```
##*************************************************************************
## C.Bruni (instructor)
## CS 116 Fall 2022
## Assignment 02 Problem 1
##*************************************************************************

import check

low_cmhc_rate = 0.04
med_cmhc_rate = 0.031
hi_cmhc_rate = 0.028
low_cmhc_break = 0.05
mid_cmhc_break = 0.1
high_cmhc_break = 0.15
no_cmhc_break = 0.2
num_months_per_year = 12

def compute_cmhc_insurance(percentage_down, mortgage):
  '''
  Returns the amount of cmhc_insurance a person needs
  based on the amount of the percentage_down of the house
  and the mortgage a person is getting. Valid as of April 2017.

  compute_cmhc_insurance: Float -> Float
  '''
  cmhc_insurance = 0.0
  if (low_cmhc_break <= percentage_down < mid_cmhc_break):
    cmhc_insurance = low_cmhc_rate * mortgage
  elif (mid_cmhc_break <= percentage_down < high_cmhc_break):
    cmhc_insurance = med_cmhc_rate * mortgage
  elif (high_cmhc_break <= percentage_down < no_cmhc_break):
    cmhc_insurance = hi_cmhc_rate * mortgage
  return cmhc_insurance


def monthly_payment(cost_of_house, down_payment, annual_rate, years):
'''
Returns the monthly payment of a mortgages based on cost_of_house,
the amount of down_payment, annual_rate, the number of years

monthly_payment: Nat Nat Float Nat -> Float
Requires:
   0 < cost_of_house
   0.05 * cost_of_house <= down_payment <= cost_of_house
   0 < annual_rate < 1
   1 <= years <= 30

Examples:
   monthly_payment(100000, 10000, 0.05, 30) => 498.116784
   monthly_payment(100000, 10000, 0.05, 25) => 542.441100
```

```
    '''
    mortgage = cost_of_house - down_payment
    percentage_down = down_payment/cost_of_house
    cmhc_insurance = compute_cmhc_insurance(percentage_down, mortgage)
    mortgage = mortgage + cmhc_insurance
    r = annual_rate/num_months_per_year
    n = years*num_months_per_year
    return mortgage * r/(1 - (1 + r)**(-n))

EPSILON = 0.00001

##Examples
check.within("Q2T1", monthly_payment(100000, 10000, 0.05, 30),
498.116784, \
             EPSILON)
check.within("Q2T2", monthly_payment(100000, 10000, 0.05, 25),
542.4411, \
             EPSILON)

##Tests
##Testing If Branching
check.within("down_pmt > 20%",
    monthly_payment(100000, 30000, 0.05, 25), 409.213029, EPSILON)
check.within("15% <= down_pmt < 20%",
    monthly_payment(100000, 18000, 0.1, 20), 813.474646, EPSILON)
check.within("10% <= down_pmt < 15%",
    monthly_payment(100000, 13000, 0.1, 20), 865.59546498, EPSILON)
check.within("down_pmt < 10% (and required >= 5% of cost)",
    monthly_payment(100000, 8000, 0.09, 35), 750.1244765, EPSILON)

##Check endpoints
check.within("Maximum: cost == down_pmt",
    monthly_payment(100000, 100000, 0.05, 30), 0.0, EPSILON)
check.within("Minimum: cost = 0.05 down_pmt",
    monthly_payment(200000, 10000, 0.09, 35), 1549.1701145, EPSILON)
check.within("End Point: cost = 0.1 down_pmt",
    monthly_payment(450000, 45000, 0.09, 35), 3273.601858, EPSILON)
check.within("End Point: cost = 0.15 down_pmt",
    monthly_payment(100000, 15000, 0.09, 35), 685.0530598, EPSILON)
check.within("End Point: cost = 0.2 down_pmt",
    monthly_payment(100000, 20000, 0.05, 25), 467.6720332, EPSILON)
check.within("Smallest Legal Cost",
    monthly_payment(1, 1, 0.05, 25), 0.0, EPSILON)
check.within("Smallest Legal Cost and Not Fully Paid",
    monthly_payment(2, 1, 0.05, 25), 0.0058459, EPSILON)
check.within("One Year Mortgage",
    monthly_payment(100000, 50000, 0.05, 1), 4280.3740894, EPSILON)
check.within("Low Rate 25 year mortgage",
    monthly_payment(100000, 30000, 0.01, 25), 263.81071796, EPSILON)
```

```
##*************************************************************************
## C.Bruni (instructor)
## CS 116 Fall 2022
## Assignment 02 Problem 2
##*************************************************************************

import check


def skip_sum(n):
  '''
  Returns the sum of every other digit in n
  for a total of i digits starting with the last digit.

