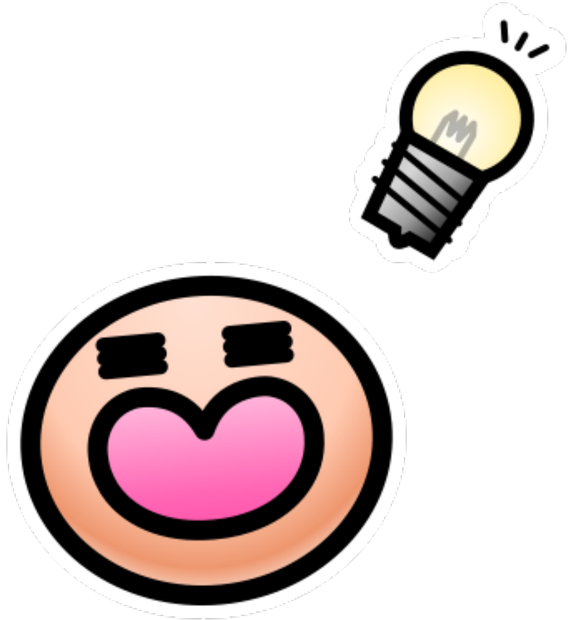
テストケースの名前は
どうつけるべきか？

2024.11.19 Kuniwak, SWET 2G , DeNA Co., Ltd.



今日伝えたいこと

テストケースには同値パーティションの名前をつける

最初にテストケースクイズ

どちらのテストケース名がレビューしやすい？

A

「1 のとき"1"」

「3 のとき"Fizz"」

「5 のとき"Buzz"」

「15 のとき"FizzBuzz"」

B

「3でも5でも割り切れないとき10進数文字列」

「3で割り切れて5で割り切れないとき"Fizz"」

「5で割り切れて3で割り切れないとき"Buzz"」

「3でも5でも割り切れるとき"FizzBuzz"」

どちらのテストケース名がレビューしやすい？

A

「1 のとき"1"」

「3 のとき"Fizz"」

「5 のとき"Buzz"」

「15 のとき"FizzBuzz"」

どうして 1 や 3 を選んで
いるかわからない

2のときはどうなるんだろう？
試さなくていいのかな？

時間がないとよくわかんないし
LGTM ってなる（事故る）

どちらのテストケース名がレビューしやすい？

長ったらしいけど、どのパターンも考慮されていてかつ被りもない、つまり過不足がないとわかる

B

「3でも5でも割り切れないとき10進数文字列」

「3で割り切れて5で割り切れないとき"Fizz"」

「5で割り切れて3で割り切れないとき"Buzz"」

「3でも5でも割り切れるとき"FizzBuzz"」

つまり、テストケースの過不足を把握しやすい
テストケース名がいいテストケース名ということ。

過不足なくテストケースを選ぶ作業がテスト技法の
核心である。上手にテストケースを選べれば、
テストケースを選んだ理由を説明できるようになる。

テストケースを選んだ理由を
説明できるようになろう

テストケースの代表的な選び方として次の2つの方法がよく知られている：

- 同値分割法（Equivalence Class Partitioning; ECP）
- 境界値分析法（Boundary-Value Analysis; BVA）

テストケースの代表的な選び方として次の2つの方法がよく知られている：

- **同値分割法（Equivalence Class Partitioning; ECP）**
- 境界値分析法（Boundary-Value Analysis; BVA）