  skip_sum: Nat Nat -> Nat
  Requires: len(str(n)) <= 12
  '''
  return (n//10**0) % 10 + (n//10**2) % 10 + (n//10**4) % 10 + \
         (n//10**6) % 10 + (n//10**8) % 10 + (n//10**10) % 10

def full_isbn(n):
  '''
  Given a partial 12 digit ISBN number n
  returns the full 13 digit one

  full_isbn: Nat -> Nat
  requires: n <= 999999999999

  Examples:
     full_isbn (0) => 0
     full_isbn (567856785678) => 5678567856782
  '''
  even_sum = skip_sum(n)
  odd_sum = skip_sum(n//10)
  checksum =  ((even_sum*3 + odd_sum) % 10)
  return n*10 + checksum


##Examples:
check.expect("Example 0", full_isbn(0), 0)
check.expect("Example 1", full_isbn (567856785678), 5678567856782)

##Tests:
check.expect("Test 1 Simple", full_isbn(111111111111), 1111111111114)
check.expect("Test 2 Lots of 0", full_isbn(100000000000),
1000000000001)
check.expect("Test 3 Repeated pairs", full_isbn(121212121212),
1212121212122)
check.expect("Test 4 Leading 0s", full_isbn(1), 13)
```

```
check.expect("Test 5 Test for largest", full_isbn(999999999999),
9999999999996)


##************************************************************
## C.Bruni (instructor)
## CS 116 Fall 2022
## Assignment 02 Problem 3
##************************************************************

import check


def sum_of_proper_divisors(n, start):
  '''
  Returns the sum of all positive divisors of n
  from start to 1

  sum_pos_div: Nat Nat Nat -> Nat
  Requires:
     1 <= n <= 1000
  '''
  if (start == 0):
    return 0
  if (n % start == 0):
    return start + sum_of_proper_divisors(n, start - 1)
  return sum_of_proper_divisors(n, start - 1)

def perfection(n):
  '''
  Returns "abundant", "perfect", or "deficient" if the sum of
  all positive proper divisors of n is greater than, equal to,
  or less than nrespectively

  perfection: Nat -> Str
  Requires: 1 <= n <= 1000

  Examples:
     perfection(1) => "deficient"
     perfection(6) => "perfect"
     perfection(3) => "deficient"
     perfection(24) => "abundant"
  '''
  s = sum_of_proper_divisors(n,n-1)
  if (s < n):
    return "deficient"
  elif (s == n):
    return "perfect"
  else:
    return "abundant"
```

```
##Examples:

check.expect("Q3T0", perfection(1), "deficient")
check.expect("Q3T1", perfection(6), "perfect")
check.expect("Q3T2", perfection(3), "deficient")
check.expect("Q3T3", perfection(24), "abundant")

##Tests:
check.expect("2nd perfect", perfection(28), "perfect")
check.expect("3rd perfect", perfection(496), "perfect")
check.expect("Maximal", perfection(1000), "abundant")
check.expect("Minimal", perfection(1), "deficient")
check.expect("Highly Abundant", perfection(960), "abundant")
check.expect("Highly Abundant", perfection(840), "abundant")
check.expect("Highly Abundant", perfection(720), "abundant")
check.expect("Very Deficient (prime)", perfection(997), "deficient")
check.expect("Smallest Odd Abundant Number", perfection(945),
"abundant")
check.expect("Almost Perfect 1", perfection(16), "deficient")
check.expect("Almost Perfect 2", perfection(32), "deficient")
check.expect("Almost Perfect 3", perfection(64), "deficient")
check.expect("Almost Perfect 4", perfection(128), "deficient")
```