事前条件[1] を満たす入力すべてからなる集合を X とする。

---

[1] 事前条件とは関数の実行前の状態が満たすべき条件。
この条件を守っていない状態で関数が実行された場合、関数はどのように振る舞ってもよいのでテストはしない。

X を次の条件を満たす

$X_1, X_2, \ldots, X_N$ へと分割する：

X を次の条件を満たす

$X_1, X_2, ..., X_N$ へと分割する：

仕様から想定されるどんな

実装でも、X_n の中の入力のテスト

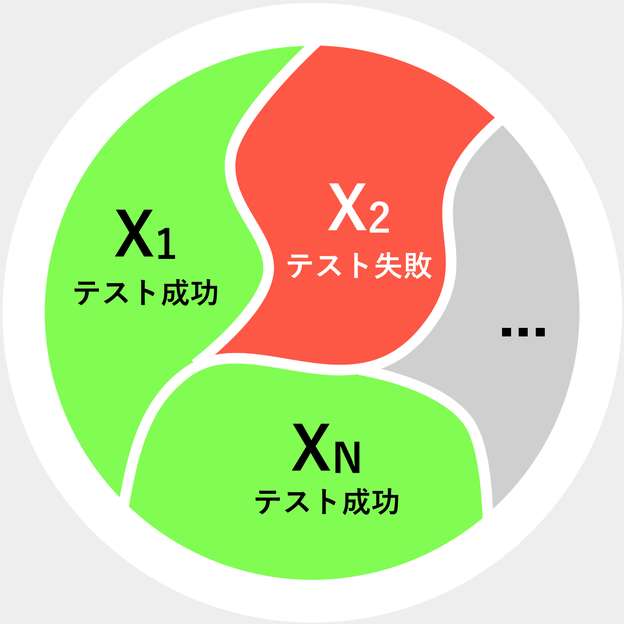
結果は X_n の中ですべて一致する[1]。

---

[1] $X_1, X_2, ..., X_N$ は $\forall prog.\ prog \in possibleImpl\ spec \rightarrow$
$(\forall n \in [1, N].\ ((\forall e \in X_n.\ test\ spec\ prog\ e)$
$\vee\ (\forall e \in X_n.\ \neg\ test\ spec\ prog\ e)))$ を満たす X の分割。

実装A

実装B



このような分割を 同値分割[1] と
いい、$X_1, X_2, ..., X_N$ のことを
同値パーティション という。

---

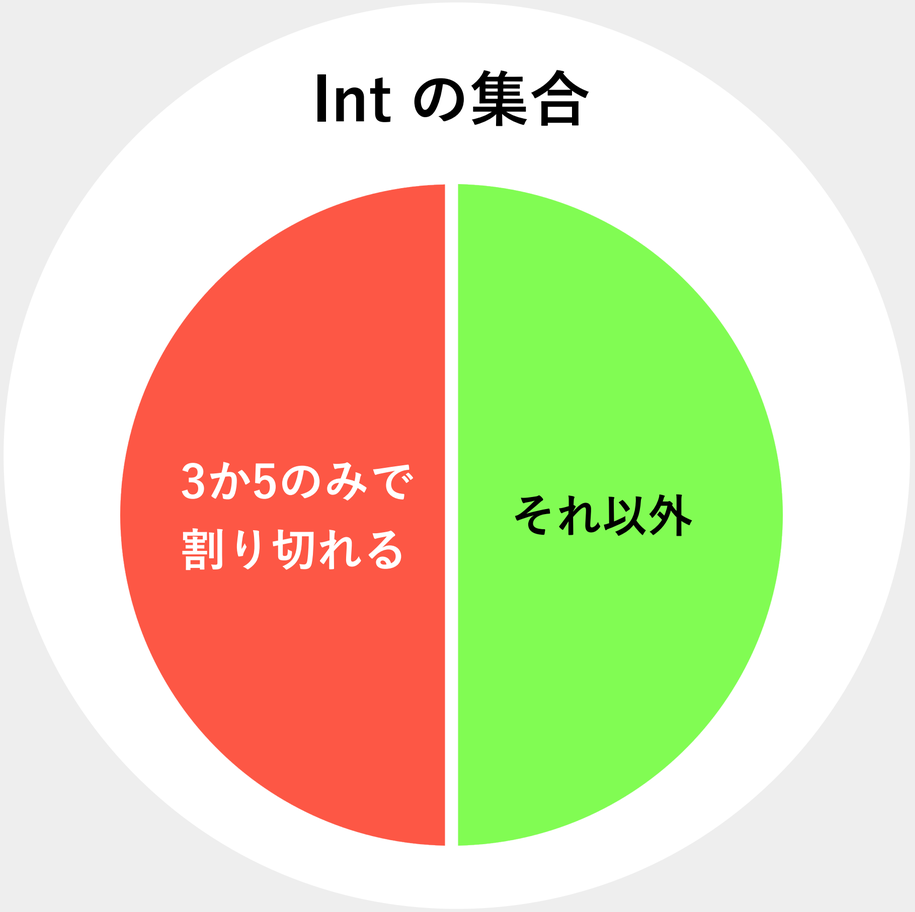
[1] 同値分割の定義は混乱していて、大きく分けると4つの派閥がある（詳細は https://blog.kuniwak.com/entry/2024/03/18/085537）。今回の定義は代表元のとりかたによらず完全正当性が保証されるように設計した形式的な定義を採用している。

FizzBuzz を同値分割してみよう。

事前条件は i > 0 だった。これを

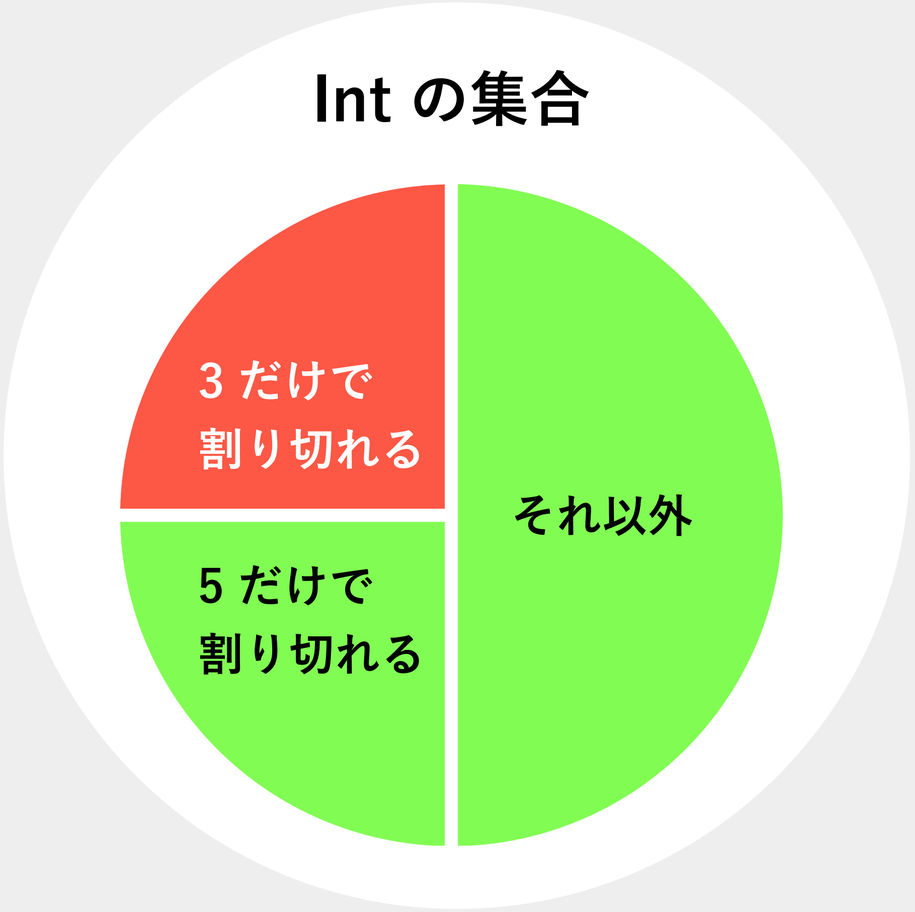
実装を想定しながら分割していく。

出力の "Fizz" と "Buzz" を取り違えていると、3 か 5 で割り切れる入力の集合のテストが失敗し、それ以外の集合はテストが成功する。



"Fizz" が typo していると想定すると、
3 で割り切れて 5 で割り切れない
入力の集合のテストが失敗し、他の
集合のテストは成功する。

つまり 3 か 5 で割り切れる部分は
3 だけで割り切れる部分とそれ以外に
分割される。



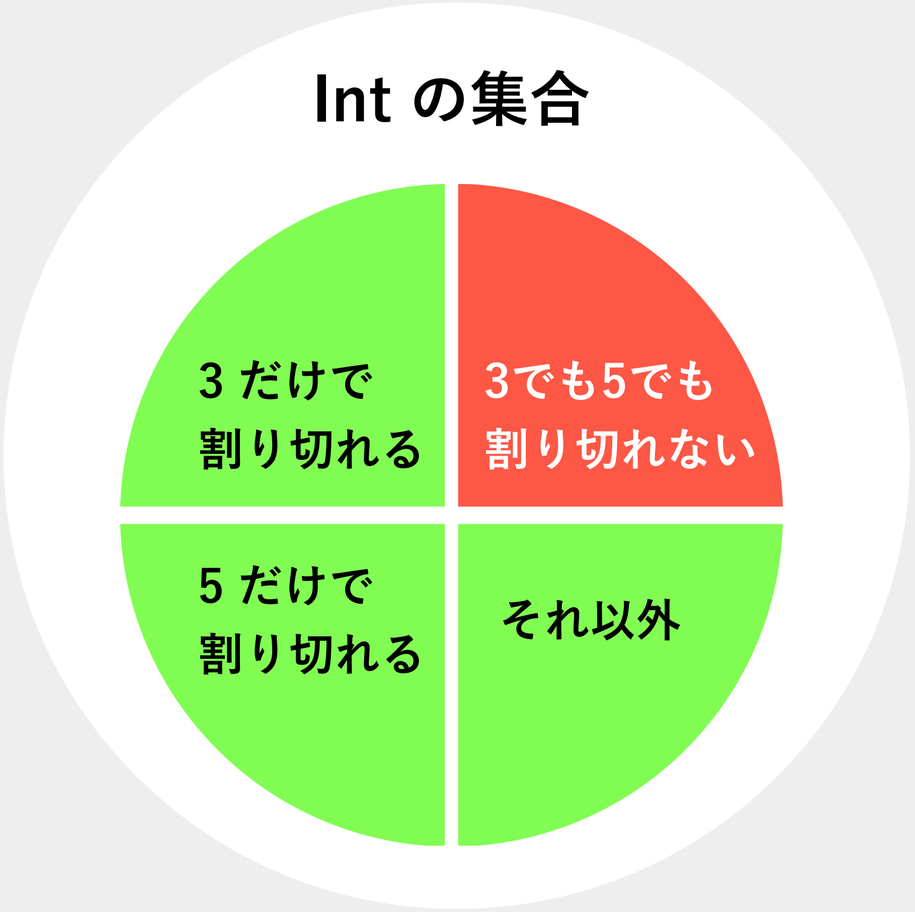
整数の10進文字列化が失敗しているとすると、

3 でも 5 でも割り切れない入力の集合の

テストが失敗し、他の集合のテストは成功する。

つまり 3 でも 5 でも割り切れない入力と

そうでない入力の集合に分割される。



結果として 4 つの同値パーティションに
分割された。

大雑把に説明すると、ここにバグがあったら
この入力のあたりが全滅するな、という
箇所を切り出していくと同値分割になる。

このようにして得られた同値パーティションにおいて、

それぞれ１つずつ任意の代表値をテストし、それがすべて

成功するなら実装は仕様を満たしていることが保証される[1]。

つまり、同値パーティションごとに任意の入力を1つ

選んでテストするだけでいい！

---

[1] https://gist.github.com/Kuniwak/a016dd5370aed2d15bd7a4715049fc12

落とし穴があります

これまでの同値分割では、実装を想定しているところに落とし穴がある。

実際の実装がここから外れていると、パーティションの中にテストを成功させる入力とそうでない入力が混ざってしまう。するとどうなるか。

運悪くテストを成功させる入力を代表値に選ぶと、

そのパーティションの他の値はテストしないから

失敗する入力、すなわちバグを見逃してしまう！

この落とし穴を回避するためには自分の実装に
どういうバグが多いか想定できればよい。

よくあるバグとして $>$ を $>=$ と間違うような
バグである境界バグが知られている。

境界バグを想定して同値分割すると、事後条件の

分岐の境界だけを含む同値パーティションができる。

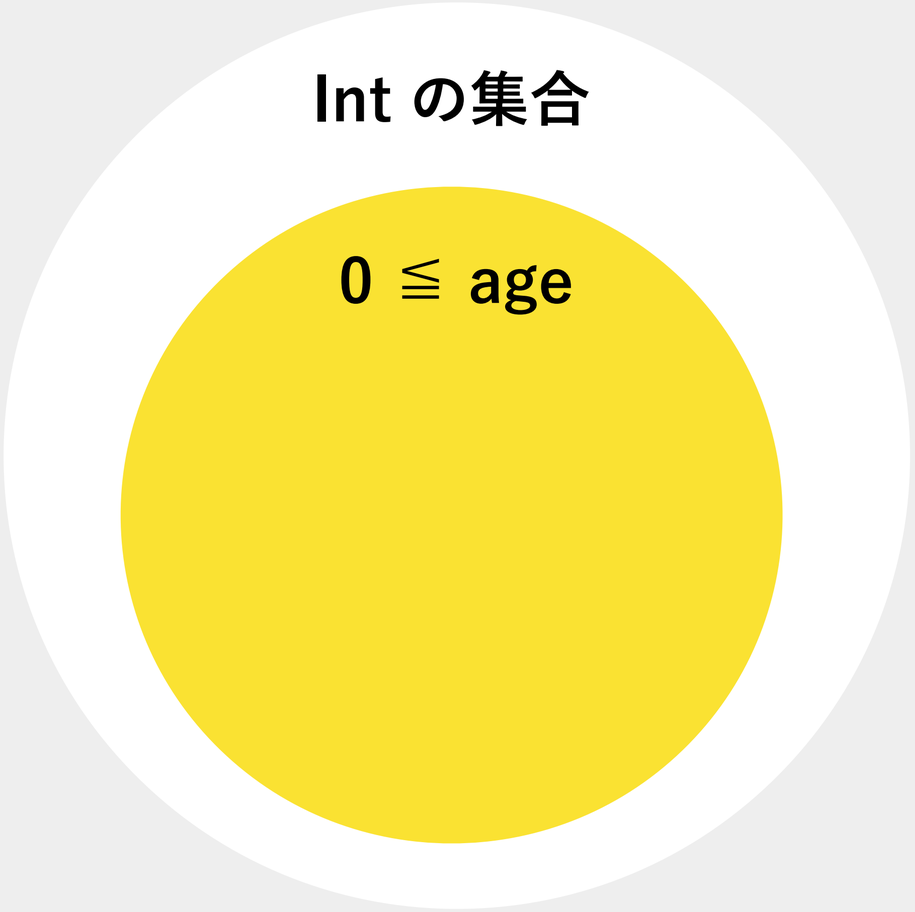
例えば、年齢を入力として、

成年なら真、未成年なら偽を返す

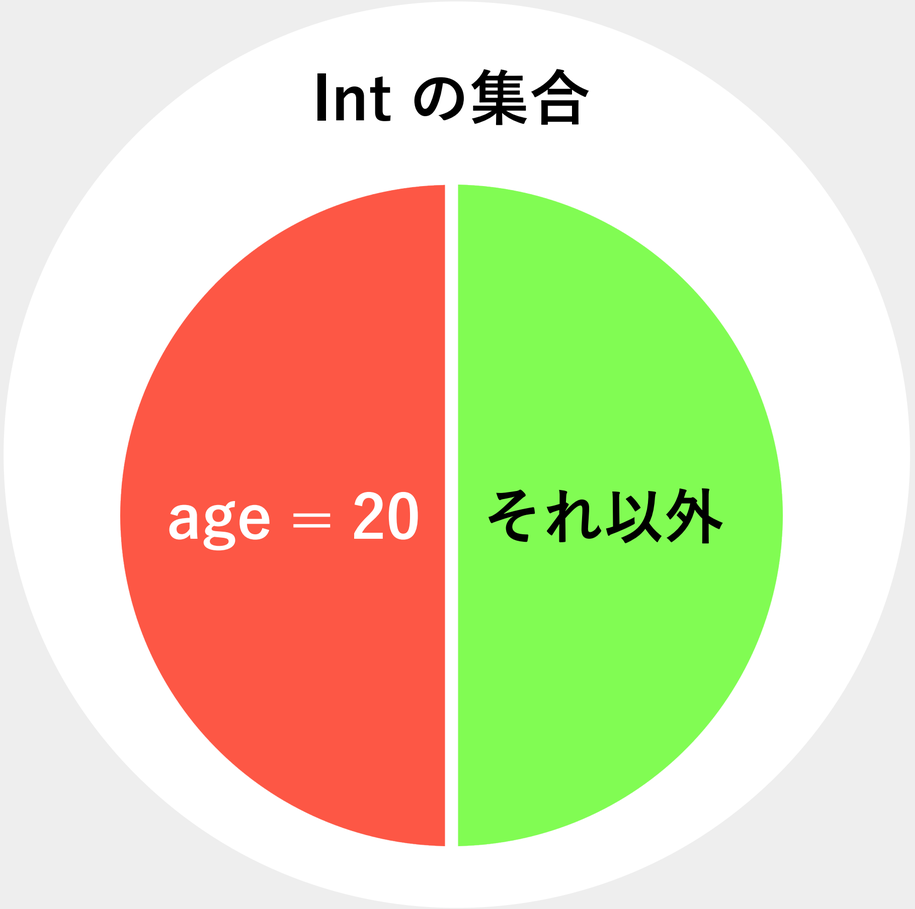
関数 isAdult(age) の仕様を考える。

事前条件は 0 ≦ age、事後条件は

出力と 20 ≦ age の真偽の一致。



正解の実装の形を return 20 ? age として、? を < と <= で間違えてしまう想定をすると、age が 20 のときのテストが失敗し、それ以外のテストは成功する。



すると、age = 20 という
事後条件の境界の値だけからなる
同値パーティションができる。

このように境界バグを考慮した
同値パーティションから代表値を
選ぶ方法を 境界値分析法 という。

テストケースの代表的な選び方として次の2つの方法がよく知られている：

- 同値分割法（Equivalence Class Partitioning; ECP）
- **境界値分析法（Boundary-Value Analysis; BVA）**

ここまでのまとめ

- 同値分割法（Equivalence Class Partitioning; ECP）
  バグがあったらこの入力すべてが失敗するという箇所を
  切り出していき、その代表値だけをテストする方法。
- 境界値分析法（Boundary-Value Analysis; BVA）
  事後条件の境界にバグがあることを想定して境界上から
  代表値を選ぶ方法。

本題：テストケースの名前

テストケースの過不足を把握しやすいテストケース名がいいテストケース名だった。

同値分割法 + 境界値分析法にもとづくテストケースの名前には、同値パーティションがわかる名前を書くとテストケースの過不足を把握しやすくできる。

先ほどの想定実装での FizzBuzz の

同値パーティションはこうだった：

- 3 でも 5 でも割り切れない入力の集合
- 3 で割り切れて 5 で割り切れない入力の集合
- 3 で割り切れて 5 で割り切れない入力の集合
- 3 でも 5 でも割り切れる入力の集合

なのでテストケース名はこうするとよい：

- 「3 でも 5 でも割り切れない場合」
- 「3 で割り切れて 5 で割り切れない場合」
- 「5 で割り切れて 3 で割り切れない場合」
- 「3 でも 5 でも割り切れる場合」

ついでに事後条件を添えるとテストレポートが

そのまま仕様の文書になるので便利：

- 「3 でも 5 でも割り切れない場合は10進数文字列」
- 「3 で割り切れて 5 で割り切れない場合は"Fizz"」
- 「5 で割り切れて 3 で割り切れない場合は"Buzz"」
- 「3 でも 5 でも割り切れる場合は"FizzBuzz"」

先ほどの想定実装の isAdult の同値パーティションは

こうだった：

- age = 20
- それ以外

このときのテストケース名は事後条件を添えて：

- 「age ＝ 20（境界値）のとき出力は真」
- 「それ以外のとき出力は age ≦ 20 の真偽と一致」

境界値からなる同値パーティションは境界値である

ことをわかりやすくするとレビューしやすい。

まとめ

テストケースには同値パーティションの名前をつける

付録

おさらい：仕様とは

- 事前条件

  関数が実行される前にどういう状態であるべきか。

  FizzBuzz でいうと、$i > 0$

- 事後条件

  事前条件を満たした状態で、関数を実行した後に

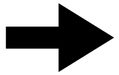
  どういう状態になるべきか。

  FizzBuzz でいうと、（続く）

おさらい：仕様とは

FizzBuzz の事後条件は：

- 3でも5でも割り切れないとき10進数文字列
- 3で割り切れて5で割り切れないとき"Fizz"
- 3で割り切れて5で割り切れないとき"Buzz"
- 3でも5でも割り切れるとき"FizzBuzz"



入力の値	出力の値
1	"1"
2	"2"
3	"Fizz"
4	"4"
5	"Buzz"
6	"Fizz"
...	...

実装が仕様を満たすとは

事前条件を満たすすべての状態に対して、

その実行後の状態が事後条件を満たすなら、

実装は仕様を満たしている。

それ以外は実装は仕様を満たしていない

（言い換えると実装にはバグがある）。

	入力の値	仕様の出力	実装の出力
	1	"1"	"1"
	2	"2"	"2"
バグ	3	"Fizz"	"3"
	4	"4"	"4"
	5	"Buzz"	"Buzz"
バグ	6	"Fizz"	"6"
	...	...	